Software Architecture

Lecture Notes

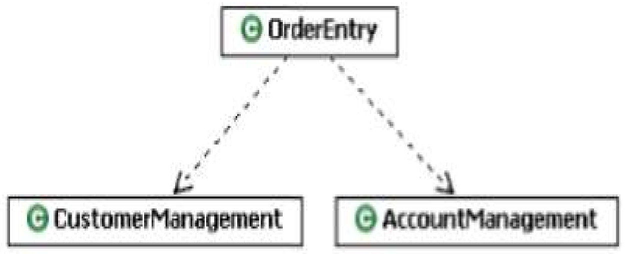
Software Design (SE324)

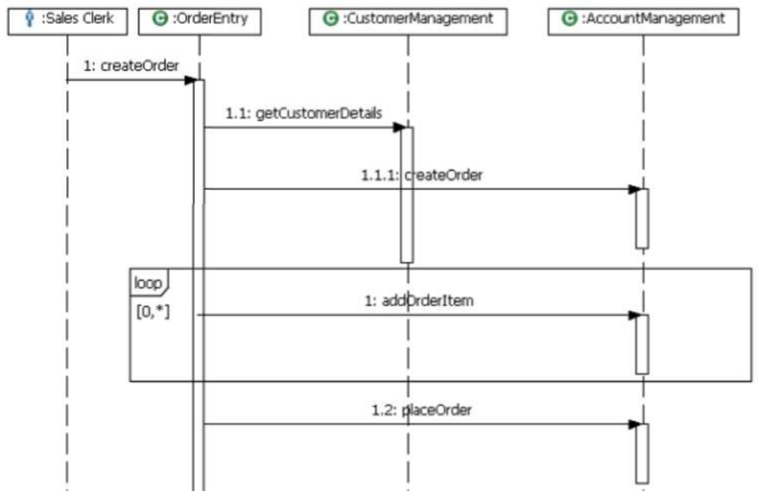
(Prepared by Dr. Khaldoon Al-Zoubi)

Software Architecture

*Architecture is the fundamental **organization** of a **system** embodied in its **components**, their **relationships** to each other, and to the **environment**, and the principles guiding its design and evolution. [IEEE 1471]*

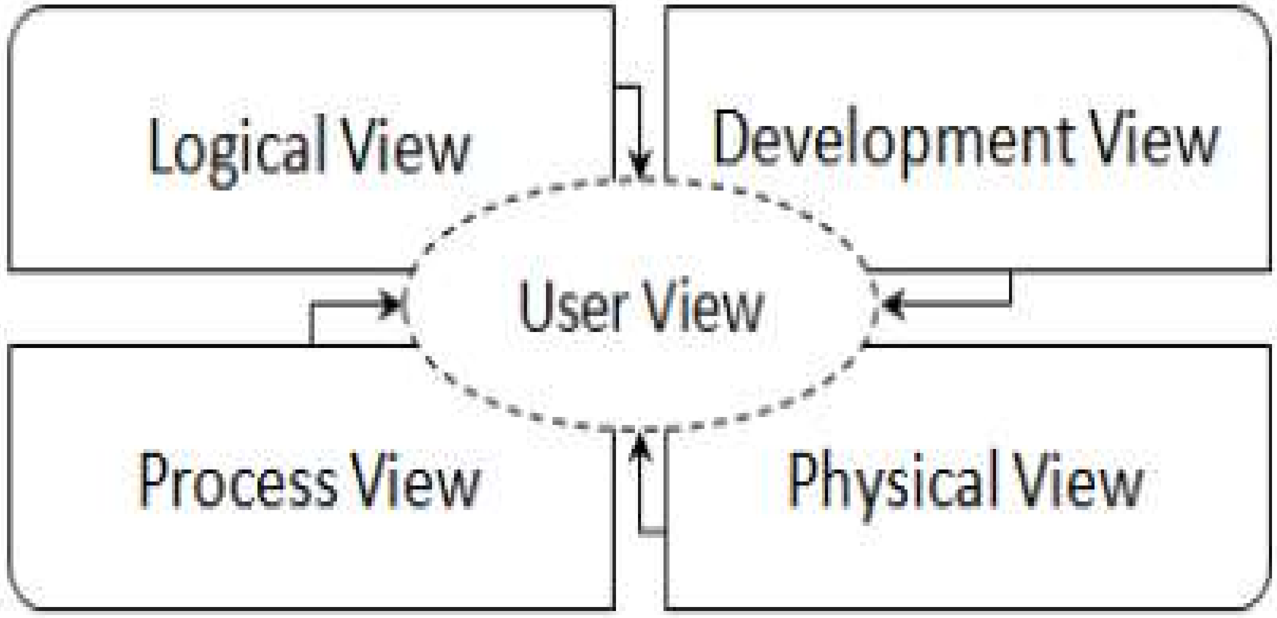
- An architecture defines structure
- An architecture defines behavior
- An architecture focuses on significant elements
- An architecture may conform to an architectural style/Pattern
- An architecture influences team structure





4 + 1 view model of software architecture

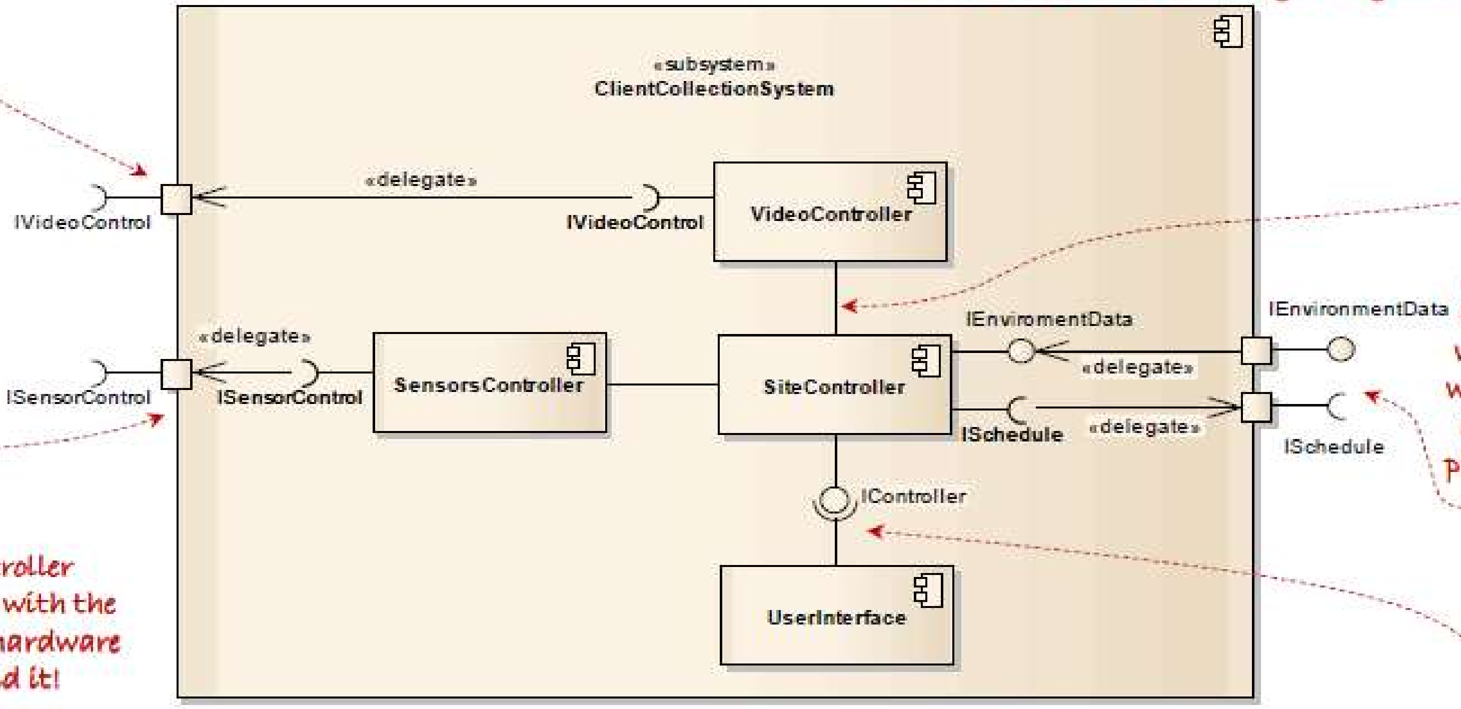
- A logical view → Structure (Major elements in the structure)
- A process view → Dynamic Behavior (Interaction at run time)
- A development(implementation) view → written Code Organization
- A physical view → deployment/installation over hardware devices
- Requirement (User View) (+1)



Logical View
(Decompose System into Components)

Video controller communicates with the actual video collection hardware to command it!

No interface defined! Direct association suggests that these components are tightly coupled, therefore making it more difficult to modify the system later on!



SiteController communicates with the outside world. It requires a schedule and provides collected data.

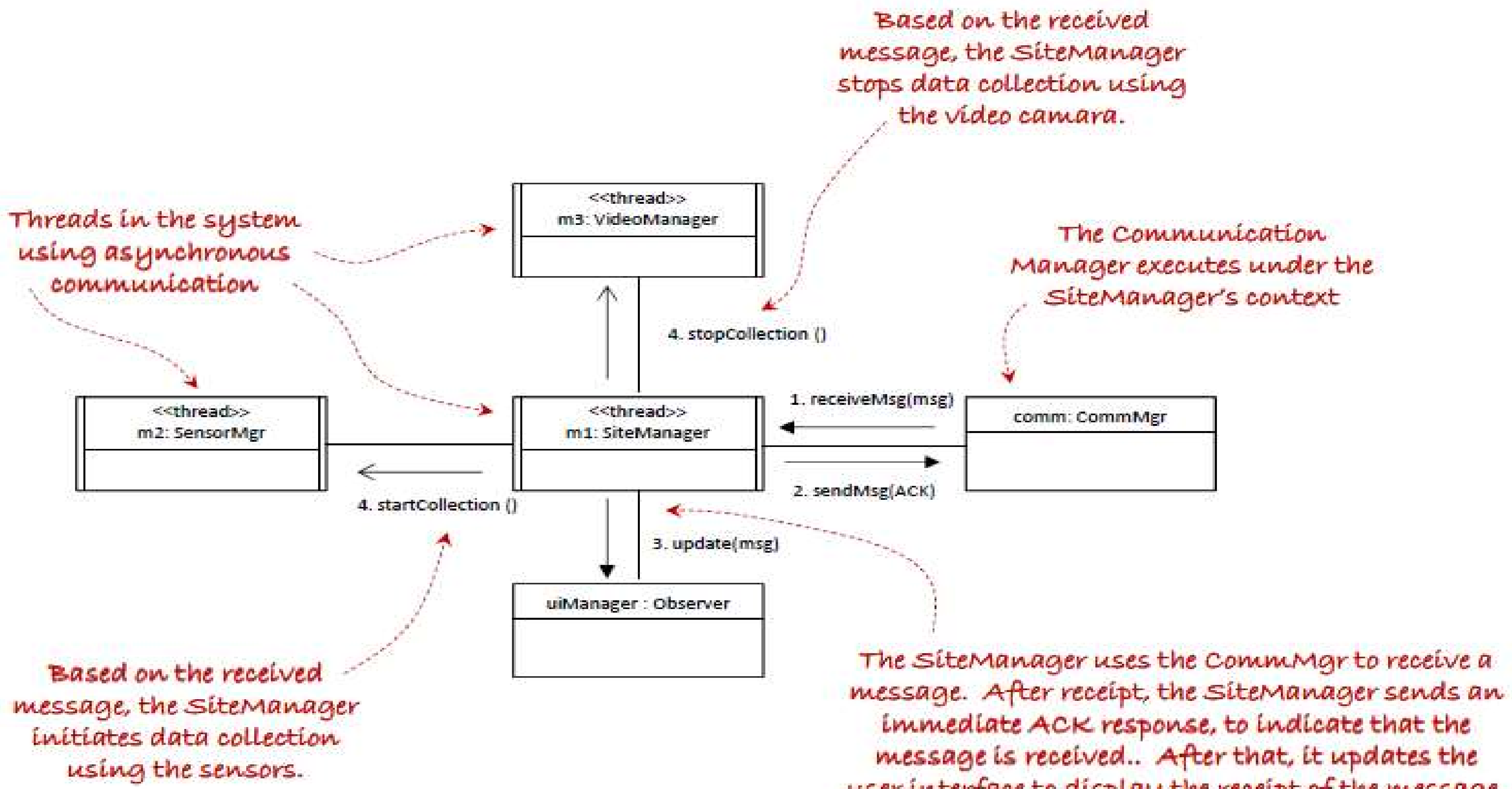
Sensor controller communicates with the actual Sensor hardware to command it!

Important:
This process is recursive, i.e., each component can be further decomposed!

Well-defined interface! This allows different user interfaces to be exchangeable easier, e.g., from console-to graphical user interface-based. on!

Modifiability	A goal that seeks to minimize the degree of complexity involved when changing the system to fit current or future needs

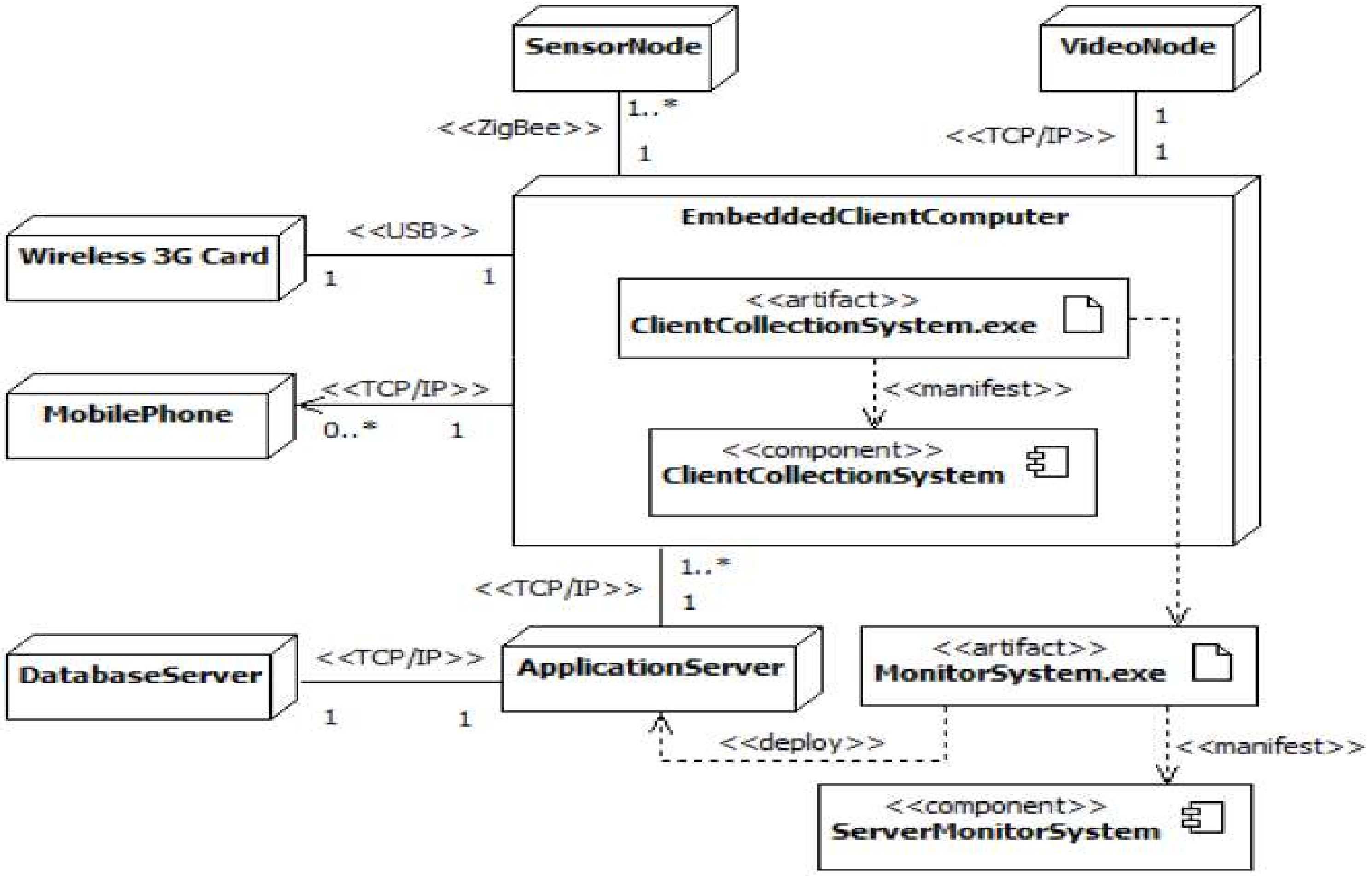
Process view
(Dynamic aspects of the system)


Based on the received message, the SiteManager stops data collection using the video camara.
Threads in the system using asynchronous communication
The Communication Manager executes under the SiteManager's context
<<thread>> m3: VideoManager
4. stopCollection ()
<<thread>> m2: SensorMgr
<<thread>> m1: SiteManager
1. receiveMsg(msg)
comm: CommMgr
2. sendMsg(ACK)
4. startCollection ()
3. update(msg)
uiManager : Observer
Based on the received message, the SiteManager initiates data collection using the sensors.
The SiteManager uses the CommMgr to receive a message. After receipt, the SiteManager sends an immediate ACK response, to indicate that the message is received.. After that, it updates the user interface to display the receipt of the message.

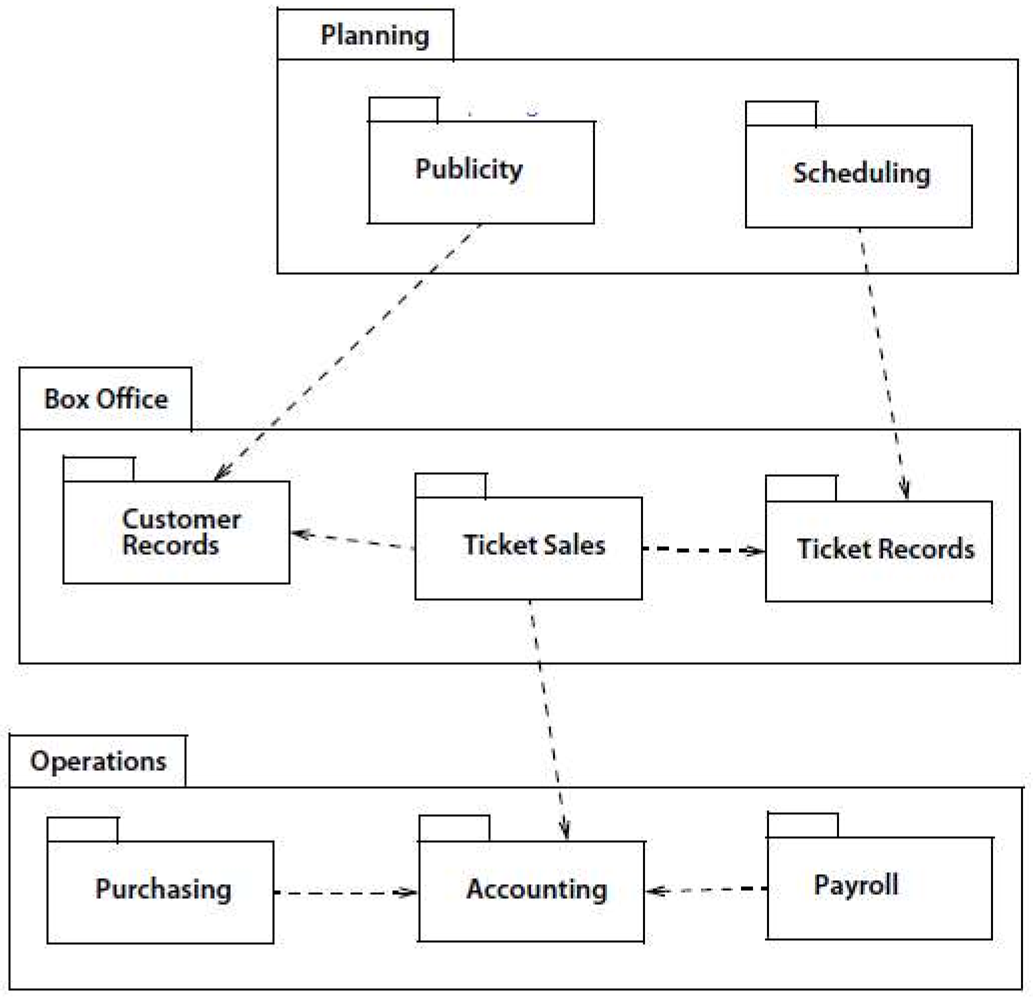
Performance	A goal that seeks to maximize the system's capacity to accomplish useful work under time and resource constraints

Physical view

- A physical view, which shows the system hardware and how software components are distributed across the processors in the system.



Development/Implementation view (Software Organization)



Architectural Patterns

Design Patterns

- What is a design pattern?
  - General solution to a design problem that returns repeatedly in many projects
  - When mentioned usually means detailed design patterns
- Architectural Patterns or styles
  - Software Architecture level
- Detailed Design patterns
  - or just design patterns
  - Now we are talking implementations

First We should know Distributed systems

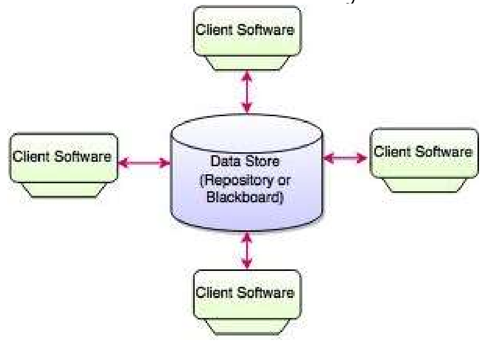
- Distributed systems are decomposed into multiple processes that (typically) collaborate through the network.
- Opposite of standalone/independent/local systems
  - No network
- Types of computing
  - Centralized Computing: all calculations are done on one particular computer (system). Example: you have a dedicated server for calculating data.
  - Distributed Computing: the calculation is distributed to multiple computers. Example: when you have a large amount of data then you can divide it and send each part to particular computers which will make the calculations for their part.

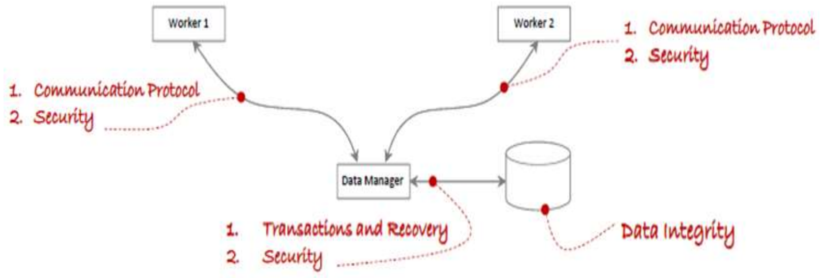
Architecture Systems (Styles)

- Data-Centered Systems
  - Example Pattern: Blackboard
- Data Flow Systems
  - Example Pattern: Pipes-and-Filters
- Interactive Systems
  - Example Pattern: Model-View-Controller (MVC)
- Service-based Systems
  - Example Pattern: Client Server, Broker

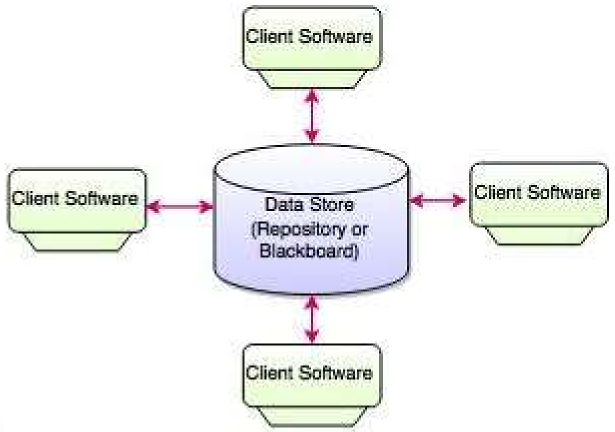
Data-Centered Architecture

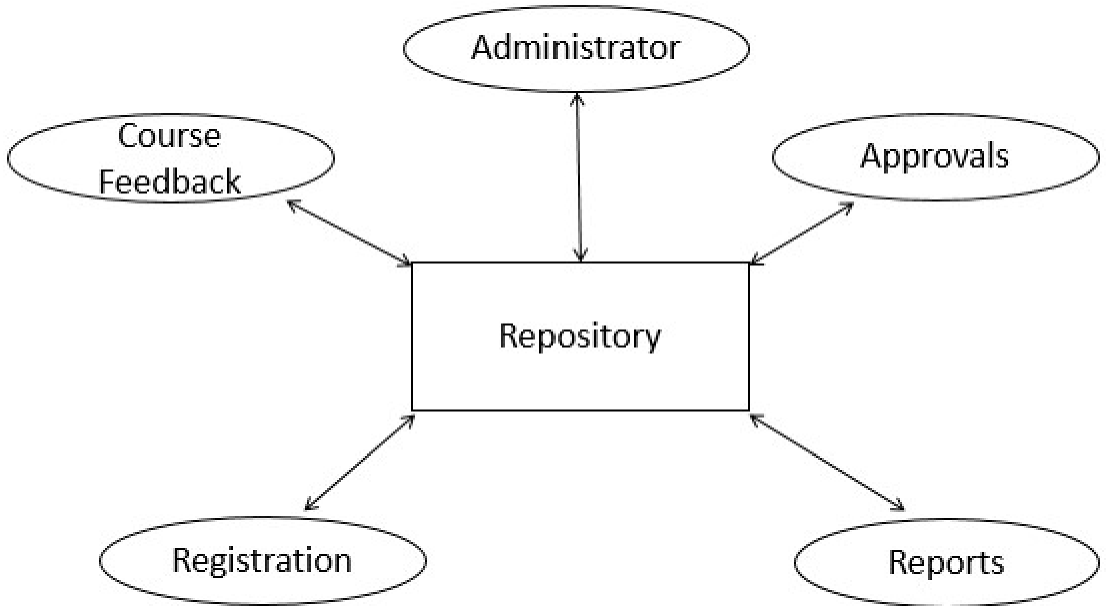
- In data-centered architecture, the data is centralized and accessed frequently by other components, which modify data.
- Data-centered architecture consists of different components that communicate through shared data repositories. The components access a shared data structure and are relatively independent, in that, they interact only through the data store.
- General Types of Components
  - **Central data** structure or **data store** or data **repository**, which is responsible for providing permanent data storage.
  - **Data accessor** (**workers**) or a collection of independent components that operate on the central data store, perform computations, and might put back the results.
- The flow of control differentiates the architecture into two categories
  - **Repository Architecture Style**
  - **Blackboard Architecture Style**
    - **We need also a Data Manager** component controls, provides, notifies, and manages access to the system's data.




Worker 1
Worker 2
1. Communication Protocol
2. Security
1. Communication Protocol
2. Security
Data Manager
1. Transactions and Recovery
2. Security
Data Integrity

Repository Architecture Style (Architecture Example of data-centered)



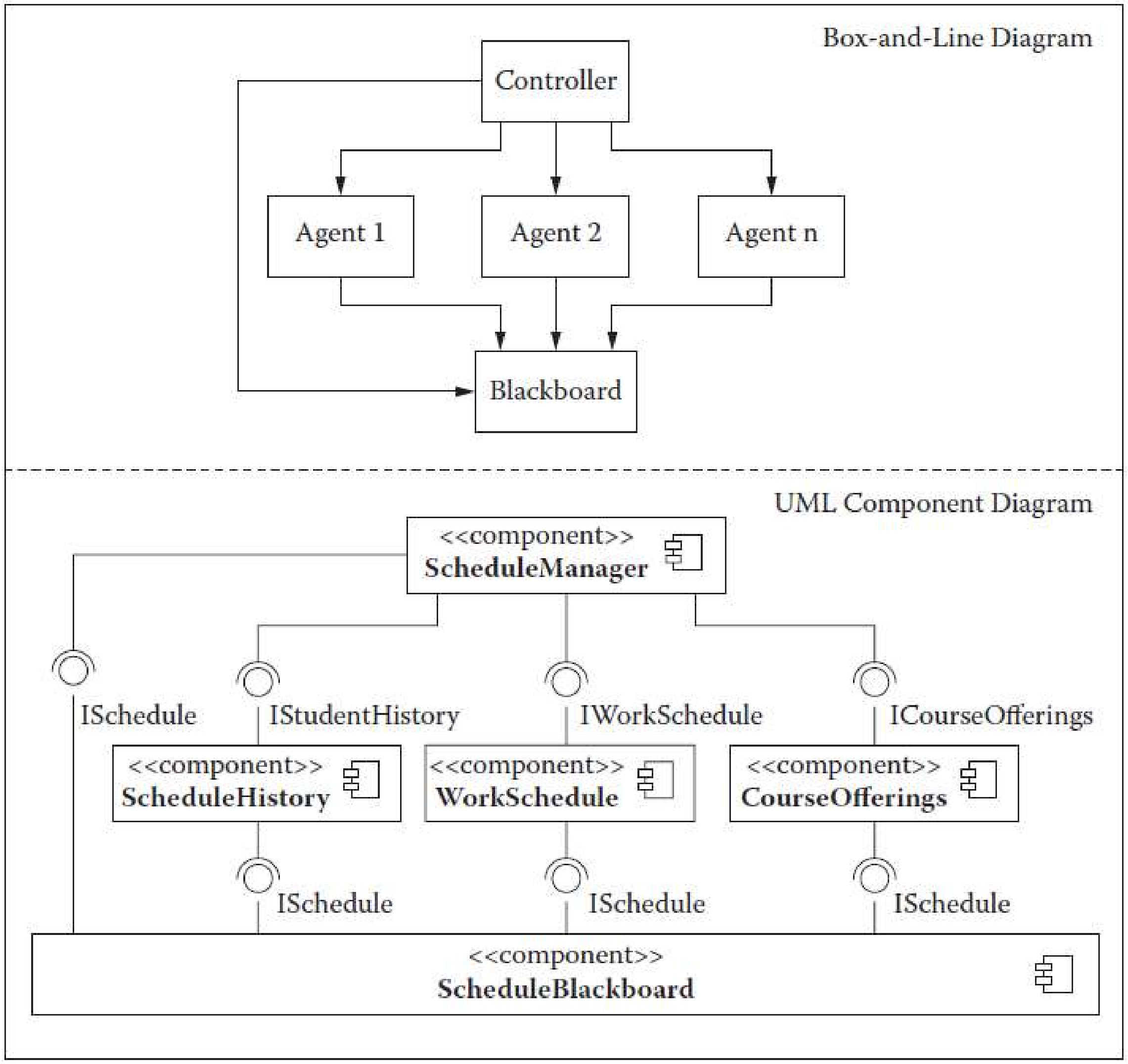


Blackboard Architecture (Architecture Example of data-centered)

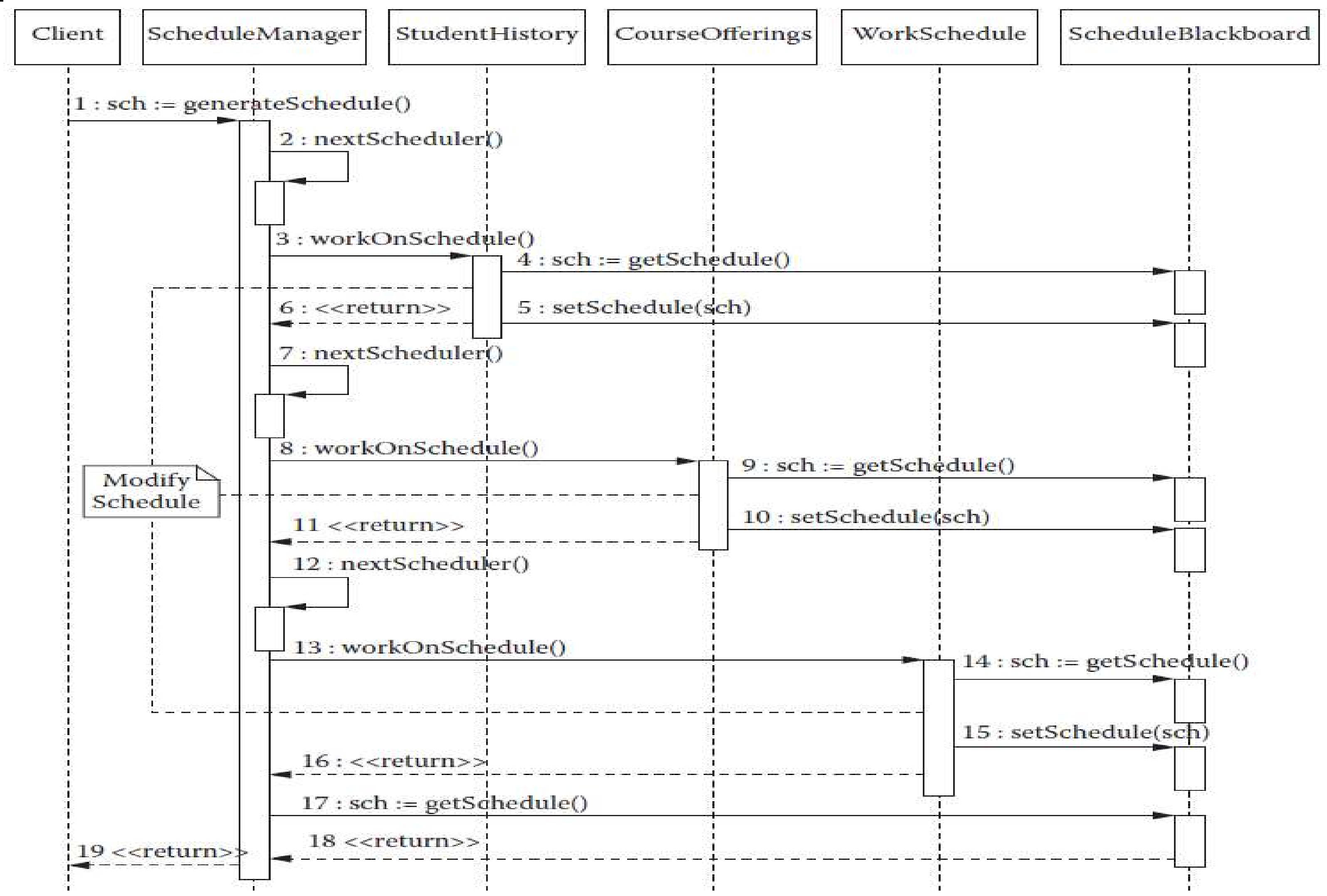
- The blackboard architectural pattern decomposes software systems into components that work around a central data component to provide solutions to complex problems. These components work independently of each other to provide partial solutions to problems using an opportunistic problem-solving approach. That is, there are no predetermined, or correct, sequences of operations for reaching the problem's solution. Each component provides solutions that build upon the problem's current state, which is defined by the collective set of solutions provided by the blackboard's components. The access to the central data component can be made through direct memory reference, procedure calls, or database query
- Example: Consider the application of the blackboard architectural pattern for the design of a university software system that manages student registrations. Registrations are managed based on course availability, students' course history, and students' work schedule

Blackboard Architecture (Example of data-centered)

- Components:
  - ScheduleManager Manages access to the ScheduleBlackboard and controls the problem-solving process
  - StudentHistory Knowledge source for students' history, including course and other preferences
  - WorkSchedule Knowledge source for students' work schedule
  - CourseOfferings Knowledge source for university's course offerings
  - ScheduleBlackboard Central data store where elements of the solution space are stored
- Quality Properties of the Blackboard Architectural Pattern
  - Modifiability: Agents are compartmentalized and independent from each other; therefore, it is easy to add or remove agents to fit new systems
  - Reusability: Specialized components can be reused easily in other applications
  - Maintainability: Allows for separation of concerns and independence of the knowledge-based agents

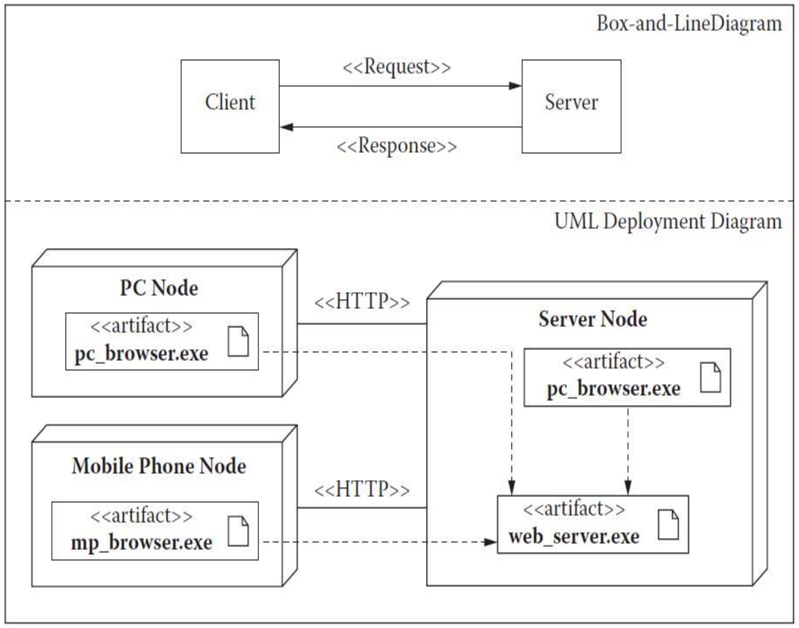


Blackboard Architecture Example (Process View)



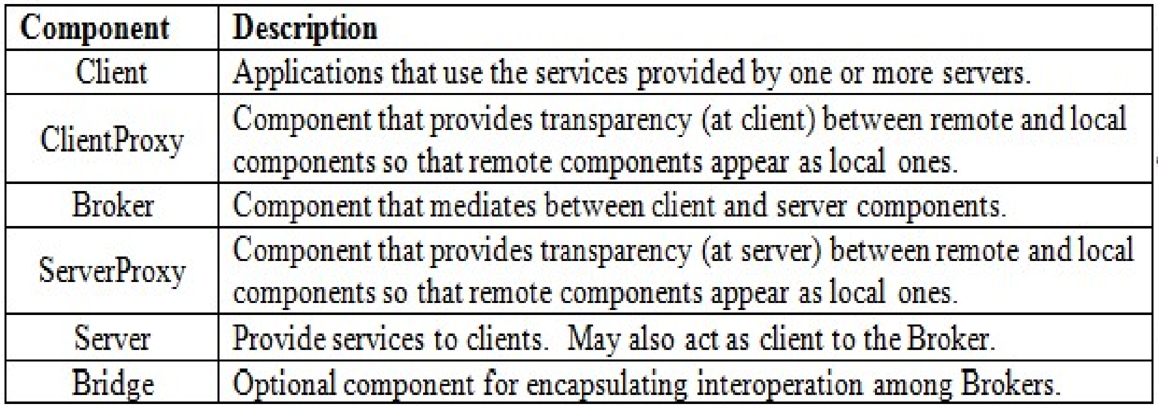
Client-server

- The client–server architecture decomposes software systems into two main components: the client and the server. These components are manifested as individual processes that can be distributed over the network or within a single node. Client–server systems are not determined merely by separating processes or by distributing processes across the network but by having one process, the client, depend on the services provided by another process, the server. The most pervasive example of a client–server system today includes the web browser client and the web server.
  - Web-services = access services via the Web (HTTP)
- Qualities of the Client–Server Architectural Pattern
- Interoperability: Allows clients on different platforms to interoperate with servers of different platforms
  - Modifiability: Allows for centralized changes in the server and quick distribution among many clients
  - Availability: By separating server data, multiple server nodes can be connected as backup to increase the server data or services' availability
  - Reusability: By separating server from clients, services or data provided by the server can be reused in different applications

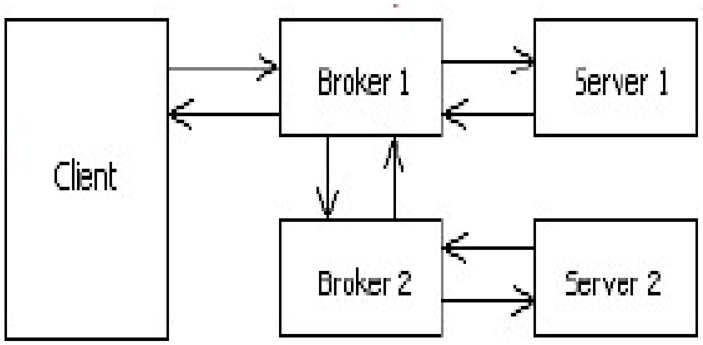


Broker Pattern (Improvement of Client-Server Pattern)

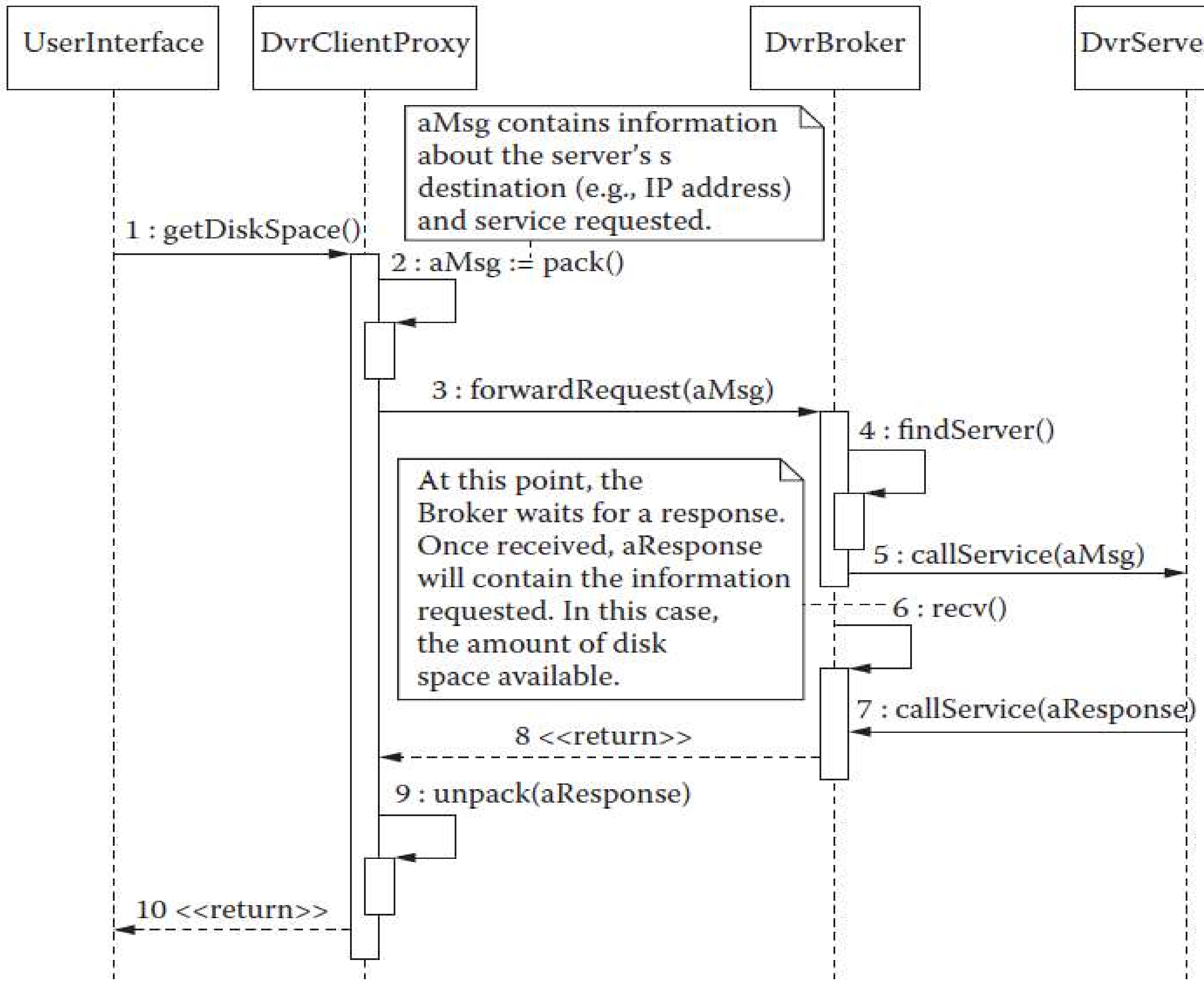
- The Broker architectural pattern provides mechanisms for achieving better flexibility between clients and servers in a distributed environment.
  - This leads to complexity for systems that need to provide services from multiple servers hosted at different locations.
    - In a typical client-server system, clients are tightly coupled with servers.
    - Clients needs to find those servers, Clients need to know how to access services
  - In some software systems, clients (in a distributed environment) need to be able to access services from multiple servers without known their actual locations or particular details of communication.
    - When these concerns are separated, it leads to systems that are flexible and interoperable.
- Broker pattern: decreases coupling between client and servers to improve access to services of multiple servers



Component	Description
Client	Applications that use the services provided by one or more servers.
ClientProxy	Component that provides transparency (at client) between remote and local components so that remote components appear as local ones.
Broker	Component that mediates between client and server components.
ServerProxy	Component that provides transparency (at server) between remote and local components so that remote components appear as local ones.
Server	Provide services to clients. May also act as client to the Broker.
Bridge	Optional component for encapsulating interoperation among Brokers.

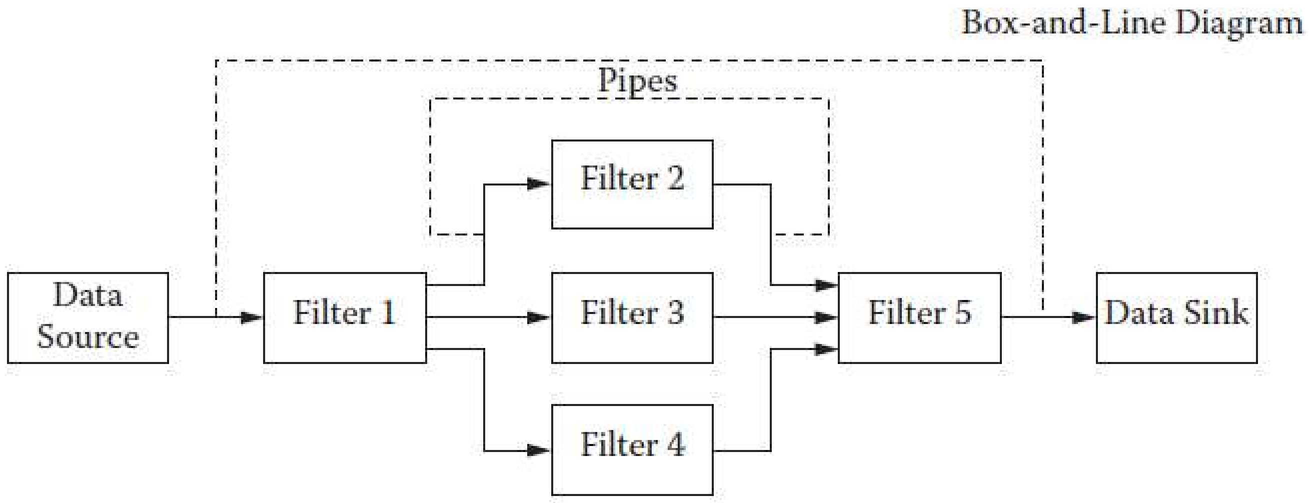


Broker Pattern Architecture Example

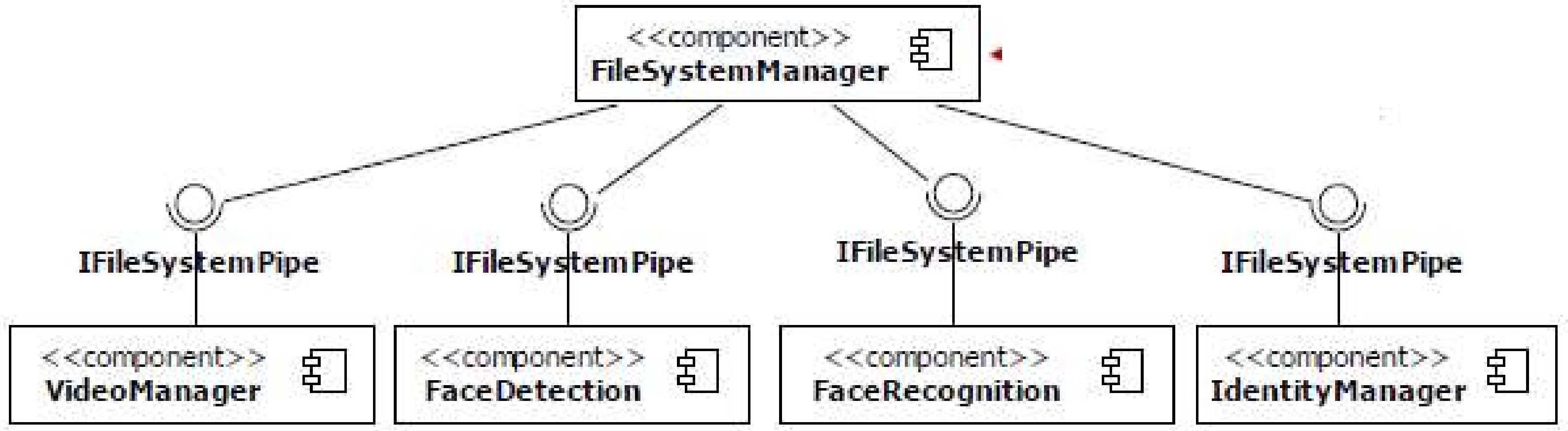

UserInterface
DvrClientProxy
DvrBroker
DvrServerProxy
aMsg contains information about the server's s destination (e.g., IP address) and service requested.
1 : getDiskSpace()
2 : aMsg := pack()
3 : forwardRequest(aMsg)
4 : findServer()
At this point, the Broker waits for a response. Once received, aResponse will contain the information requested. In this case, the amount of disk space available.
5 : callService(aMsg)
6 : recv()
7 : callService(aResponse)
8 <<return>>
9 : unpack(aResponse)
10 <<return>>

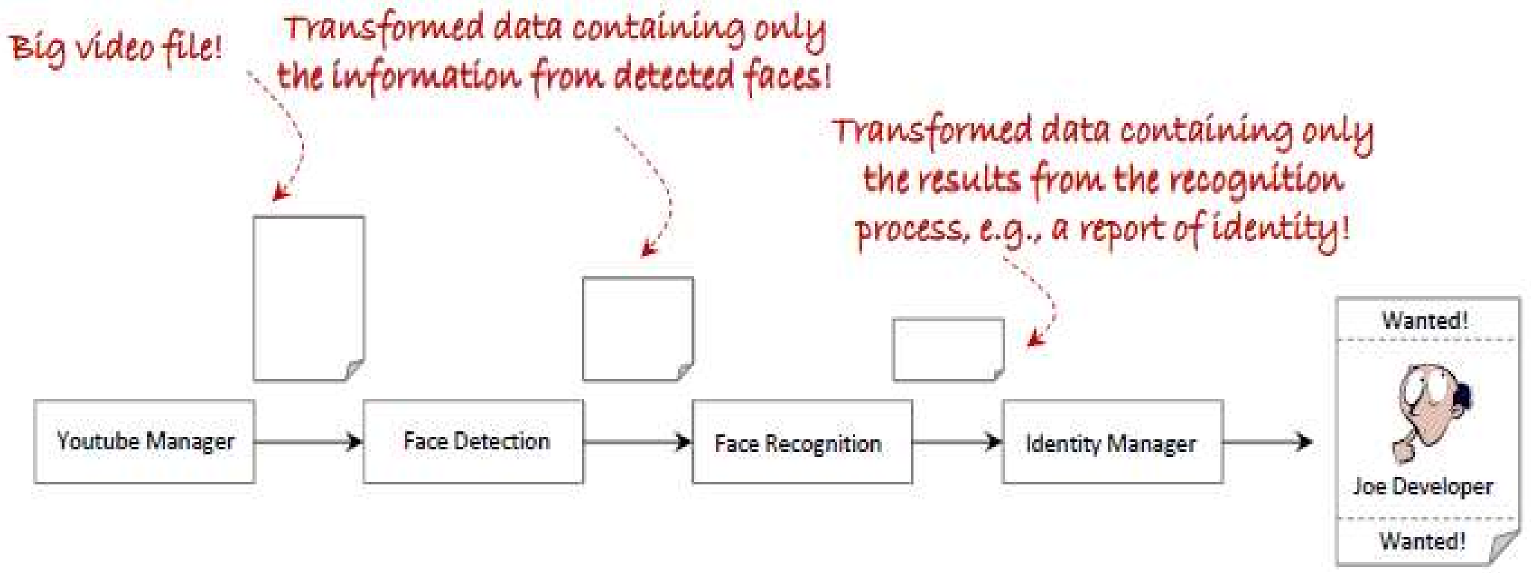
Data flow (pipe and filter pattern)

- **Data flow** systems are decomposed around the central theme of transporting data (or data streams) and transforming the data along the way to meet application-specific requirements.
  - ✓ Typical responsibilities found in components of data-flow systems include:
    - Worker components, those that perform work on data
    - Transport components, those that transporting data
- Pattern Example: **Pipes-and-Filters** is composed of the following components:
  - ✓ **Data source**: Produces the data
  - ✓ **Filter: Processes**, enhances, modifies, etc. the data
  - ✓ **Pipes**: Provide connections between data source and filter, filter to filter, and filter to data sink.
  - ✓ **Data Sink**: Data consumer



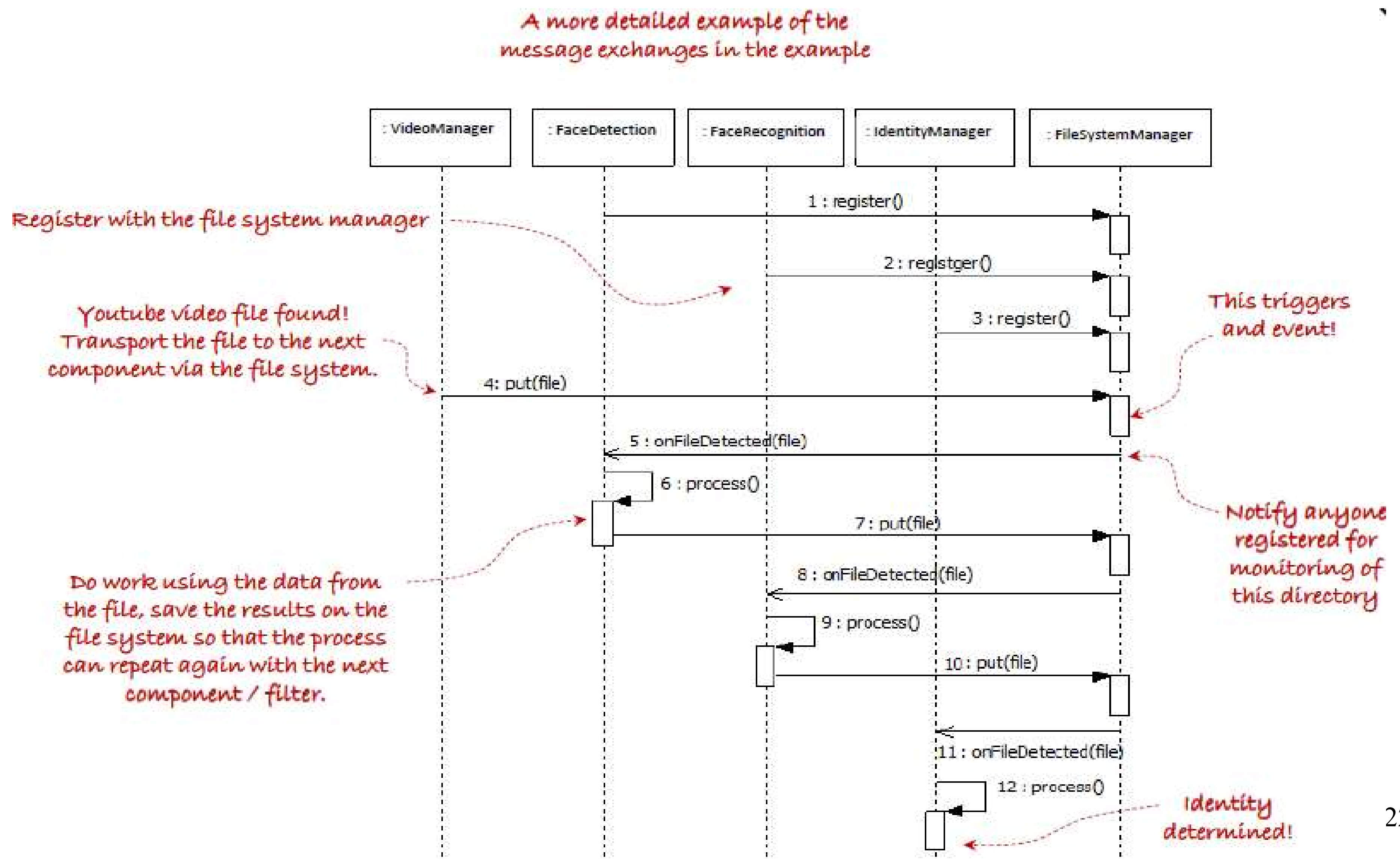
Pipe and Filter pattern Example-1





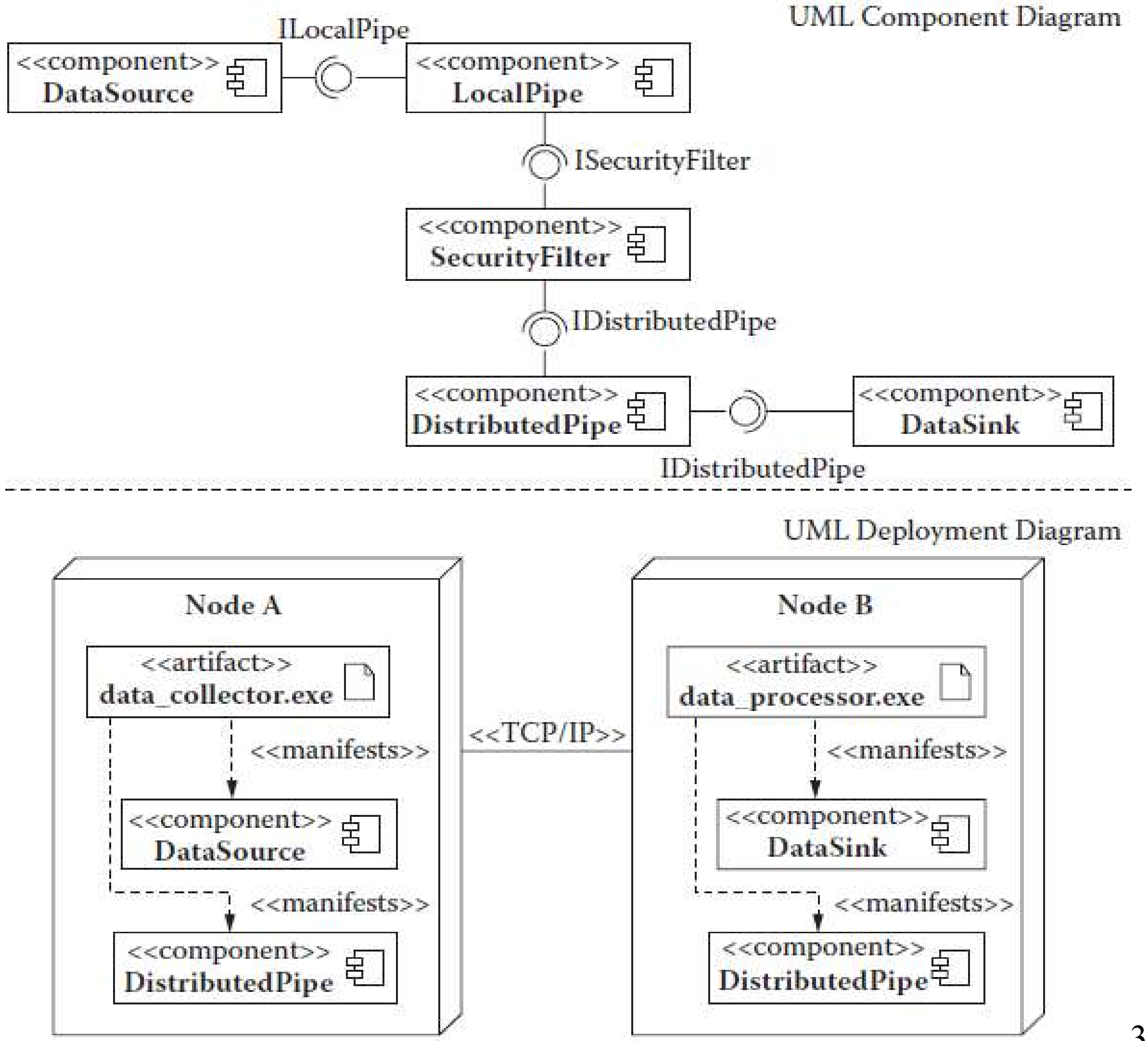
Big video file!
Transformed data containing only the information from detected faces!
Transformed data containing only the results from the recognition process, e.g., a report of identity!
Youtube Manager
Face Detection
Face Recognition
Identity Manager
Wanted!
Joe Developer
Wanted!

Pipe and Filter pattern Example-1 (Process View)


A more detailed example of the message exchanges in the example
: VideoManager
: FaceDetection
: FaceRecognition
: IdentityManager
: FileSystemManager
1 : register()
2 : reglstger()
3 : register()
4: put(file)
5 : onFileDetected(file)
6 : process()
7 : put(file)
8 : onFileDetected(file)
9 : process()
10 : put(file)
11 : onFileDetected(file)
12 : process()
Register with the file system manager
Youtube video file found! Transport the file to the next component via the file system.
This triggers and event!
Notify anyone registered for monitoring of this directory
Do work using the data from the file, save the results on the file system so that the process can repeat again with the next component / filter.
Identity determined!

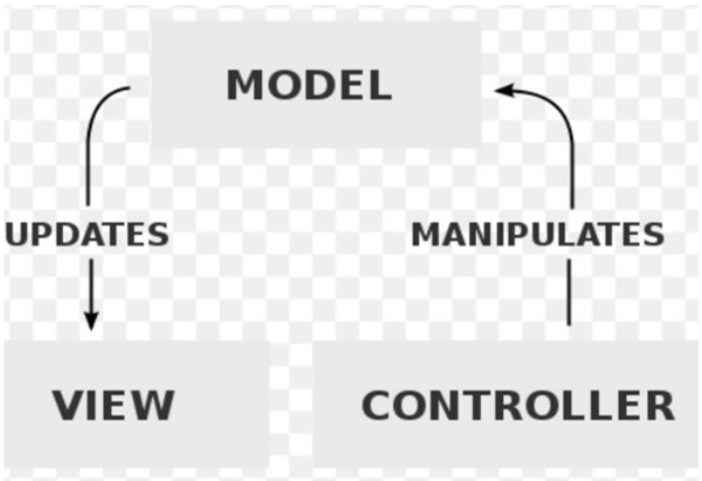
Pipe and Filter pattern Example-2

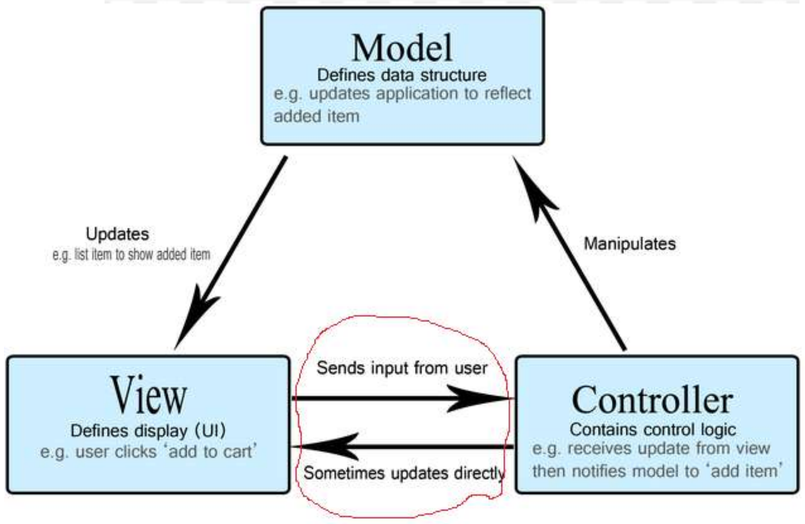
- Problem example: Consider the design of a signal processing system that operates on real-time video or audio streams and is designed to collect data, encrypt it, and forward it for further processing.
- Components:
  - Data source Produces real-time video or audio streams
  - Local pipe Mechanism for transferring video or audio data streams locally from data source to security filter
  - Security filter Transforms data streams by encrypting flow of data
  - Distributed pipe Mechanism for transferring encrypted data streams wirelessly (e.g., Satellite Communications [SATCOM]) from local site to remote site; provides the logical connection between data source and data sink
  - Data sink Destination component where data streams are stored for later review



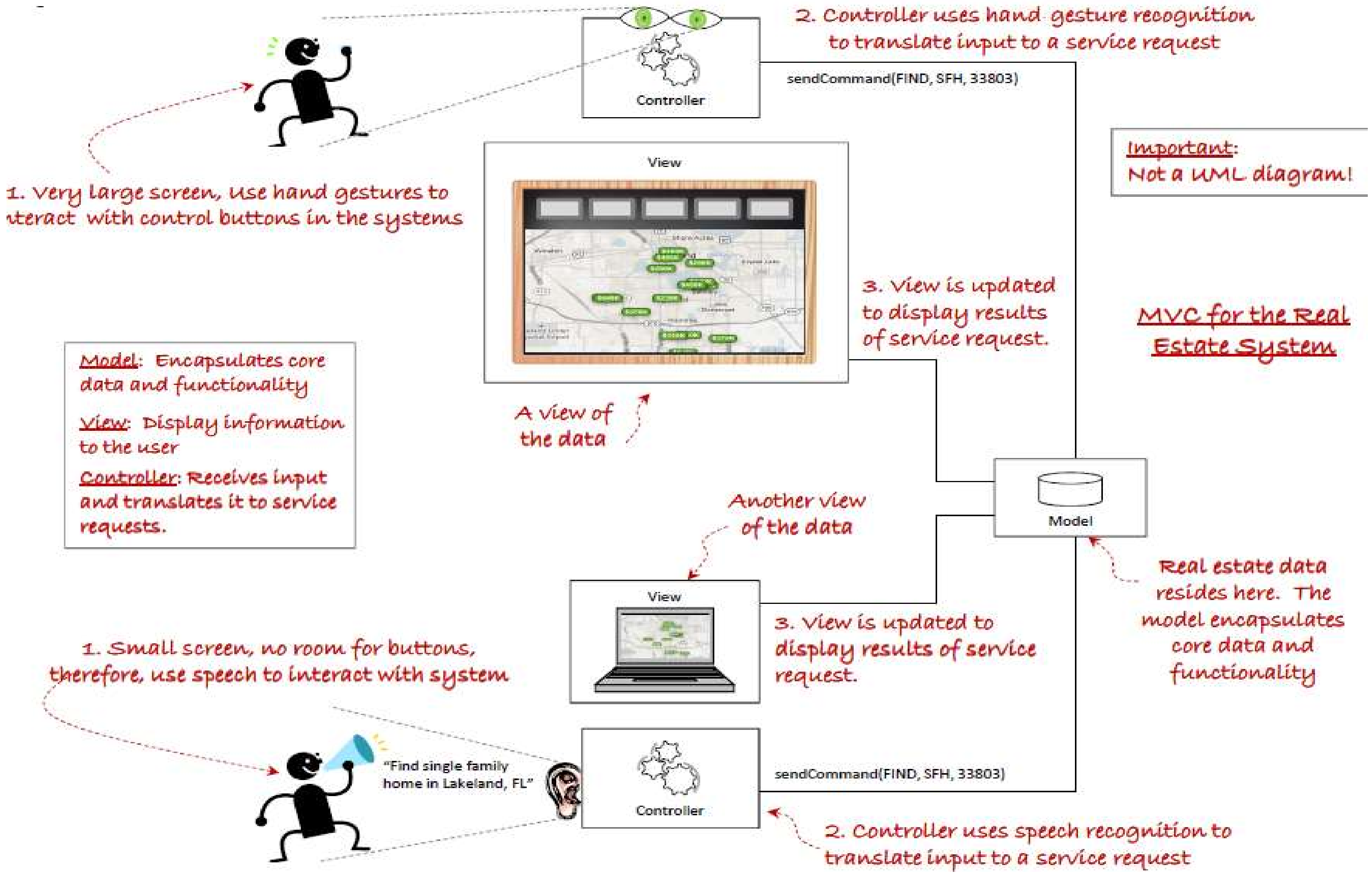
Interactive Systems
Model-View-Controller (MVC) pattern

- The MVC architectural pattern is used in interactive applications that require flexible incorporation of human-computer interfaces. With the MVC, systems are decomposed into three main components that handle independently the system's input, processing, and output.
- MVC Components
  - The **model** directly manages the data, logic, and rules of the application.
  - A **view** can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
  - The **controller**, accepts input and converts it to commands for the model or view



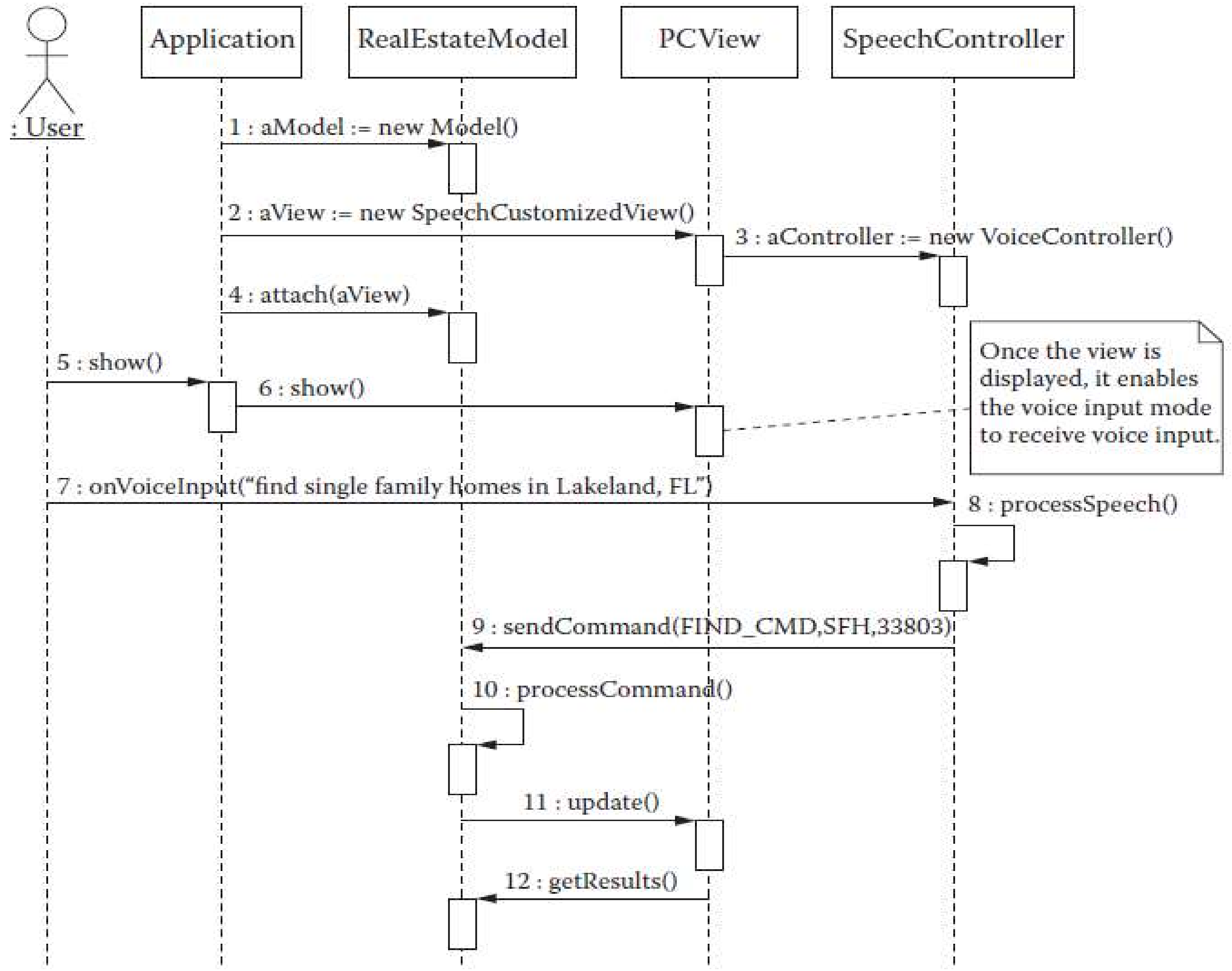

Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

MVC Example



2. Controller uses hand gesture recognition to translate input to a service request
Controller
sendCommand(FIND, SFH, 33803)
Important: Not a UML diagram!
View
1. Very large screen, use hand gestures to nteract with control buttons in the systems
3. View is updated to display results of service request.
MVC for the Real Estate System
Model: Encapsulates core data and functionality
View: Display information to the user
Controller: Receives input and translates it to service requests.
A view of the data
Model
Another view of the data
Real estate data resides here. The model encapsulates core data and functionality
View
3. View is updated to display results of service request.
1. Small screen, no room for buttons, therefore, use speech to interact with system
"Find single family home in Lakeland, FL"
Controller
sendCommand(FIND, SFH, 33803)
2. Controller uses speech recognition to translate input to a service request

MVC Example


: User
Application
RealEstateModel
PCView
SpeechController
1 : aModel := new Model()
2 : aView := new SpeechCustomizedView()
3 : aController := new VoiceController()
4 : attach(aView)
5 : show()
6 : show()
Once the view is displayed, it enables the voice input mode to receive voice input.
7 : onVoiceInput("find single family homes in Lakeland, FL")
8 : processSpeech()
9 : sendCommand(FIND_CMD,SFH,33803)
10 : processCommand()
11 : update()
12 : getResults()