INGI2359 – Séminaire de génie logiciel : Architectural Patterns

Command Processor
View Handler
Forwarder-Receiver
Client-Dispatcher-Server
Publisher-Subscriber
Patterns Systems

Pierre Bodson and Laurent Pierson

UCL 2008-2009

- Command Processor
- View Handler
- Forwarder-Receiver
- Client-Dispatcher-Server
- Publisher-Subscriber
- Pattern Systems

Outline

- **Command Processor**
- View Handler
- Forwarder-Receiver
- Client-Dispatcher-Server
- Publisher-Subscriber
- Pattern Systems

Command Processor

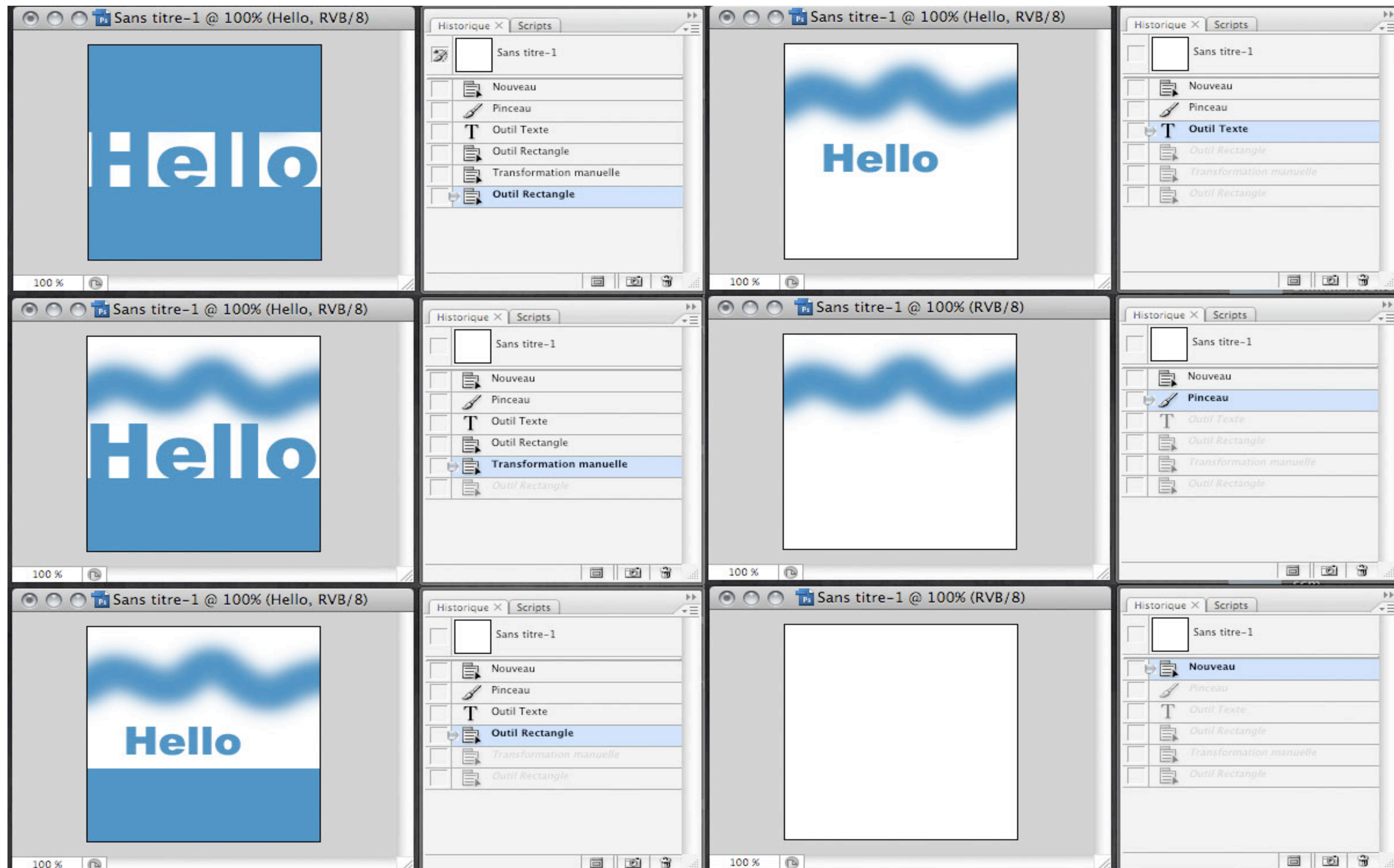
- Goal
  - Separate request for a service from its execution
  - Manage requests as separate objects, schedule their execution and store them for undo operations

- Applicability
  - Applications that need flexible and extensible user interfaces or that provide services related to the execution of user functions (undo, macros, logging,…)

  - Example : Multiple undo operations in Photoshop

Command Processor

- Example



Command Processor

- Components
  - Abstract command
    - Defines a uniform interface of all commands objects
    - At least has a procedure to execute a command
    - May have other procedures for additional services as undo, logging,…

Command Processor

- Components
  - Command
    - Implements interface of abstract command
    - Encapsulates a function request
    - Uses suppliers to perform requests
    - E.g. undo in text editor : save text + cursor position

Command Processor

- Components
  - Controller
    - Represents the interface to the application
    - Accepts service requests (e.g. bold text) and translates them into command objects
    - Transfer commands to command processor for execution

Command Processor

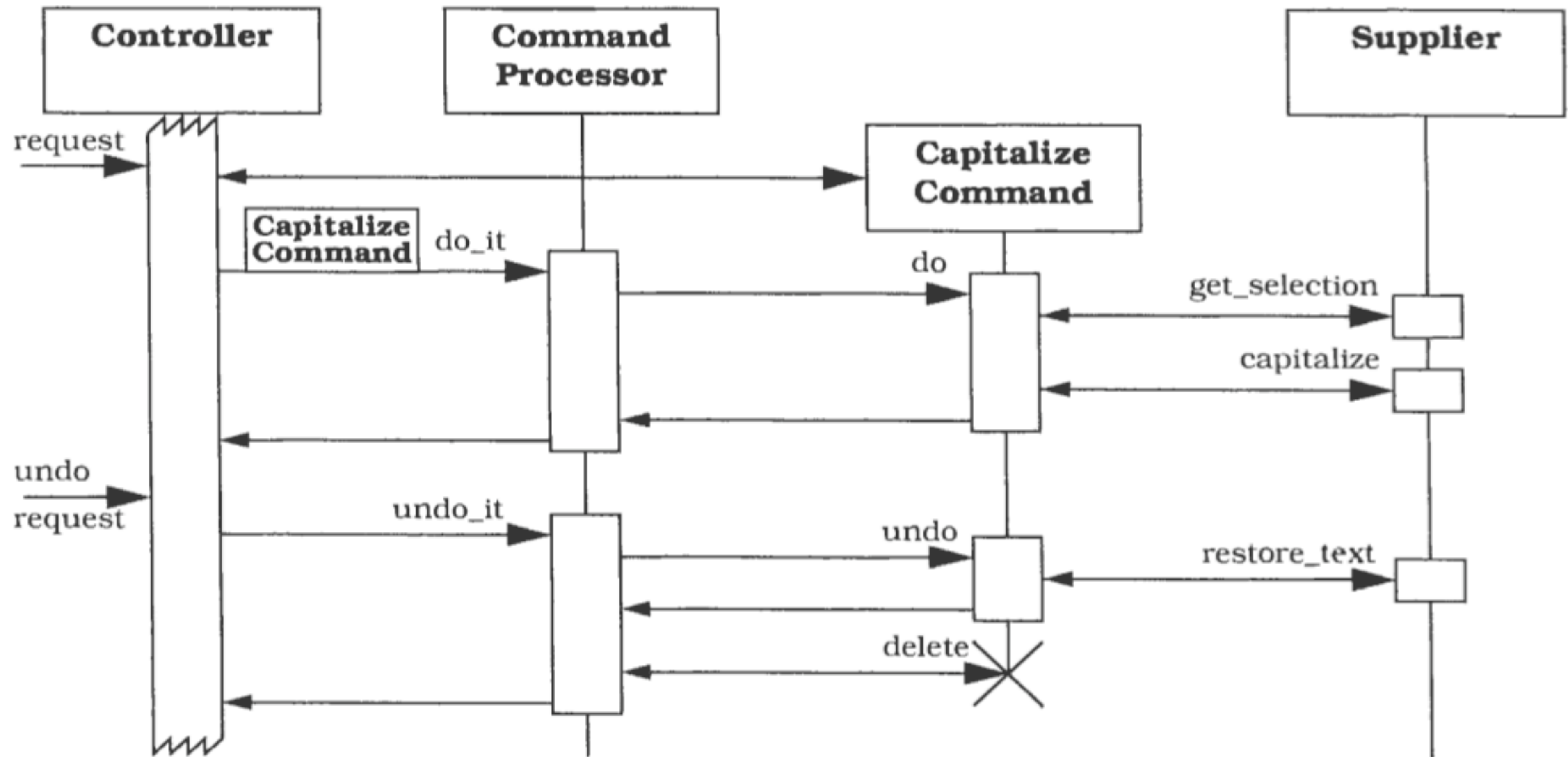
- Components
  - Command processor
    - Manages command objects (schedule, start execution)
    - Key component that implements additional services (e.g. stores commands for later undo)
    - Remains independent of specific commands (uses abstract command interface)

Command Processor

- Components
  - Supplier
    - Provides functionality required to execute concrete commands
    - Related commands often share suppliers
    - E.g. undo : supplier has to provide a means to save and restore its internal state

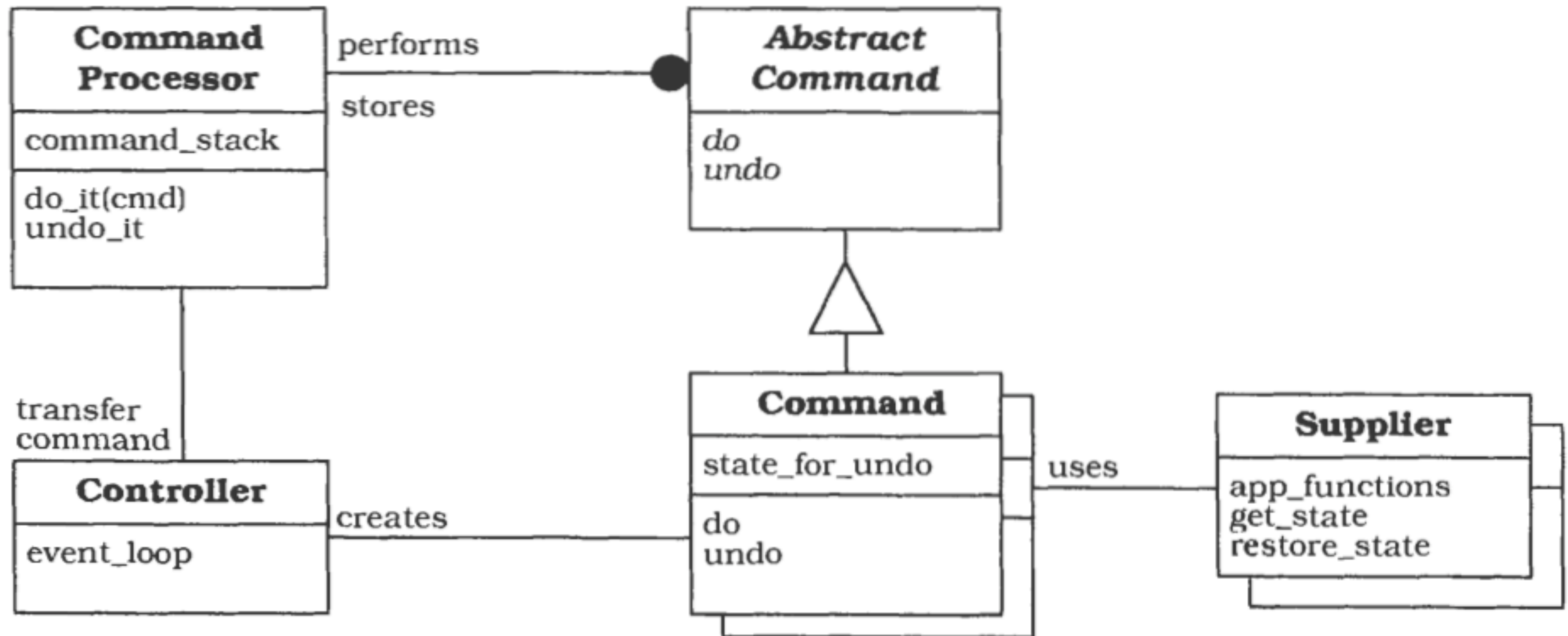
Command Processor

- Interaction protocol



Command Processor

- Component structure and inter-relationships



Command Processor

- Strengths
  - Flexibility in the way requests are activated
    - Different requests can generate the same kind of command object (e.g. use GUI or keyboard shortcuts)
  - Flexibility in the number and functionality of requests
    - Controller and command processor implemented independently of functionality of individual commands
    - Easy to change implementation of commands or to introduce new ones

Command Processor

- Strengths
  - Programming execution-related services
    - Command processor can easily add services like logging, scheduling,…
  - Testability at application level
    - Regression tests written in scripting language
  - Concurrency
    - Commands can be executed in separate threads
    - Responsiveness improved but need for synchronization

Command Processor

- Weaknesses
  - Efficiency loss
  - Potential for an excessive number of command classes
    - Application with rich functionality may lead to many command classes
    - Can be handled by grouping, unifying simple commands
  - Complexity in acquiring command parameters

Command Processor

- Variants
  - Spread controller functionality
    - Role of controller distributed over several components (e.g. each menu button creates a command object)
  - Combination with Interpreter pattern
    - Scripting language provides programmable interface
    - Parser component of script interpreter takes role of controller

Command Processor

- Known uses
  - ET++
    - Framework of command processors
    - Support unlimited, bounded and single undo and redo
  - Objectiver
  - Microsoft Word
  - Adobe Photoshop

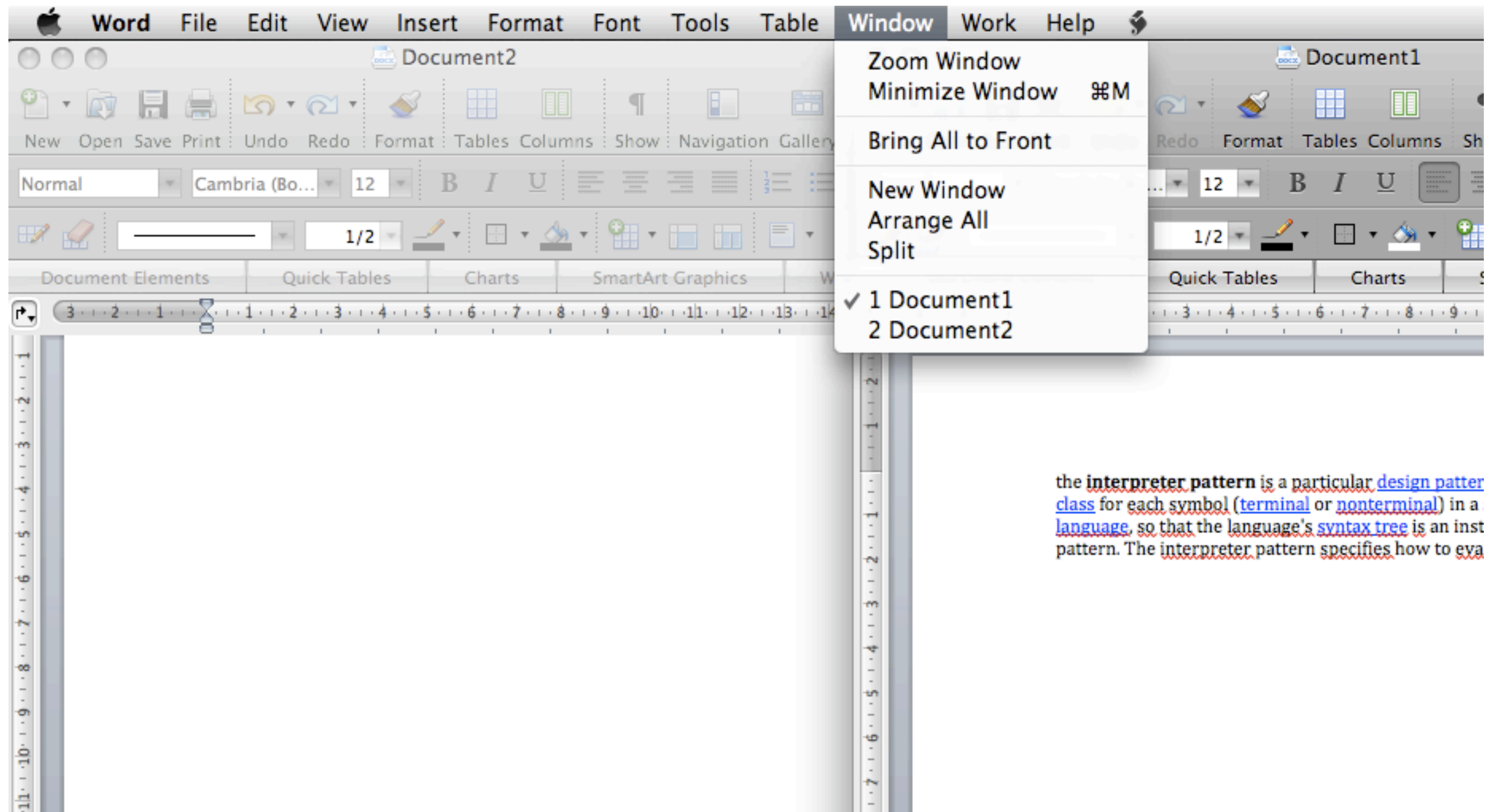
Outline

- Command Processor
- **View Handler**
- Forwarder-Receiver
- Client-Dispatcher-Server
- Publisher-Subscriber
- Pattern Systems

View Handler

- Goals
  - Help to manage all views that a software system provides
  - Allow clients to open, manipulate and dispose of views
  - Coordinate dependencies between views and organizes their update
- Applicability
  - Software system that provides multiple views of application-specific data, or that supports working with multiple documents
  - Example : Windows handler in Microsoft Word

View Handler



View Handler

- View Handler and other patterns
  - MVC
    - View Handler pattern is a refinement of the relationship between the model and its associated views.
  - PAC
    - Implements the coordination of multiple views according to the principles of the View Handler pattern.

View Handler

- Components
  - View Handler
    - Is responsible for opening new views, view initialization
    - Offers functions for closing views, both individual ones and all currently-open views
    - Offers view management services (e.g. bring to foreground, tile all view, clone views)
    - Coordinates views according to dependencies

View Handler

- Components
  - Abstract view
    - Defines common interface for all views
    - Used by the view handler : create, initialize, coordinate, close, etc.

View Handler

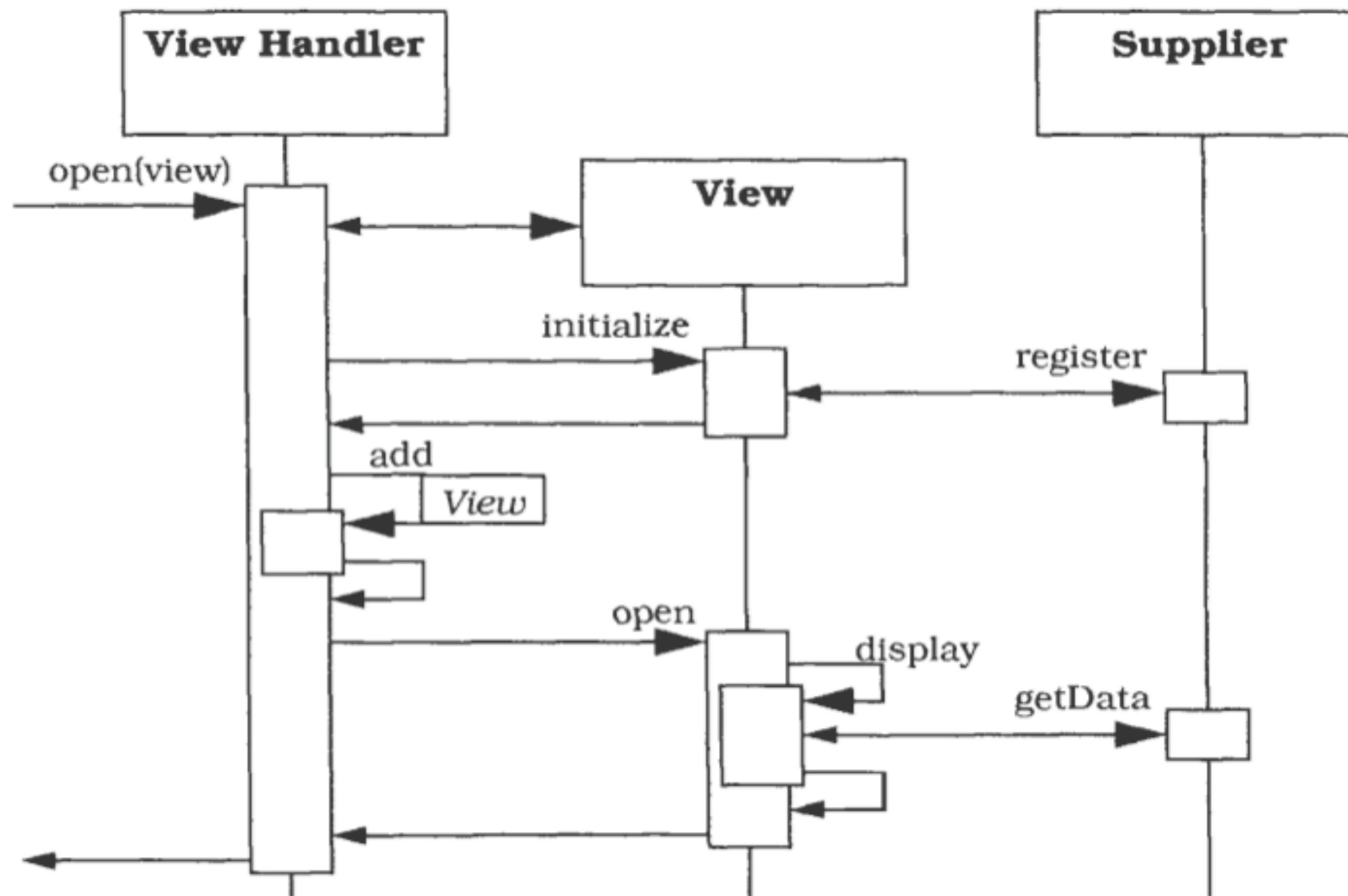
- Components
  - Specific view
    - Implements Abstract view interface
    - Knows how to display itself
      - Retrieves data from supplier(s)
      - Prepares data for display
      - Presents them to the user
    - Display function called when opening or updating a view

View Handler

- Components
  - Supplier
    - Provides the data that is displayed by the view components
    - Offers interface to retrieve or change data
    - Notifies dependent component about changes in data

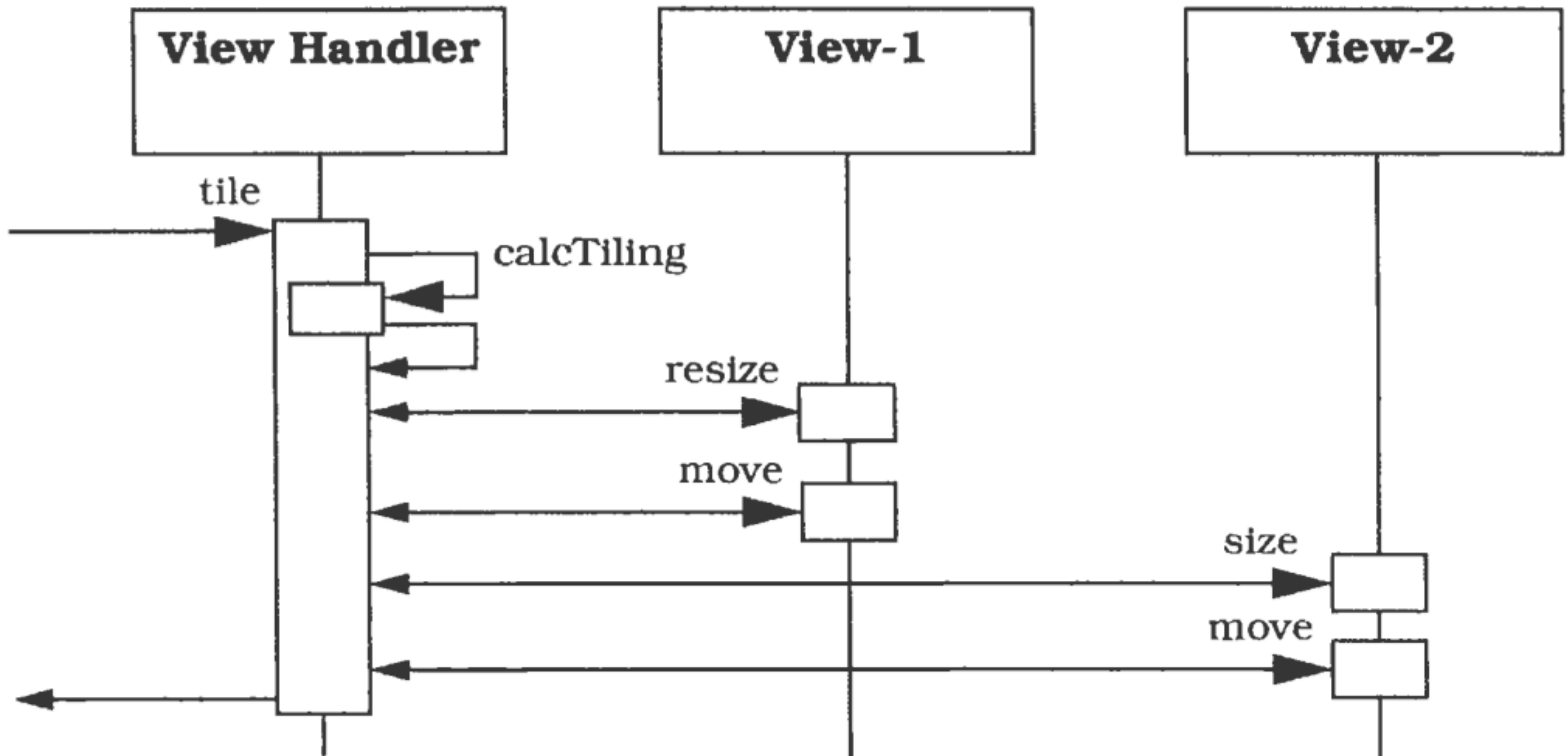
View Handler

- Interaction protocol



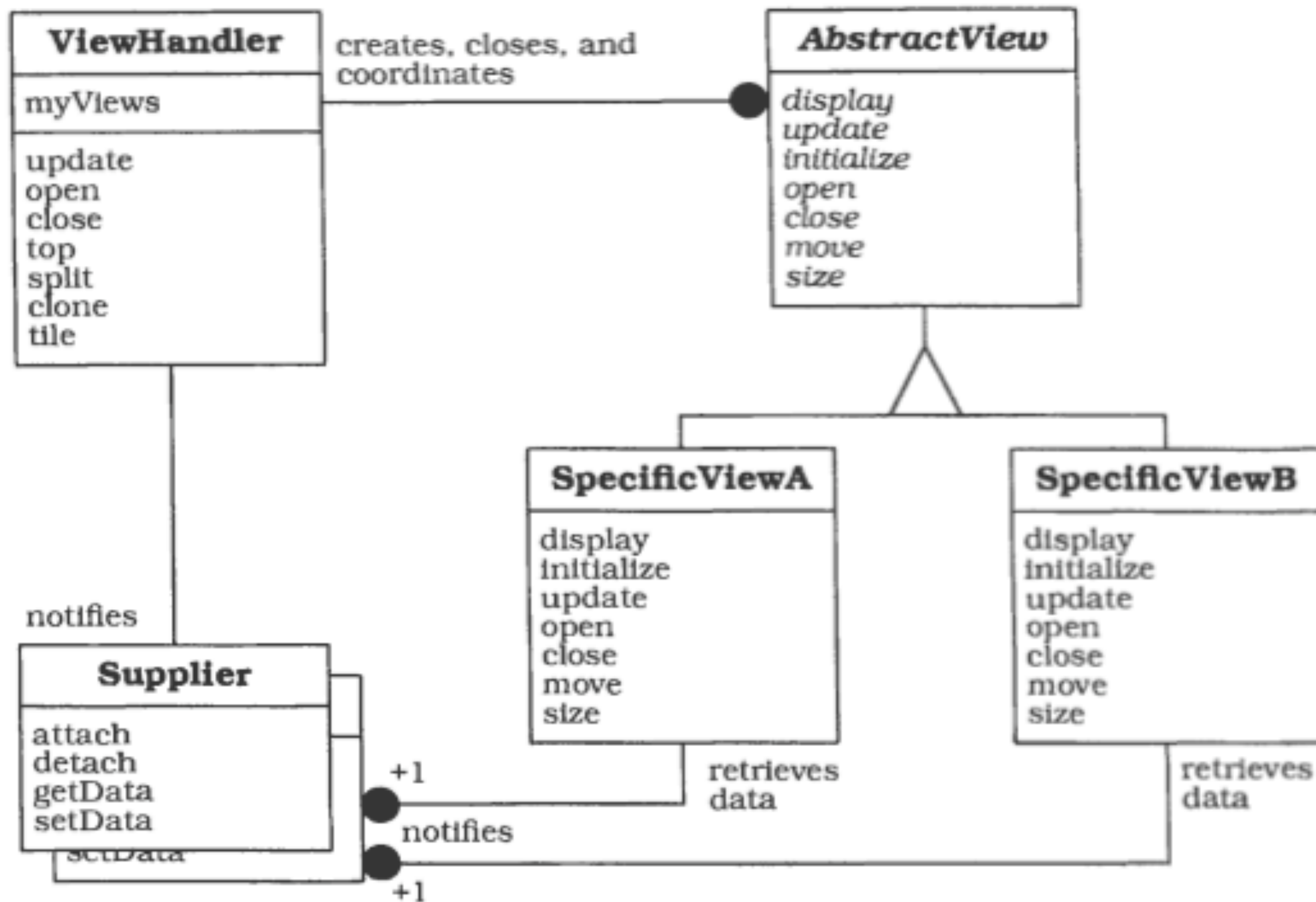
View Handler

- Interaction protocol



View Handler

- Component structure and inter-relationships



View Handler

- Strengths
  - Uniform handling of views
    - All views share a common interface
  - Extensibility and changeability of views
    - New views or changes in the implementation of one view don't affect other component
  - Application-specific view coordination
    - Views are managed by a central instance, so it is easy to implement specific view coordination strategies (e.g. order in updating views)

View Handler

- Weaknesses
  - Efficiency loss (indirection)
    - Negligible
  - Restricted applicability : useful only with
    - Many different views
    - Views with logical dependencies
    - Need of specific view coordination strategies

View Handler

- Variant
  - View Handler with Command objects
    - Uses command objects to keep the view handler independent of specific view interface
    - Instead of calling view functionality directly, the view handler creates an appropriate command and executes it

View Handler

- Known uses
  - Macintosh Window Manager
    - Window allocation, display, movement and sizing
    - Low-level view handler : handles individual window
  - Microsoft Word
    - Window cloning, splitting, tiling…

Outline

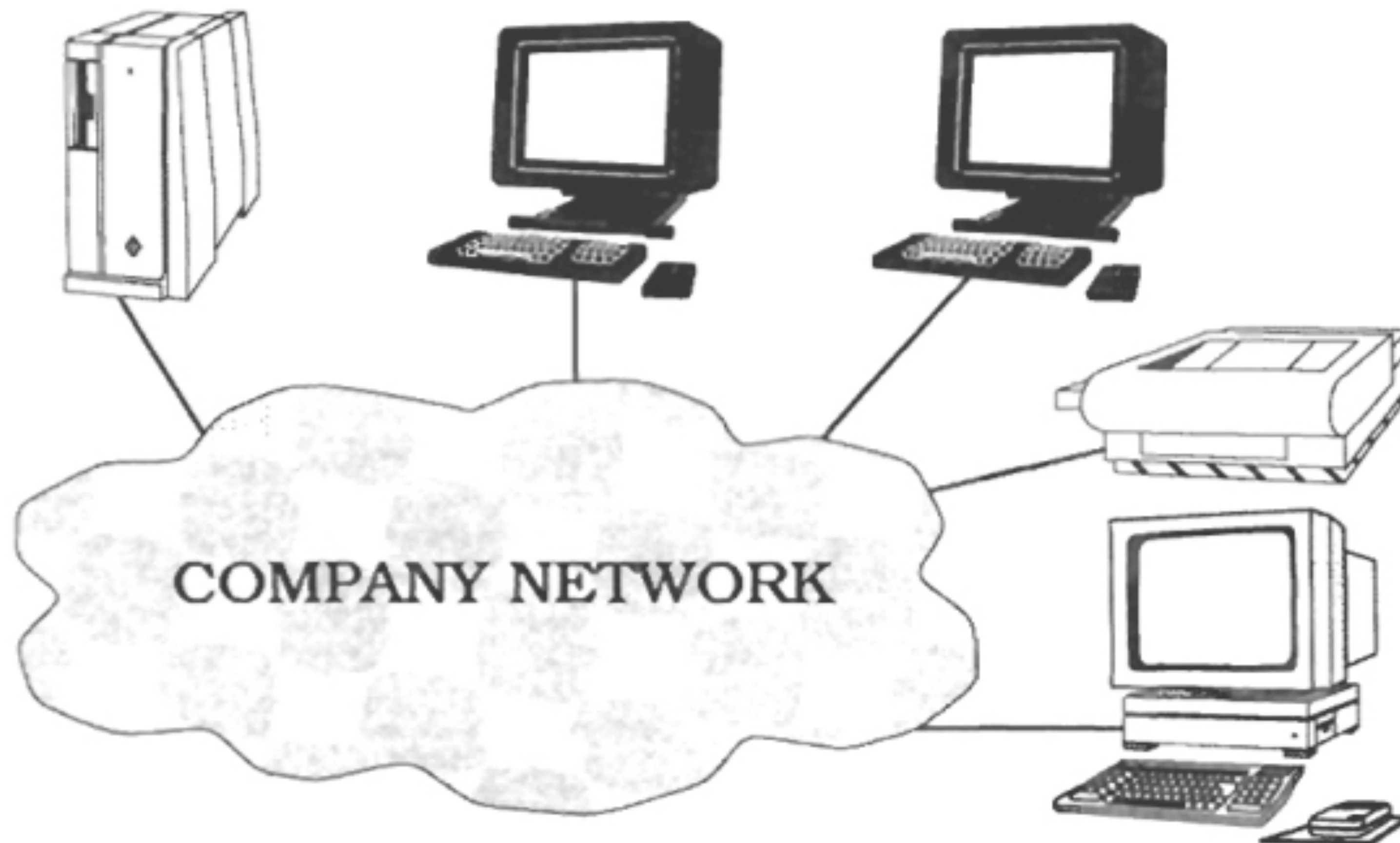
- Command Processor
- View Handler
- **Forwarder-Receiver**
- Client-Dispatcher-Server
- Publisher-Subscriber
- Pattern Systems

Forwarder-Receiver

- Goal
  - Provide transparent inter-process communication for peer-to-peer applications
  - Decouple peers from the underlying communication mechanisms
- Applicability
  - Distributed applications that should allow the exchangeability of inter-process communication mechanisms (TCP/IP, sockets,…)
  - Cooperation of components follows a peer-to-peer model
  - Communication between peers should not have a major impact on performance

Forwarder-Receiver

- Example



Forwarder-Receiver

- Components
  - Peer
    - Responsible for application tasks
    - Knows name(s) of remote peer(s) it needs to communicate with
    - Uses forwarder to send messages (requests)
    - Uses receiver to receive messages (responses)

Forwarder-Receiver

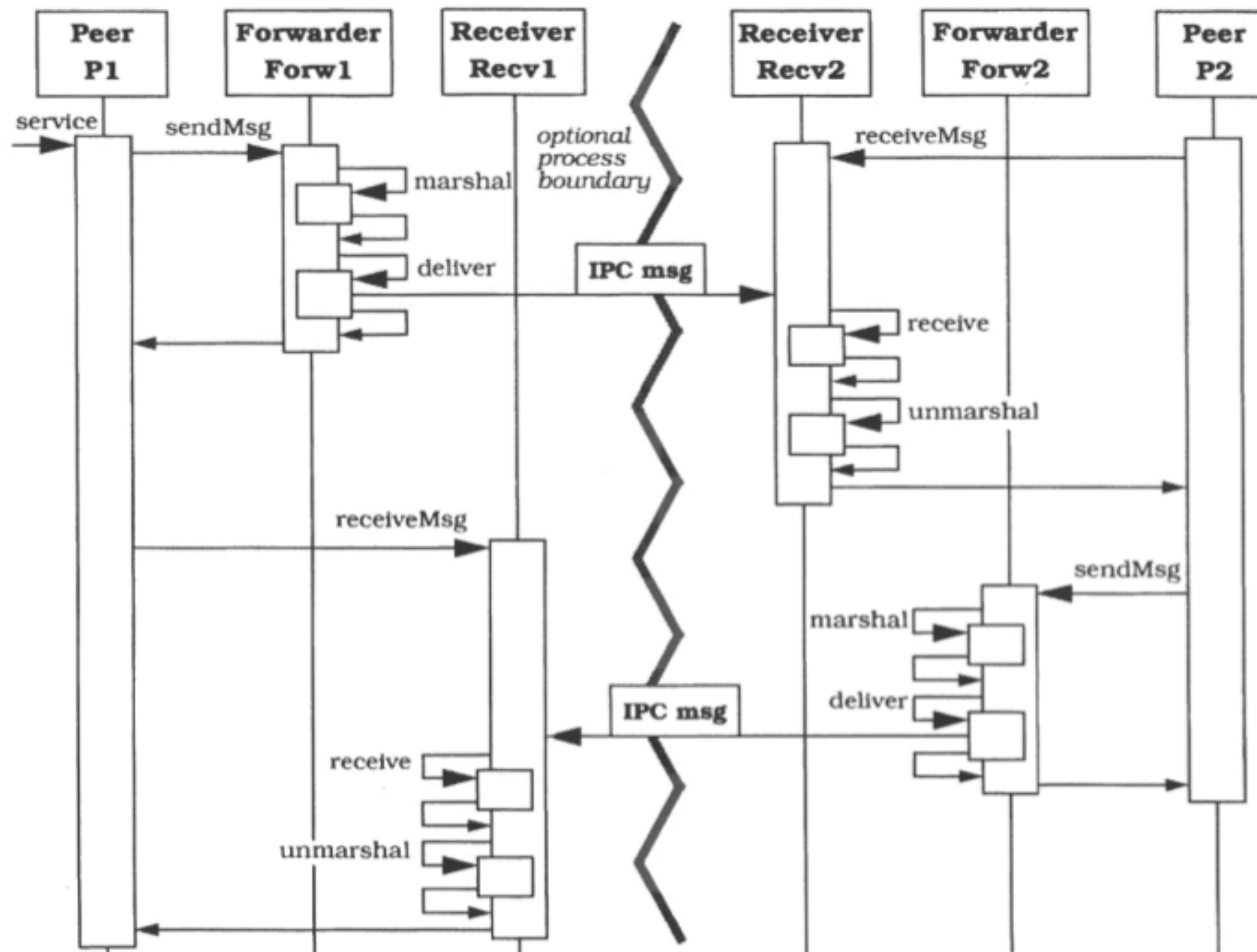
- Components
  - Forwarder
    - Provides general interface for sending messages across process boundaries
    - Marshals and delivers messages to remote receivers
    - Maps names to physical addresses
    - Name of forwarder's peer is included in transmitted message

Forwarder-Receiver

- Components
  - Receiver
    - Provides general interface for receiving messages across process boundaries
    - Receives and unmarshals messages from remote forwarders

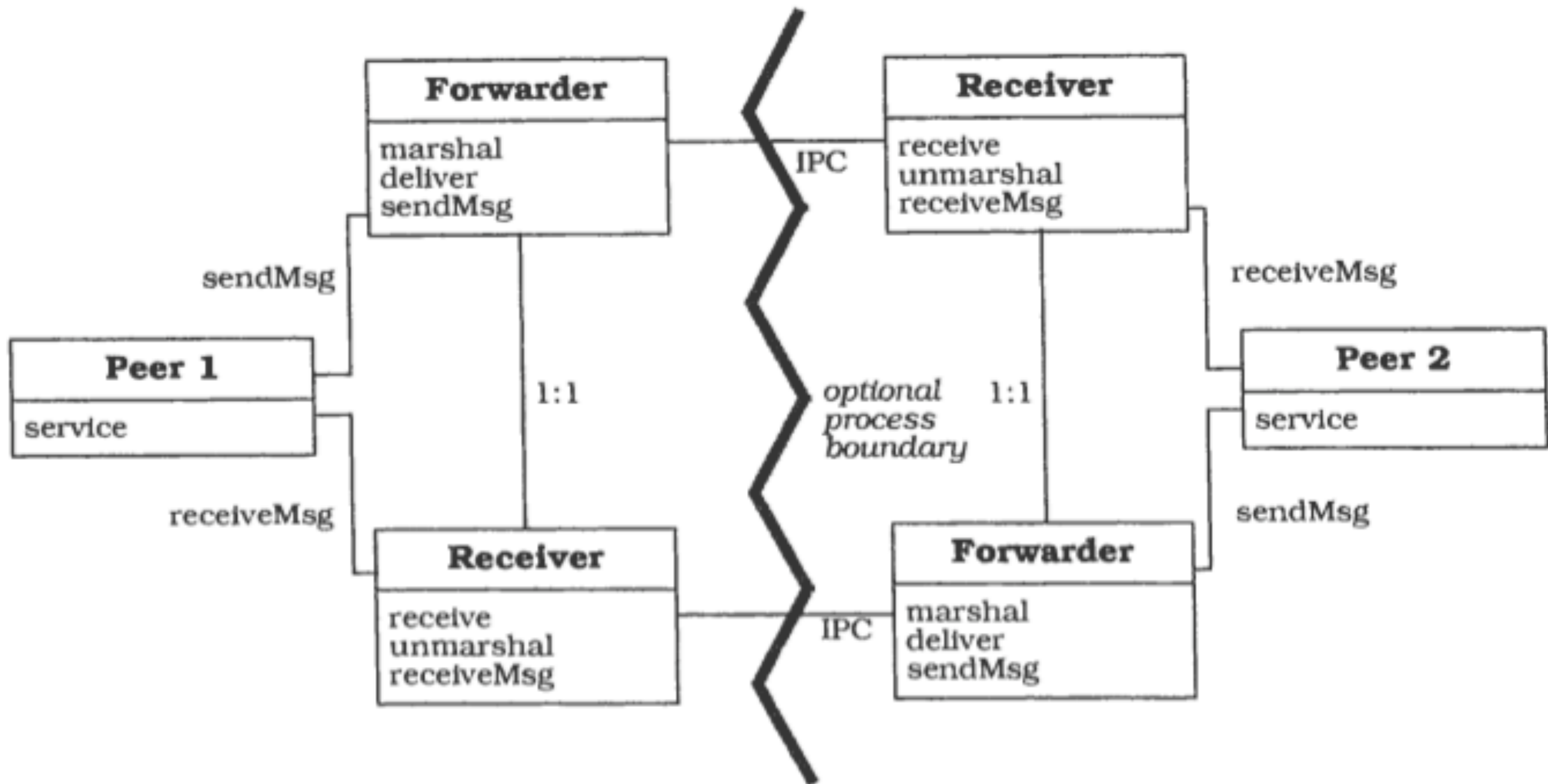
Forwarder-Receiver

- Interaction protocol



Forwarder-Receiver

- Component structure and inter-relationships



Forwarder-Receiver

- Strengths
  - Efficient inter-process communication
    - Peer-to-peer communication between components
    - Every forwarder knows the physical location of its potential receivers, no need to locate them
    - But additional level of indirection (negligible in most cases)
  - Encapsulation of IPC facilities
    - Change of IPC doesn't affect other components

Forwarder-Receiver

- Weakness
  - No support for flexible re-configuration of components
    - Hard to adapt if distribution of peers may change at run-time
    - Can be solved by adding a central dispatcher component using Client-Dispatcher-Server design pattern
(see next pattern)

Forwarder-Receiver

- Variant
  - Forwarder-Receiver without name-to-address mapping
    - Used when performances are more important than encapsulation of IPC mechanisms
    - Peers need to tell their forwarders the physical location of the recipient
    - Harder to switch IPC mechanism

Forwarder-Receiver

- Known uses
  - ATM-P switching system
    - Uses Forwarder-Receiver to implement the IPC between statically-distributed components
  - BrouHaHa
    - Implements IPC in distributed Smalltalk environment

Outline

- Command Processor
- View Handler
- Forwarder-Receiver
- **Client-Dispatcher-Server**
- Publisher-Subscriber
- Pattern Systems

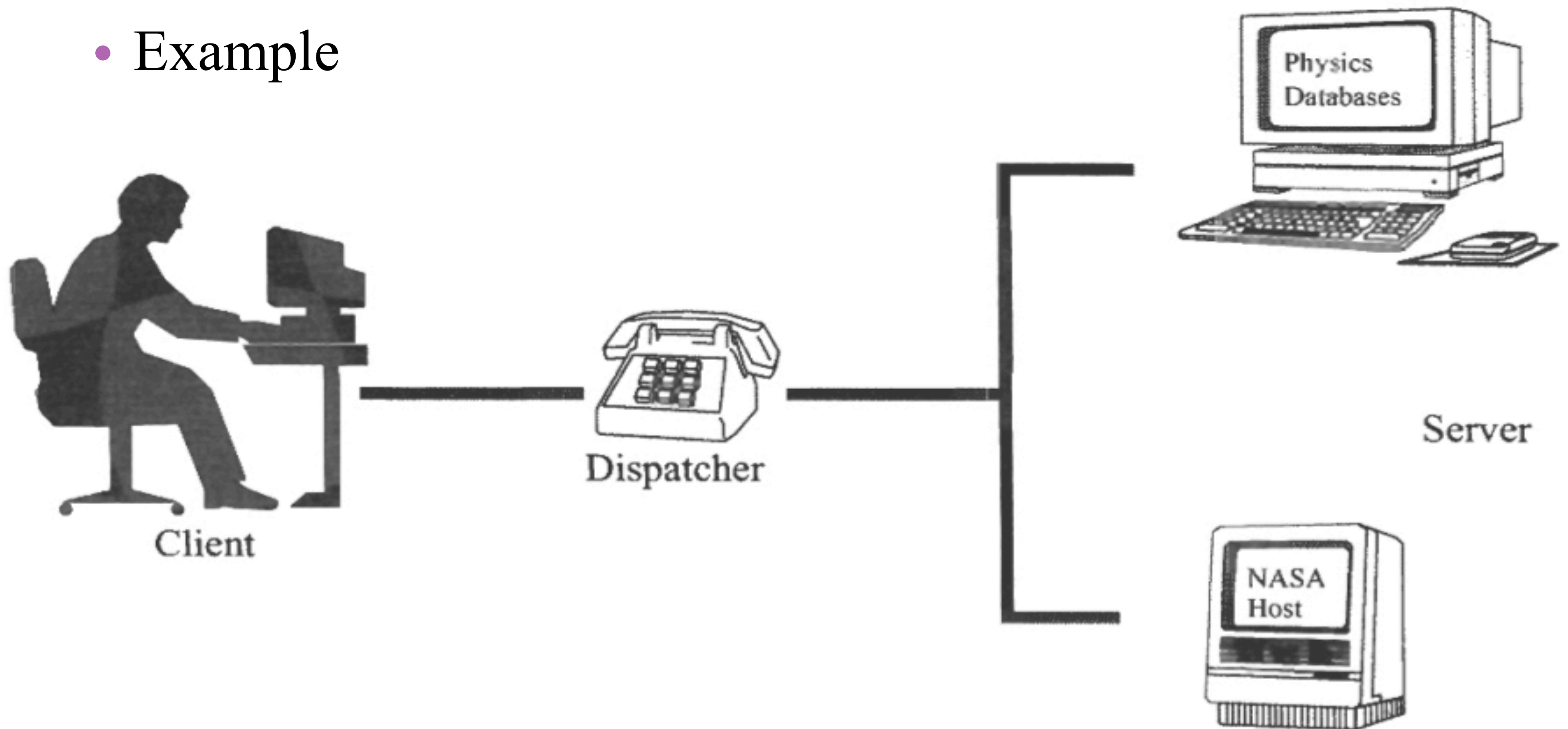
Client-Dispatcher-Server

- Goals
  - Introduce an intermediate layer between clients and servers : the dispatcher
  - Provide location transparency
  - Hides details of establishment of communication

- Applicability
  - A software system integrating a set of distributed servers, with the servers running locally or distributed over a network.

Client-Dispatcher-Server

- Example



Client-Dispatcher-Server

- Components
  - Client
    - Performs some domain-specific tasks
    - Accesses operations offered by servers
      - Ask the dispatcher for a communication channel
      - Send its request to the server by this channel

Client-Dispatcher-Server

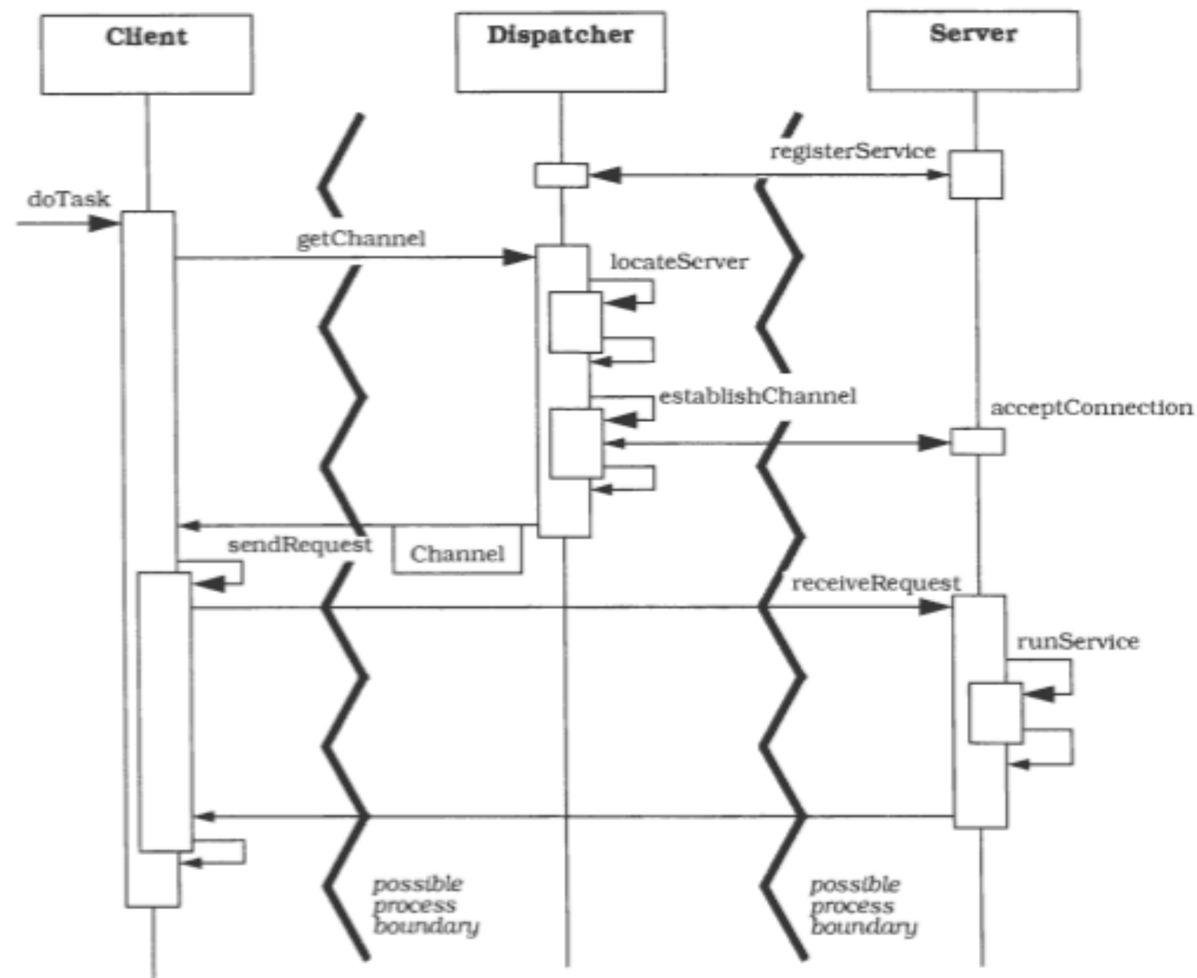
- Components
  - Server
    - Provides services to clients
    - Registers itself with the dispatcher

Client-Dispatcher-Server

- Components
  - Dispatcher
    - Establishes communications channels
    - Locates servers
    - (Un-)Registers servers
    - Maintains a map of server locations and name

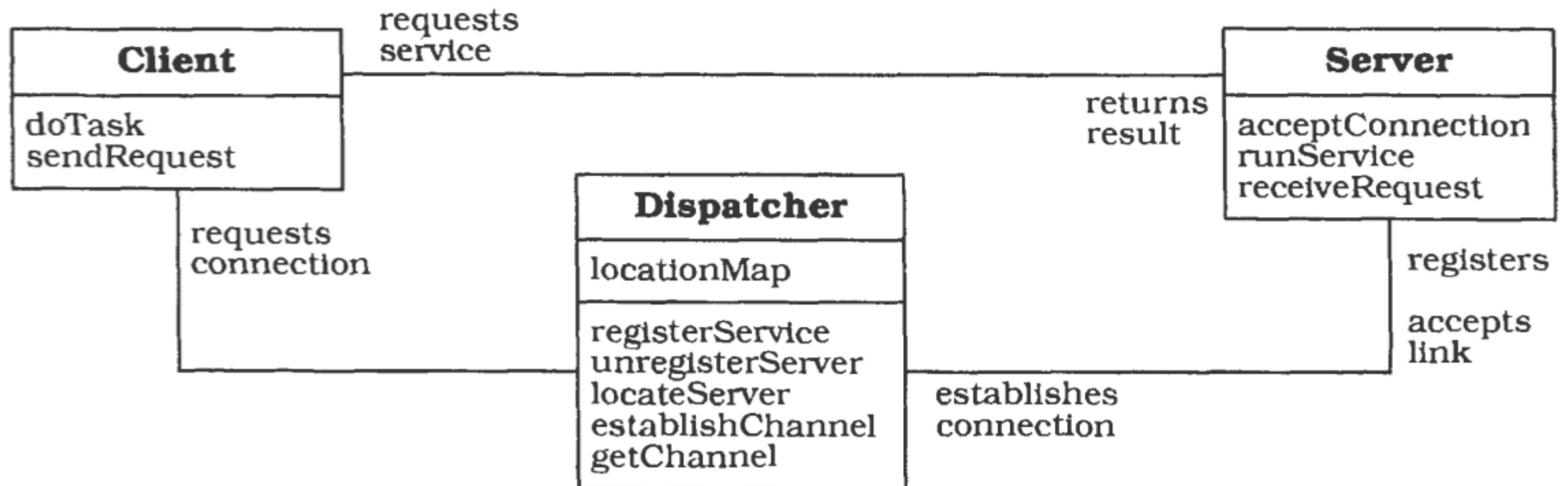
Client-Dispatcher-Server

- Interaction protocol



Client-Dispatcher-Server

- Component structure and inter-relationships



Client-Dispatcher-Server

- Strengths
  - Exchangeability of servers
    - Servers can be changed without any modification
  - Location and migration transparency
    - Clients don't depend on server location
  - Re-configuration
    - Configuration may be changed
  - Fault tolerance
    - New servers can be activated at run-time

Client-Dispatcher-Server

- Weaknesses
  - Efficiency loss : workload of dispatcher
  - Sensitivity to change in the interfaces of the dispatcher component
    - Dispatcher plays a central role, any change in its interface may force changes in every implementations

Client-Dispatcher-Server

- Variants
  - Distributed Dispatchers
    - Instead of using a single dispatcher component in a network environment, distributed dispatchers may be introduced
  - Client-Dispatcher-Server with communication managed by clients
    - Instead of establishing a communication channel to servers, a dispatcher may only return the physical server location to the client.

Client-Dispatcher-Server

- Variants
  - Client-Dispatcher-Server with heterogeneous communication
    - Dispatcher is capable of supporting more than one communication mechanism
    - Each server register itself with the dispatcher and specifies the communication mechanism it supports

Client-Dispatcher-Server

- Variants
  - Client-Dispatcher-Service
    - Clients address services and not servers.
    - When the dispatcher receives a request, it looks up which servers provide the specified service in its repository, and establishes a connection to one of these service providers. If it fails to establish the connection, it may try to access another server providing the same service instead, if one is available

Client-Dispatcher-Server

- Known uses
  - Sun's implementation of Remote Procedure Calls
    - Combination of the variants Distributed Dispatchers and Client-Dispatcher-Server with communication managed by client
  - OMG Corba
    - Uses the principles of the Client-Dispatcher-Server design pattern for refining and instantiating the Broker architectural pattern

Outline

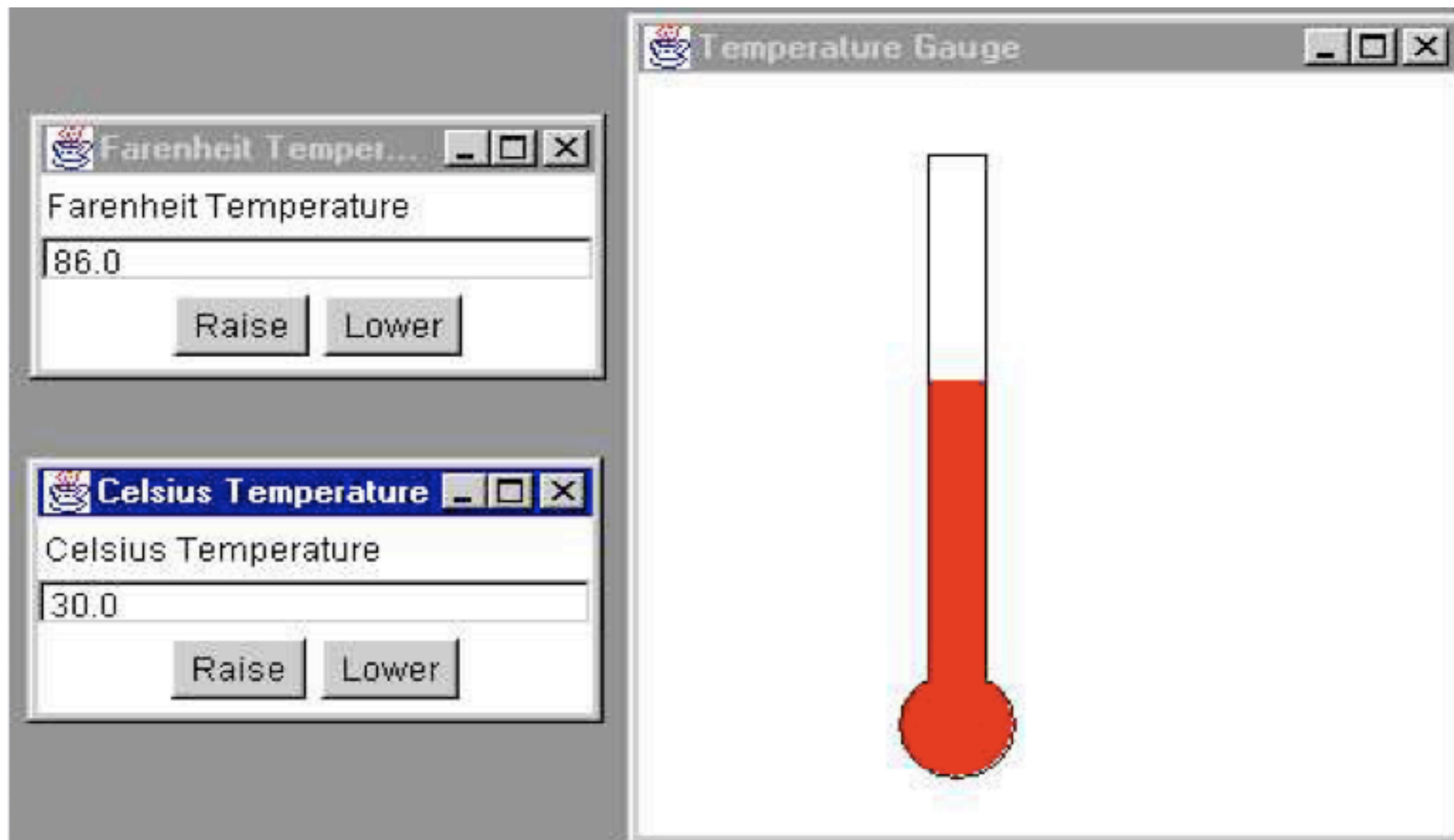
- Command Processor
- View Handler
- Forwarder-Receiver
- Client-Dispatcher-Server
- **Publisher-Subscriber**
- Pattern Systems

Publisher-Subscriber

- Goal
  - Help to keep the state of cooperation components synchronized
  - One publisher notifies any number of subscribers about changes to its state
- Applicability
  - Applications in which data changes in one place but many other components depend on this data
  - Number and identities of dependant components may change over time
  - Example : graphical user interfaces

Publisher-Subscriber

- Example



Publisher-Subscriber

- Components
  - Publisher
    - Maintains registry of currently-subscribed components
    - Sends notification to subscribers when its state has changed
  - Subscriber
    - Can use the (un)subscribe interface of the publisher
    - Retrieve changed data from publisher

Publisher-Subscriber

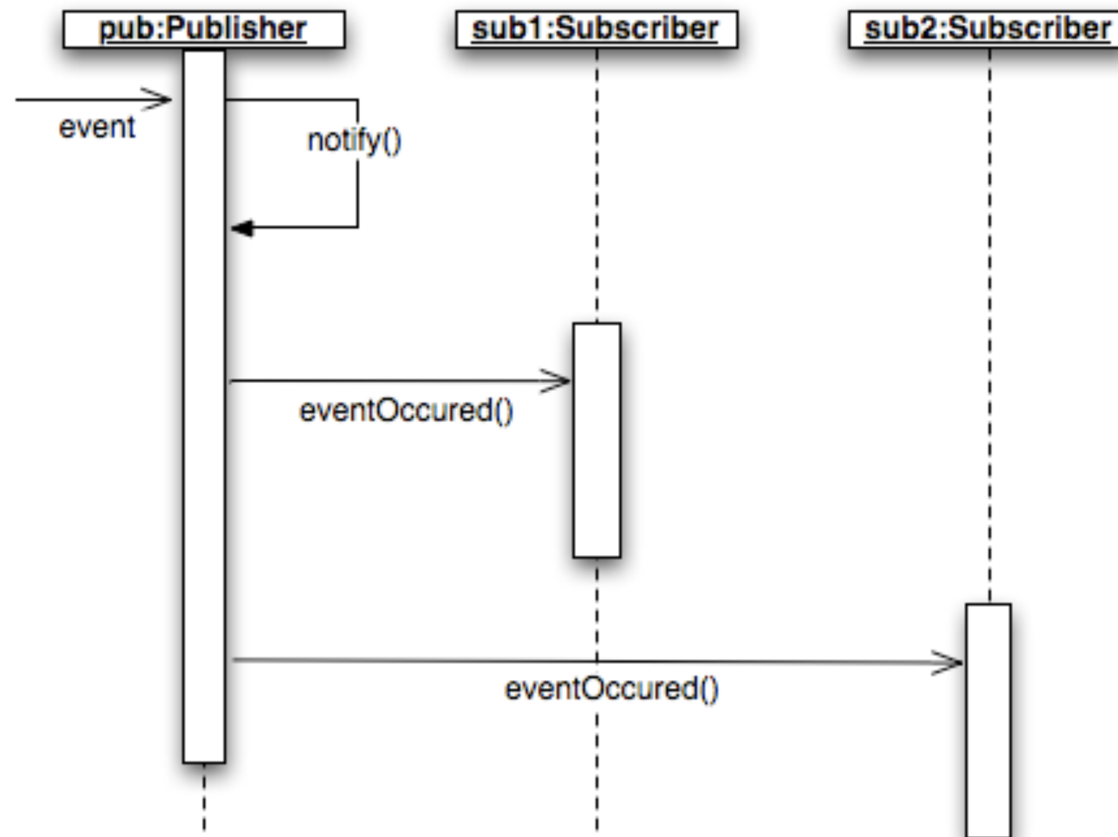
- Push model
  - Publisher sends all changed data when it notifies the subscriber
  - Rigid dynamic behavior
  - Poor choice for complex data changes
  - Useful when subscribers need published information most of the time

Publisher-Subscriber

- Pull model
  - Publisher only sends minimal information when sending a change notification
  - Subscribers are responsible for retrieving the data they need
  - Offers more flexibility but higher number of messages between publisher and subscriber
  - Useful when only individual subscribers can decide if and when they need a specific piece of information

Publisher-Subscriber

- Interaction protocol



Publisher-Subscriber

- Strengths
  - Loosely-coupled
    - Publishers are loosely coupled to subscribers
  - Scalable in small installations
- Weaknesses
  - Not so scalable in large installations
  - Publisher assumes that subscriber is listening

Publisher-Subscriber

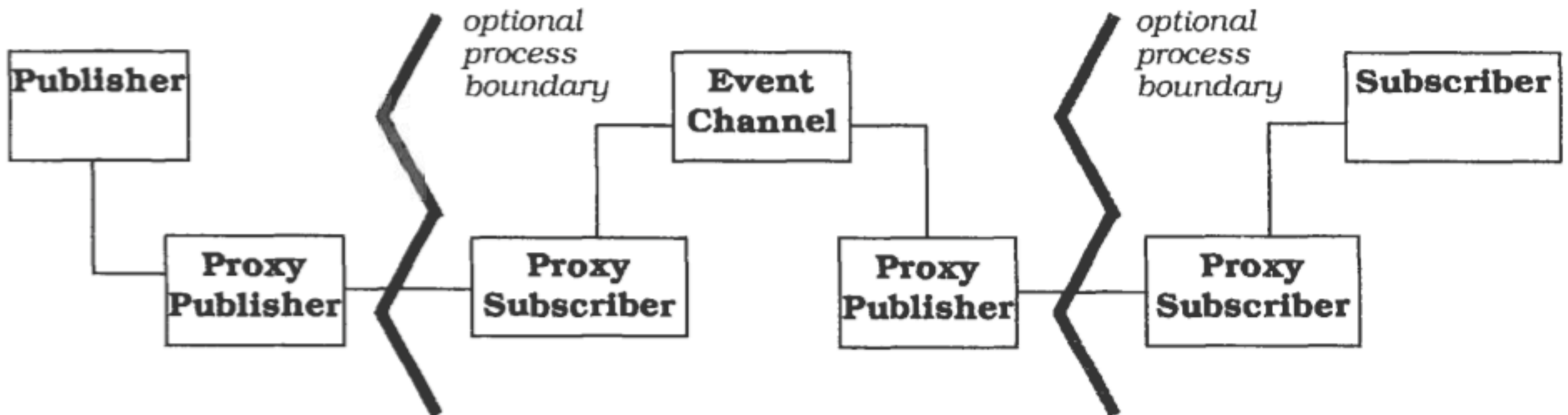
- Variants
  - Gatekeeper
    - Publisher notifies remote subscribers
  - Event Channel
    - Strongly decouples publishers and subscribers
    - Possible to have more than one publisher
    - Subscribers only wish to be notified about changes, don't care in which component changes occured
    - Publishers are not interested in which components are subscribing

Publisher-Subscriber

- Variants
  - Event channel (cont’d)
    - Event channel created and placed between publishers and subscribers
    - Appears as a subscriber to publishers
    - Appears as a publisher to subscribers
    - Event channel, subscriber and publisher can be in different processes
    - Can use buffers, can be chained (Unix pipes)

Publisher-Subscriber

- Variants
  - Event channel (cont'd)



Publisher-Subscriber

- Variants
  - Use of Producer-Consumer style of cooperation
    - Producer supplies information, consumer accepts it
    - Strongly decoupled thanks to a buffer
    - Only synchronization is for buffer under/overflow
    - Event-Channel pattern can simulate a P-C with more than one producer or consumer
- Known uses
  - Java Swing, GUIs

Outline

- Command Processor
- View Handler
- Forwarder-Receiver
- Client-Dispatcher-Server
- Publisher-Subscriber
- **Pattern Systems**

Pattern systems

- Patterns don’t exist in isolation, many interdependencies
- A simple catalog is not enough
- Describe
  - How individual patterns are connected with other patterns
  - How patterns can be implemented
  - How software development with patterns is supported
- Powerful vehicle for expressing and constructing software architecture
- Support the development of high-quality software systems

Pattern systems

A *pattern system for software architecture* is a collection of patterns for software architecture, together with guidelines for their implementation, combination and practical use in software development.

- Requirements
  - Sufficient base of patterns
  - Describe all patterns uniformly
  - Expose relationships between patterns
  - Organize constituent patterns
  - Support the construction of software systems
  - Support its own evolution

Pattern systems

- Pattern classification
  - Pattern system is useless if developers have to inspect all patterns
  - Classification is needed
  - Should be simple and easy to learn
  - Should consist of only a few classification criteria
  - Each classification criterion should reflect natural properties of patterns
  - Should provide a roadmap that leads users to set of useful patterns
  - Schema should be open to integration of new patterns

Pattern systems

- Pattern categories
  - Most fundamental classification criterion
    - Architectural patterns
    - Design patterns
    - Idioms

Pattern systems

- Problem categories
  - Second classification criterion
  - Abstracting from specific problems
  - Examples :
    - Distributed Systems
    - Interactive Systems
    - Communication
  - A pattern can belong to more than one category

Pattern systems

- Classification schema

	Architectural Patterns	Design Patterns	Idioms
From Mud to Structure	Layers (31) Pipes and Filters (53) Blackboard (71)		
Distributed Systems	Broker (99) Pipes and Filters (53) Microkernel (171)		
Interactive Systems	MVC (125) PAC (145)		
Adaptable Systems	Microkernel (171) Reflection (193)		
Structural Decomposition		Whole-Part (225)	
Organization of Work		Master-Slave (245)	
Access Control		Proxy (263)	
Management		Command Processor (277) View Handler (291)	
Communi-cation		Publisher-Subscriber (339) Forwarder-Receiver (307) Client-Dispatcher-Server (323)	
Resource Handling			Counted Pointer (353)

Pattern systems

- Pattern selection

1. Specify the problem
2. Select the pattern category
3. Select the problem category
4. Compare the problem descriptions
5. Compare strengths and weaknesses
6. Select the variant

=> Pattern not found? Select an alternative problem category (back to step 4)

Pattern systems

- Pattern systems as implementation guidelines
  - The implementation steps for individual patterns are its building-blocks.
  - They can be plugged with the implementation steps of other patterns, namely those that refer to the pattern you are implementing.
  - You can therefore solve complex problems by recursively applying the implementation steps of all patterns that are involved in its solution

Pattern systems

- Evolution of pattern systems
  - New patterns will emerge
  - Existing patterns may be outdated and removed
  - Pattern description can evolve
  - Relationships between pattern may be updated

Pattern systems

- Evolution of pattern descriptions
  - Whenever a pattern is applied, the experience gained from its application should be used for a critical review of the pattern and its description
    - Update strengths, weaknesses and limitations
    - Update structure and dynamics
    - Add a new variant
  - Updating the pattern descriptions will help the pattern system to remain useful

Pattern systems

- Writer's workshops : structured format for pattern review
  - Goal : acquire as much feedback for constructive improvement as possible
  - Pattern is discussed by a group of people including the author and a group of reviewers familiar with the contents of the pattern description

Pattern systems

- Pattern-Mining : Mine pattern addressing unsolved problems

1. Find at least three examples
2. Extract the solution schema
3. Declare the solution schema to be a “pattern-candidate”
4. Run a writer's workshop
5. Apply the candidate pattern in real-world
6. Declare the candidate to be a pattern if its application is successful

Pattern systems

- Integration of new patterns

1. Specify the relationship of the new pattern with existing patterns in the pattern system

2. Classify the pattern in the appropriate pattern and problem categories

Pattern systems

- Removing outdated patterns
  - Due to
    - Disappearance of the problem
    - Better alternatives
    - Technology evolution
  - Update the pattern system according to the removal

Pattern systems

- Extending the Organization Schema
  - Add new pattern categories
  - Add new (domain-specific) problem categories
    - E.g. : Adaptation provides patterns that help with interface and data conversion

Pattern systems

	Architectural Patterns	Design Patterns	Idioms
From Mud to Structure	Layers (31) Pipes and Filters (53) Blackboard (71)	*Interpreter*	
Distributed Systems	Broker (99) Pipes and Filters (53) Microkernel (171)		
Interactive Systems	MVC (125) PAC (145)		
Adaptable Systems	Microkernel (171) Reflection (193)		
Creation		*Abstract Factory* *Prototype* *Builder*	*Singleton* *Factory Method*
Structural Decomposition		Whole-Part (225) *Composite*	
Organization of Work		Master-Slave (245) *Chain of Responsibility* *Command* *Mediator*	
Access Control		Proxy (263) *Facade* *Iterator*	
Service Variation		*Bridge* *Strategy* *State*	*Template Method*
Service Extension		*Decorator* *Visitor*	
Management		Command Processor (277) View Handler (291) *Memento*	
Adaptation		*Adapter*	
Communi-cation		Publisher-Subscriber (339) Forwarder-Receiver (307) Client-Dispatcher-Server (323)	
Resource Handling		*Flyweight*	Counted Pointer (353)