Architectural Patterns

Pascal Molli
(a System of patterns Buschman et al)

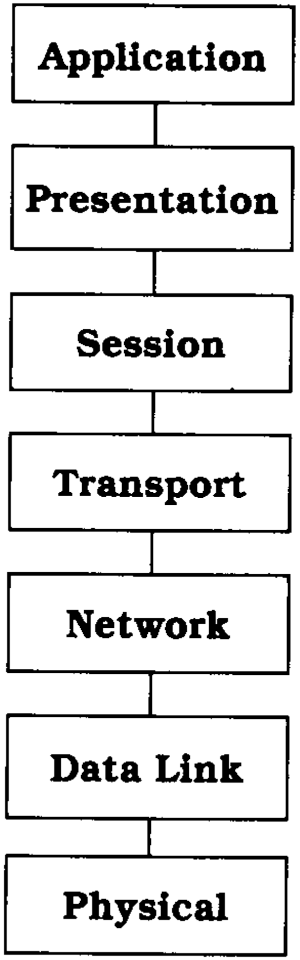
Architectural Patterns…

- From MUD to Structure…
  - Layers, Pipe and Filters, Blackboard
- Distributed Systems…
  - Broker, Pipe and Filters, Microkernel
- Interactive Systems…
  - MVC, PAC
- Adaptable Systems…
  - Microkernel, Reflection…

Layer

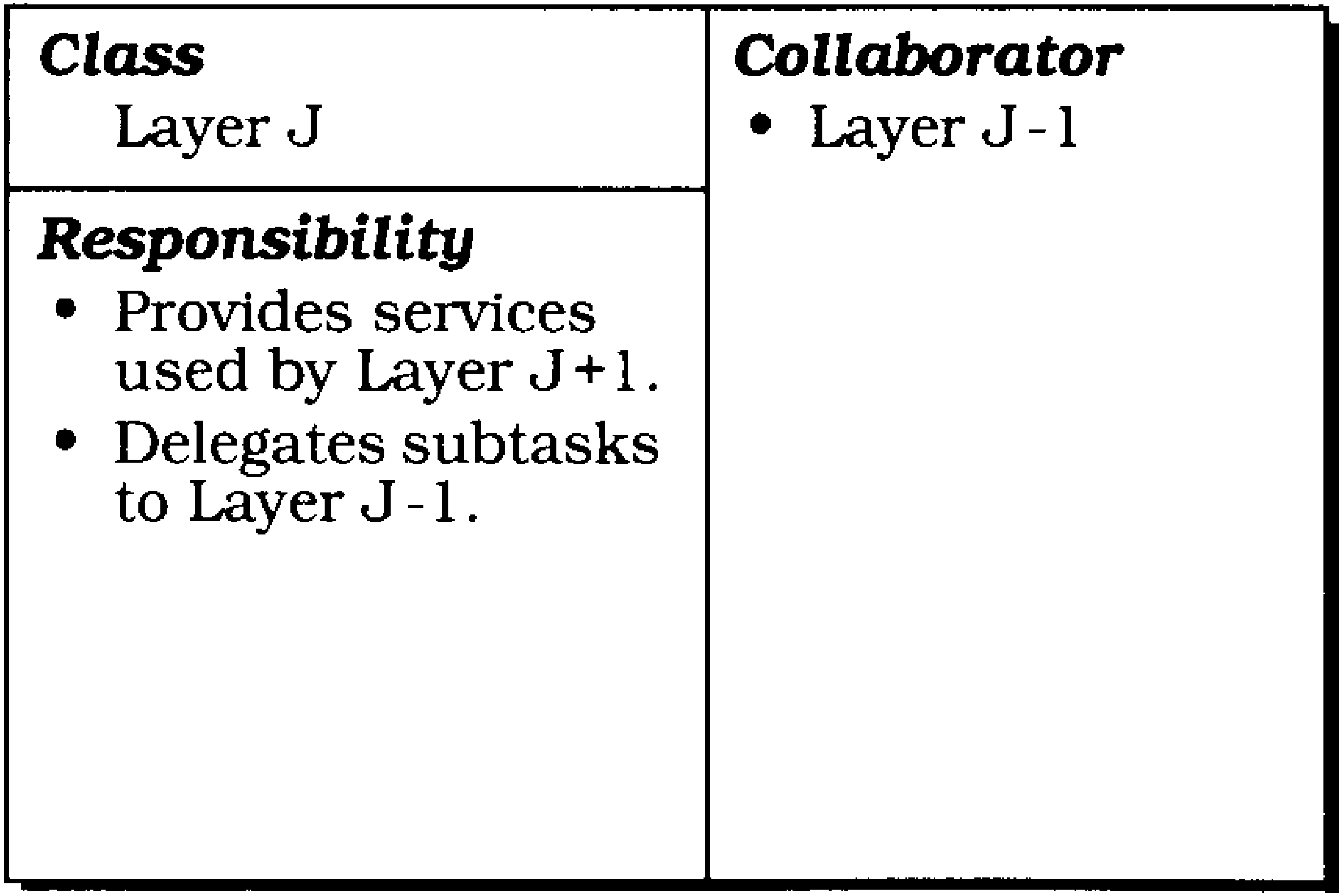
- helps to structure application that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction.

Layer: examples



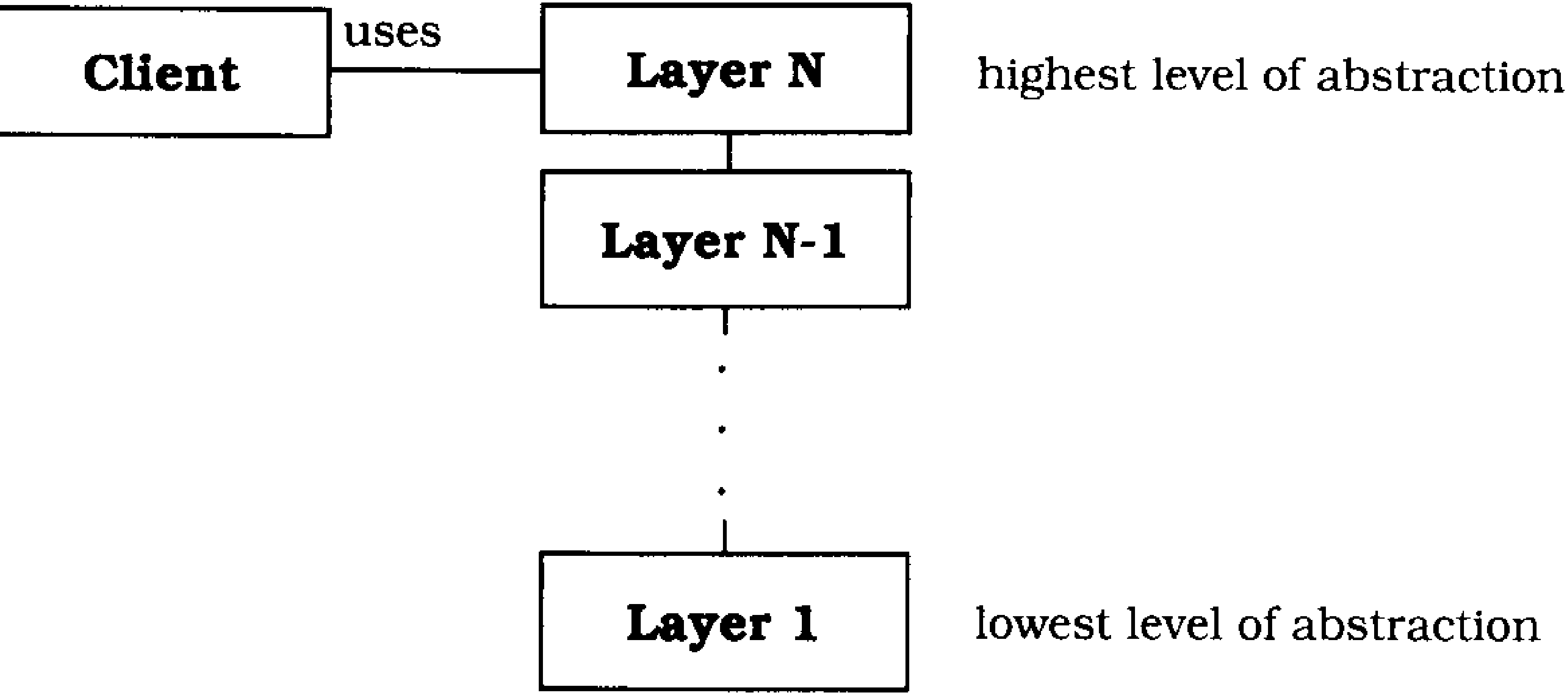
Application	Layer 7	Provides miscellaneous protocols for common activities
Presentation	Layer 6	Structures information and attaches semantics
Session	Layer 5	Provides dialog control and synchronization facilities
Transport	Layer 4	Breaks messages into packets and guarantees delivery
Network	Layer 3	Selects a route from sender to receiver
Data Link	Layer 2	Detects and corrects errors in bit sequences
Physical	Layer 1	Transmits bits: velocity, bit-code, connection, etc.

Layer :Structure

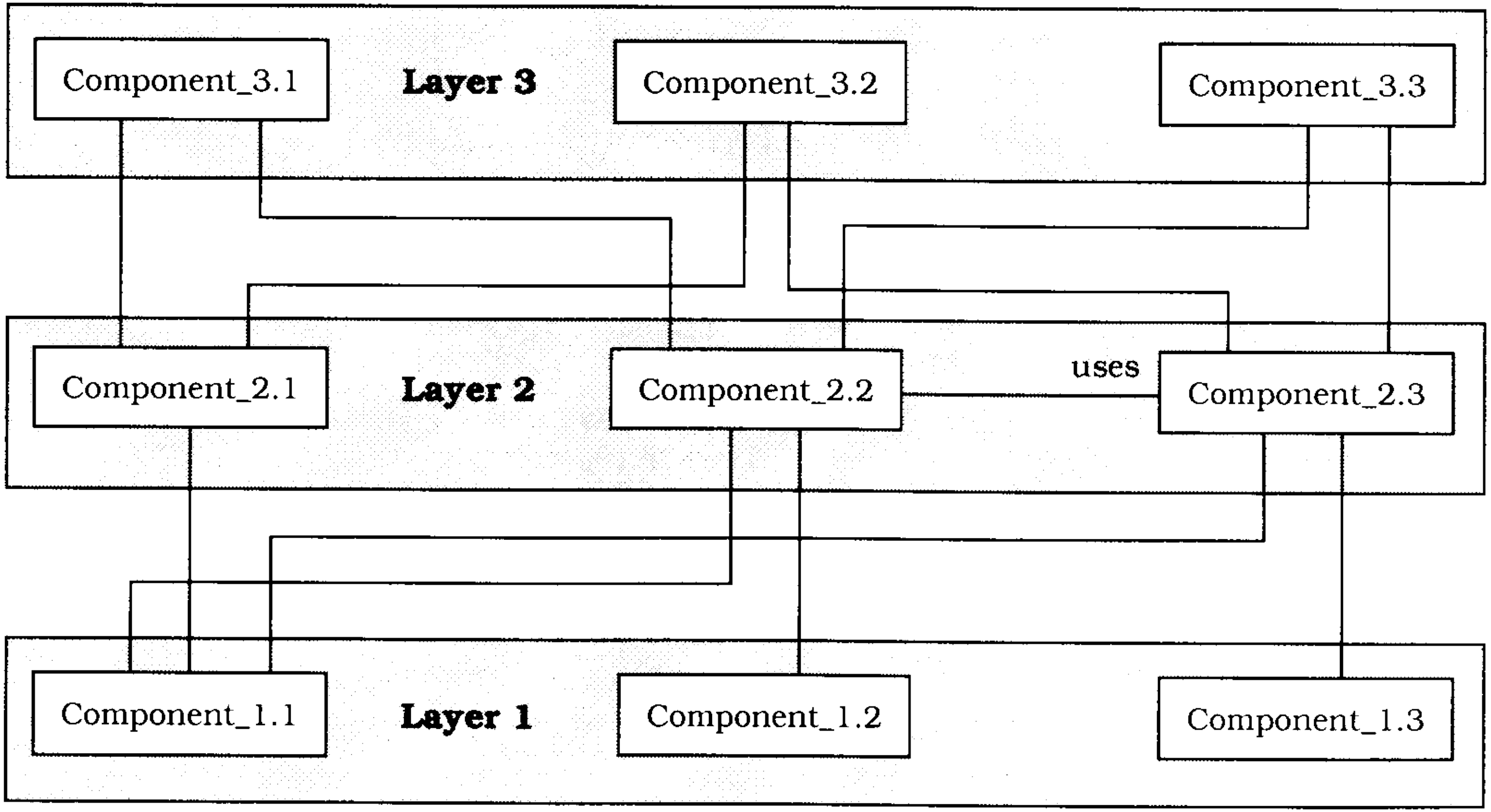


Class Layer J	***Collaborator*** • Layer J - 1
Responsibility • Provides services used by Layer J+1. • Delegates subtasks to Layer J-1.	

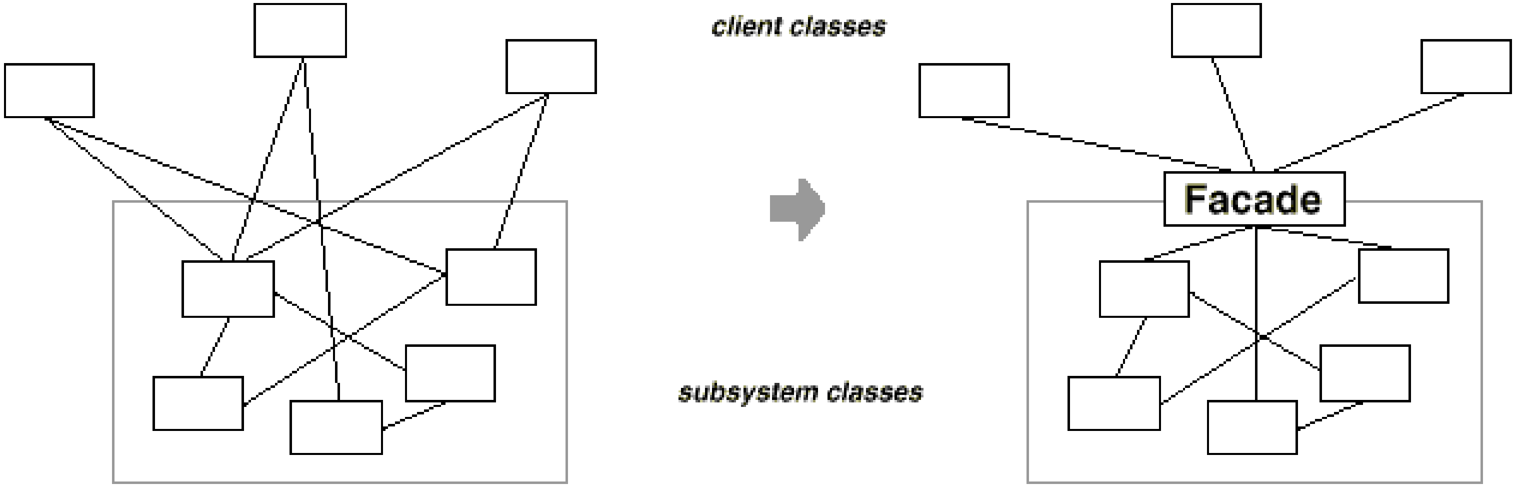
Layer: Structure



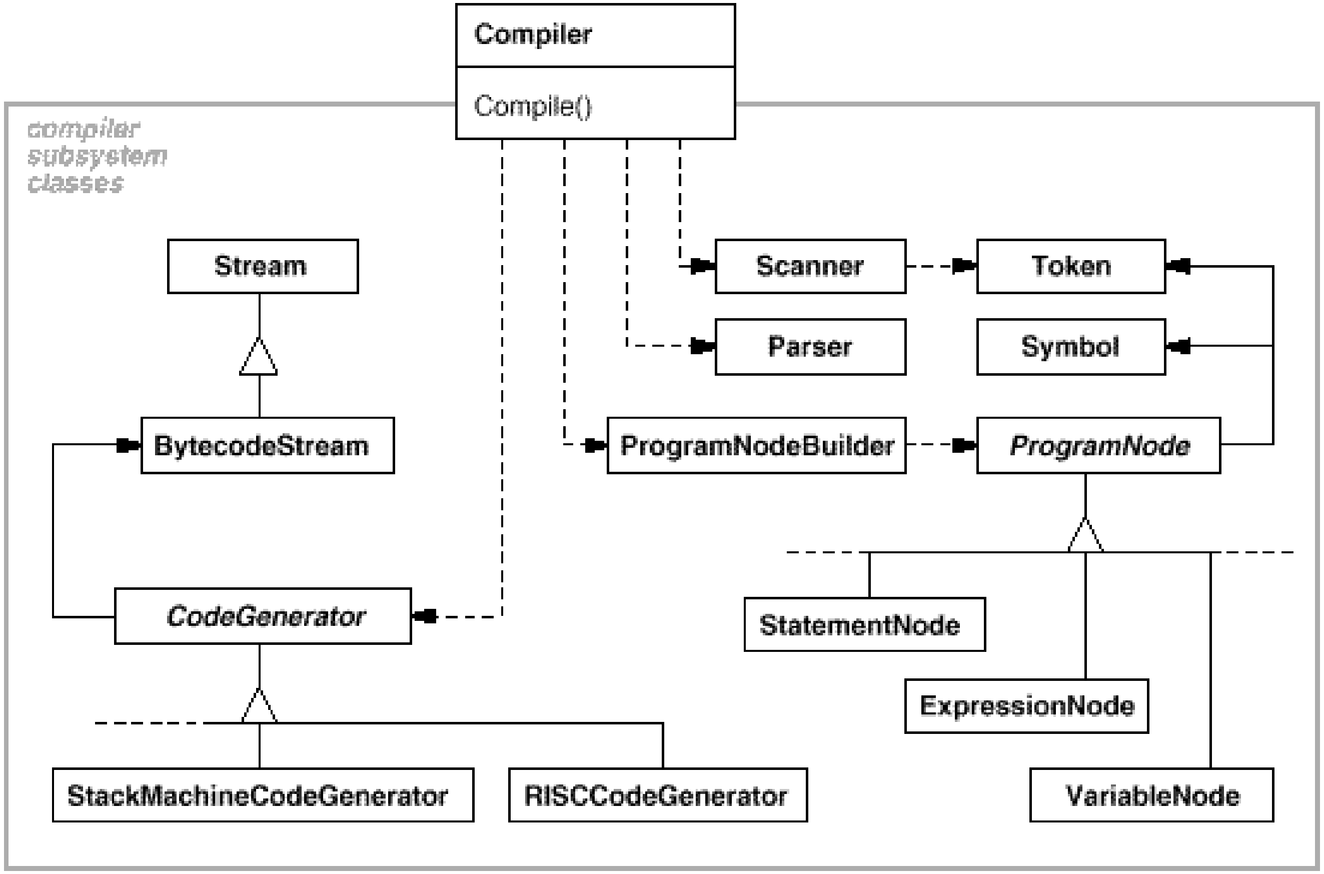
Layer and components…



Layer and Facade DP…



Layer and Facade DP



Layers : Variants

- Relaxed Layered System:
  - A layer « j » can use service of j-1, j-2…
  - A layer can be partially opaque
    - Some service to layer j+1, others to all upper services…
- Layering through inheritance:
  - Lower layers are implemented as base classes
  - Higher level can override lower level…

Layers : Known Uses

- Virtual machines: JVM and binary code format
- API : Layer that encapsulates lower layers
- Information System
  - Presentation, Application logic, Domain Layer, Database
- Windows NT (relaxed for: kernel and IO and hardware)
  - System services,
  - Resource management (Object manager, security monitor, process manager, I/O manager, VM manager, LPC),
  - Kernel (exception handling, interrupt, multipro synchro, threads),
  - HAL (Hardware Abstraction Level)
  - Hardware

Layers: benefits

- Reuse of layers
- Support for standardization (POSIX)
- Dependencies are kept local
- Exchangeabilities :
  - Replacement of old implementation with Adapter Pattern
  - Dynamic exchange with Bridge Pattern

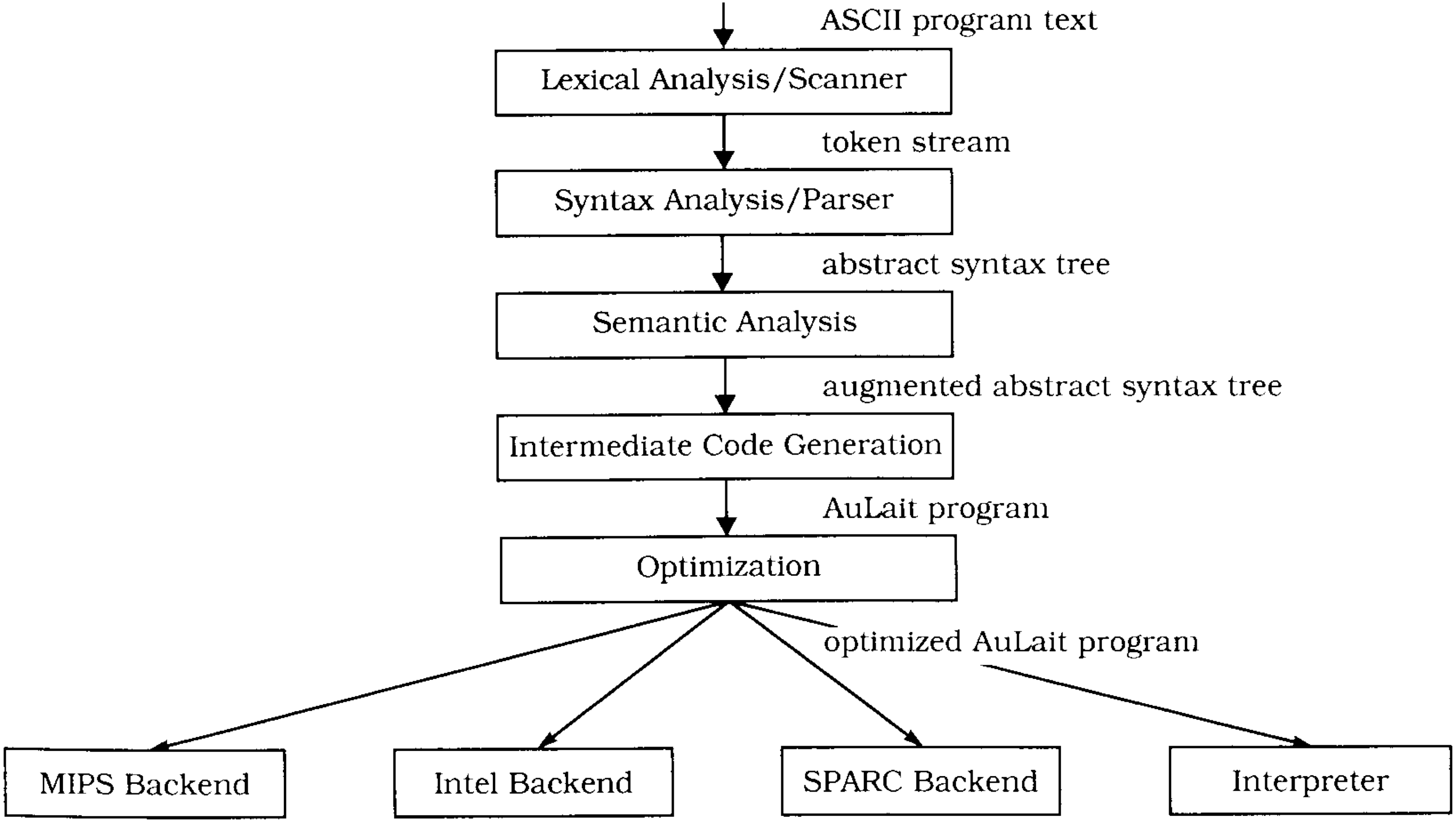
Layers: Liabilities

- Cascades of changing behavior
- Lower efficiency
- Unnecessary work: functions of a layer called many times for one service
- Difficulty of establishing correct granularity of layers: To few layer -> less benefits, to much layer -> complexity and overhead…

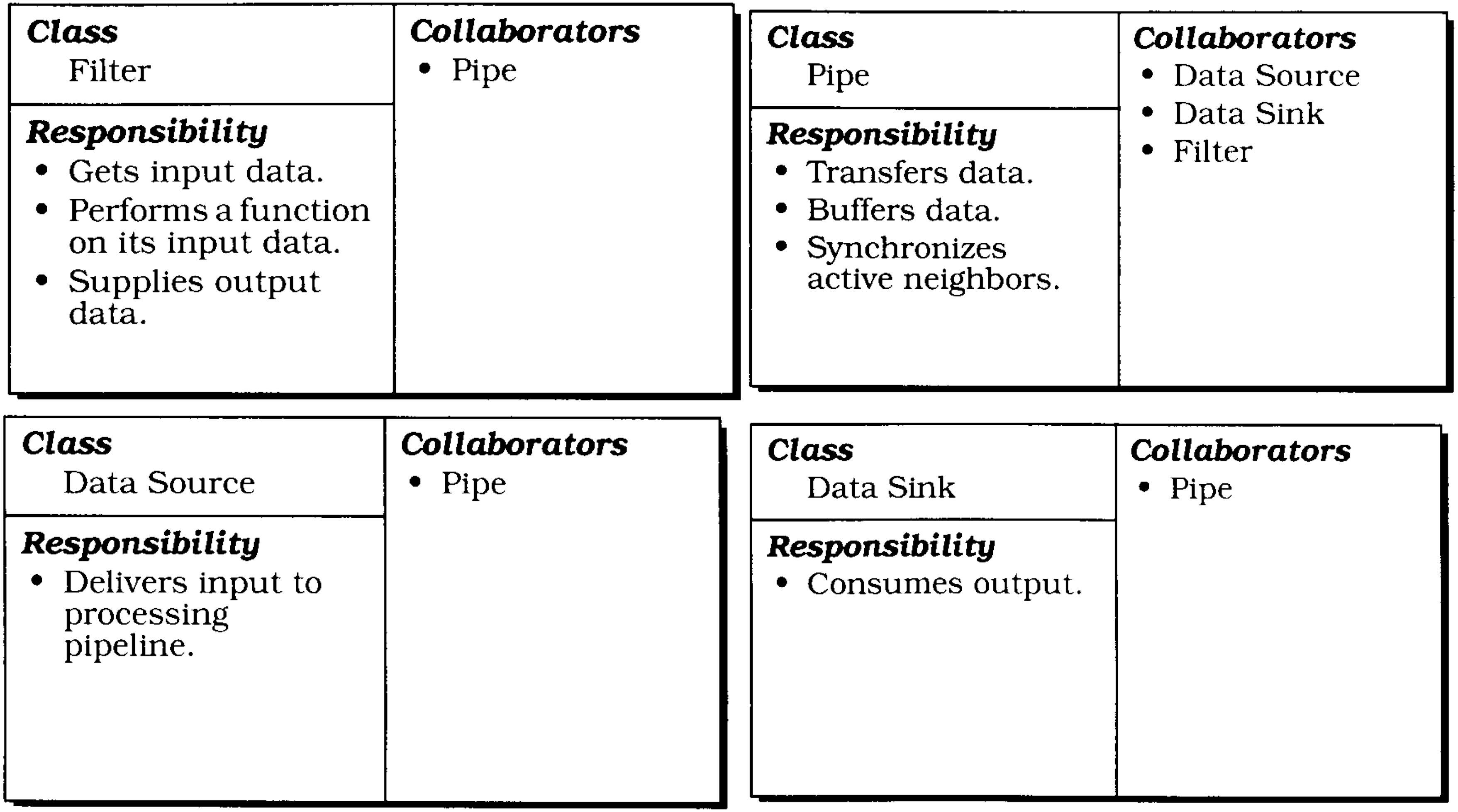
Pipes and Filters

- Provides a structure for systems that process a stream of Data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters.
- Recombining filters allows to build families of related systems.

Pipes and Filters: Example



Pipes and Filters: Structure



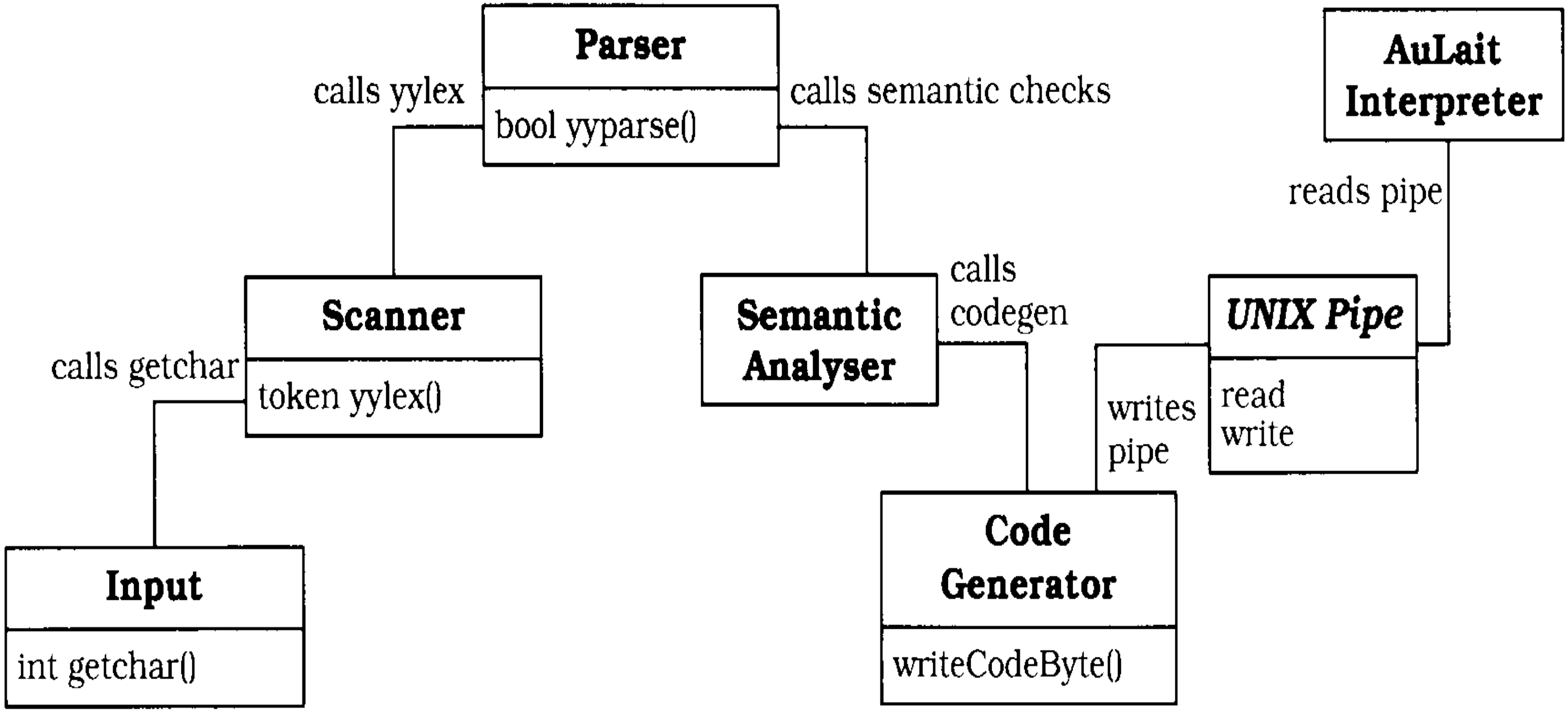
Class Filter	***Collaborators*** • Pipe
Responsibility • Gets input data. • Performs a function on its input data. • Supplies output data.	

Class Pipe	***Collaborators*** • Data Source • Data Sink • Filter
Responsibility • Transfers data. • Buffers data. • Synchronizes active neighbors.	

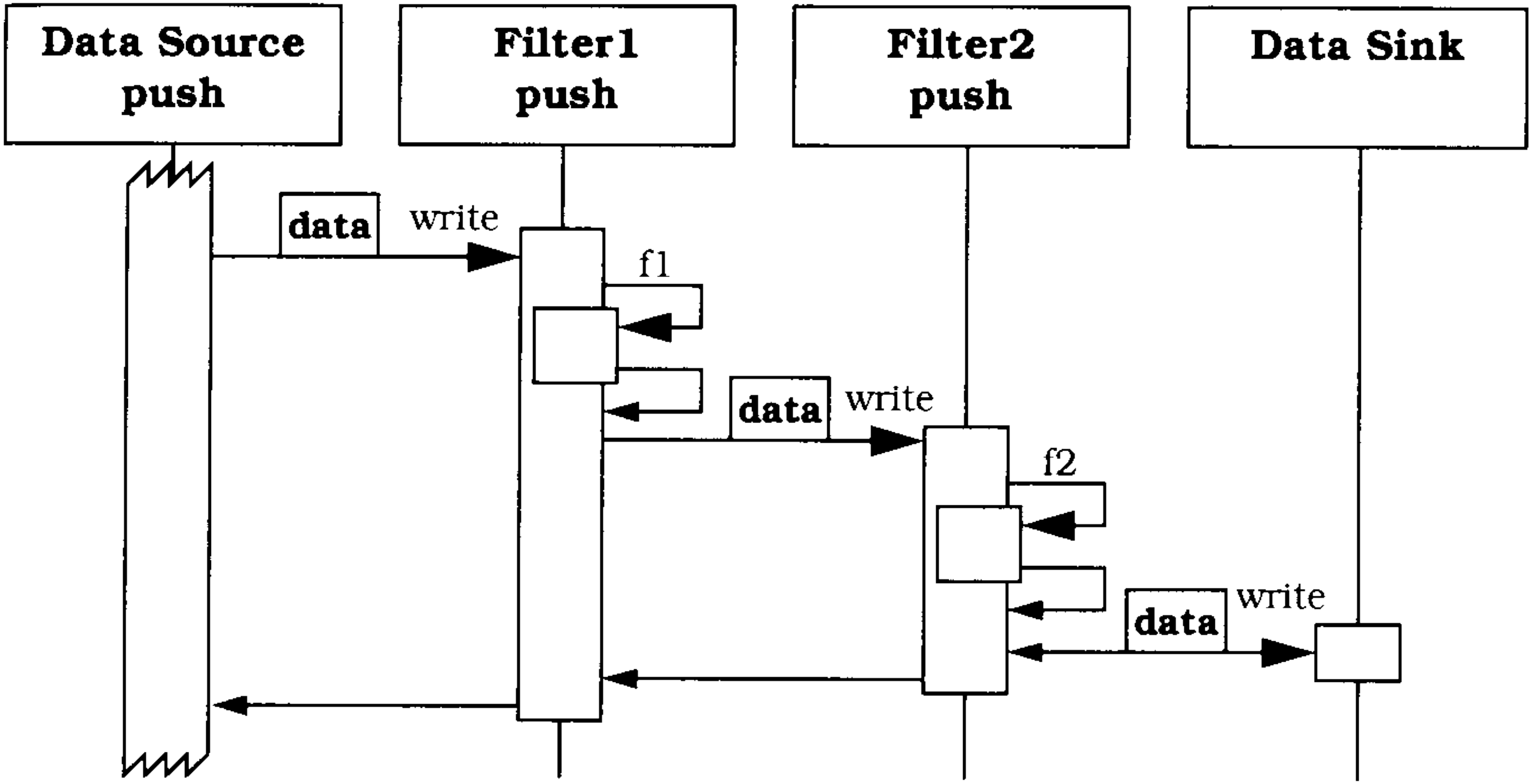
Class Data Source	***Collaborators*** • Pipe
Responsibility • Delivers input to processing pipeline.	

Class Data Sink	***Collaborators*** • Pipe
Responsibility • Consumes output.	

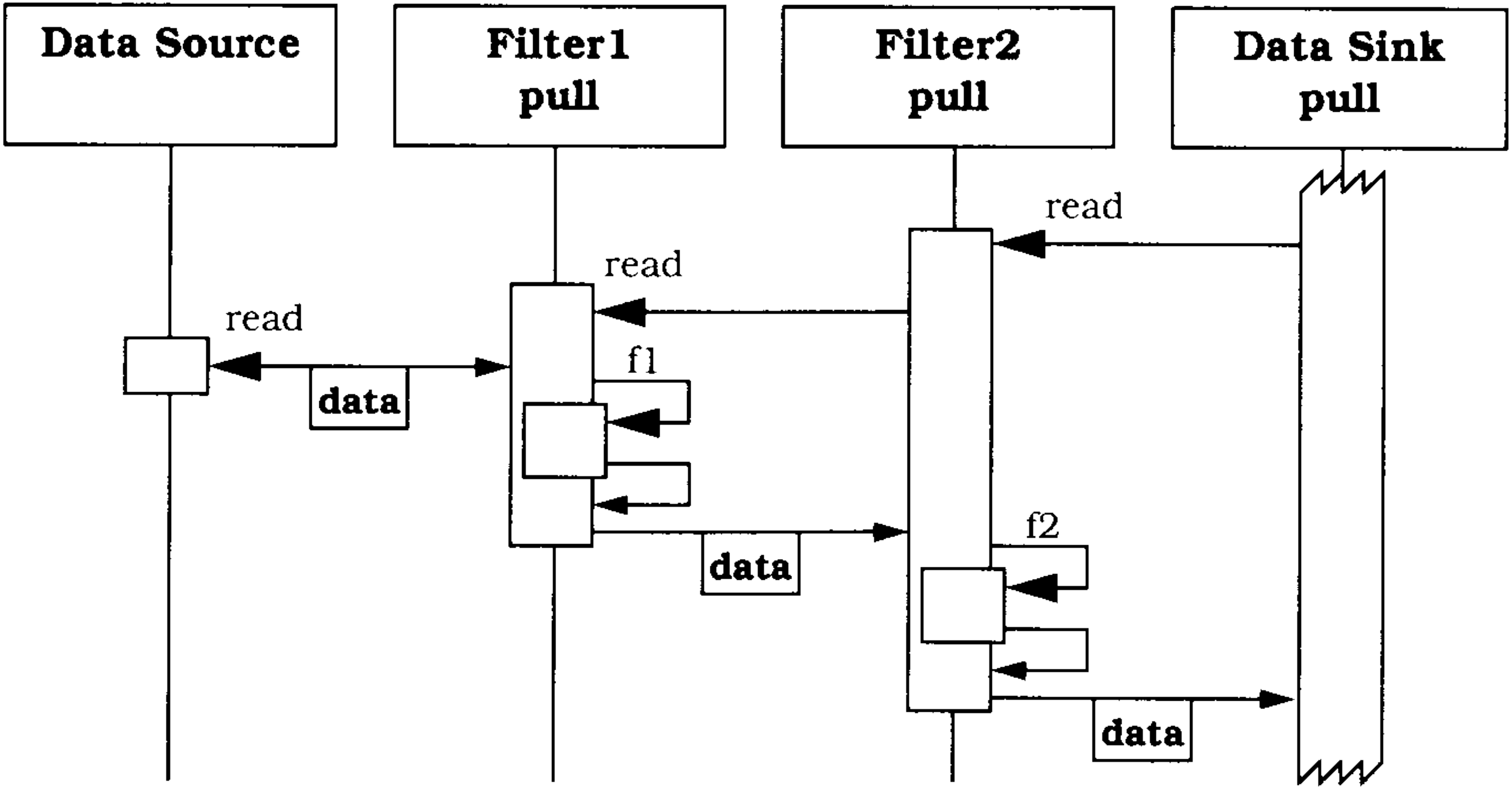
Pipes and Filters



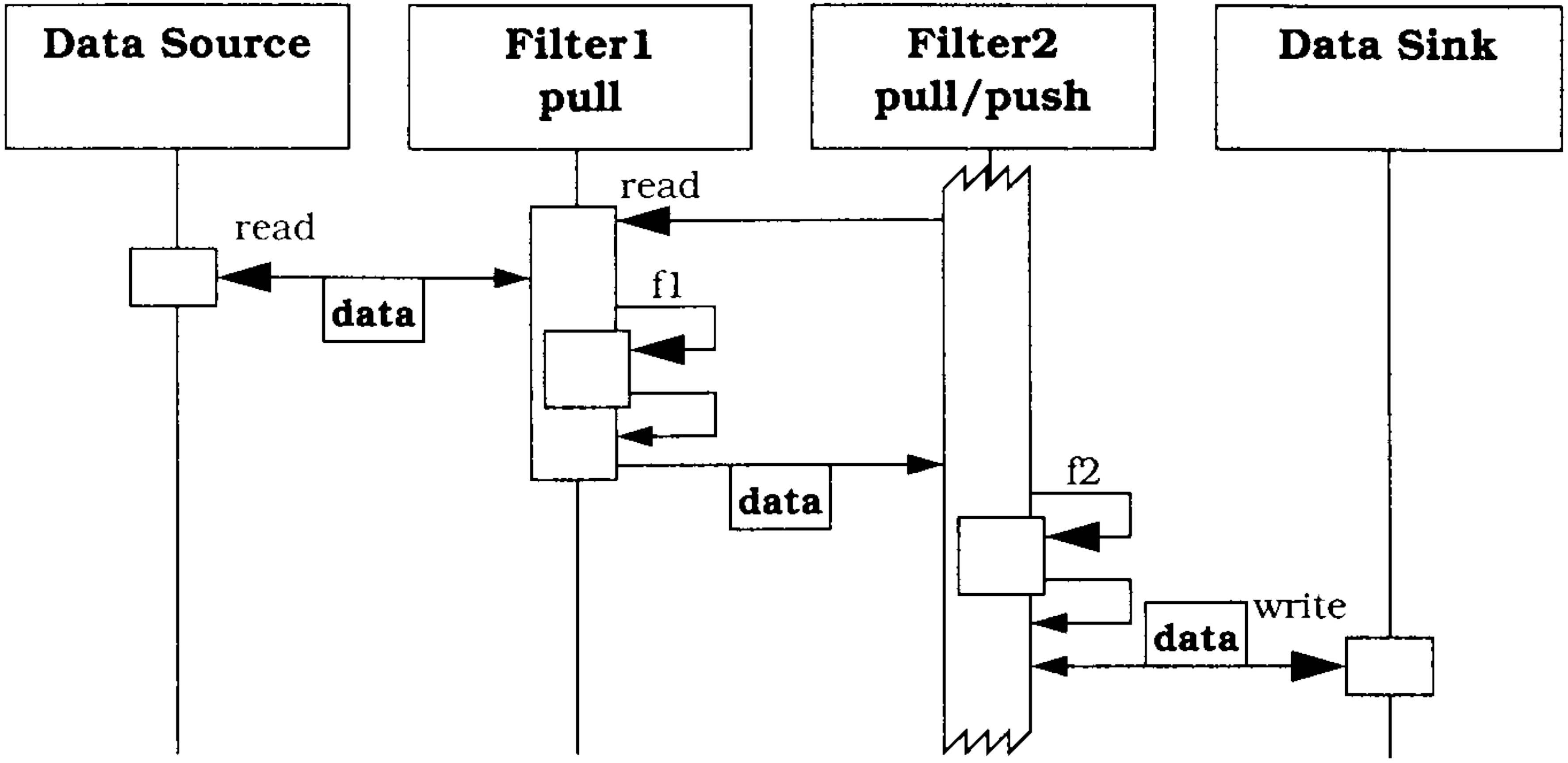
Pipes and Filters: push pipeline



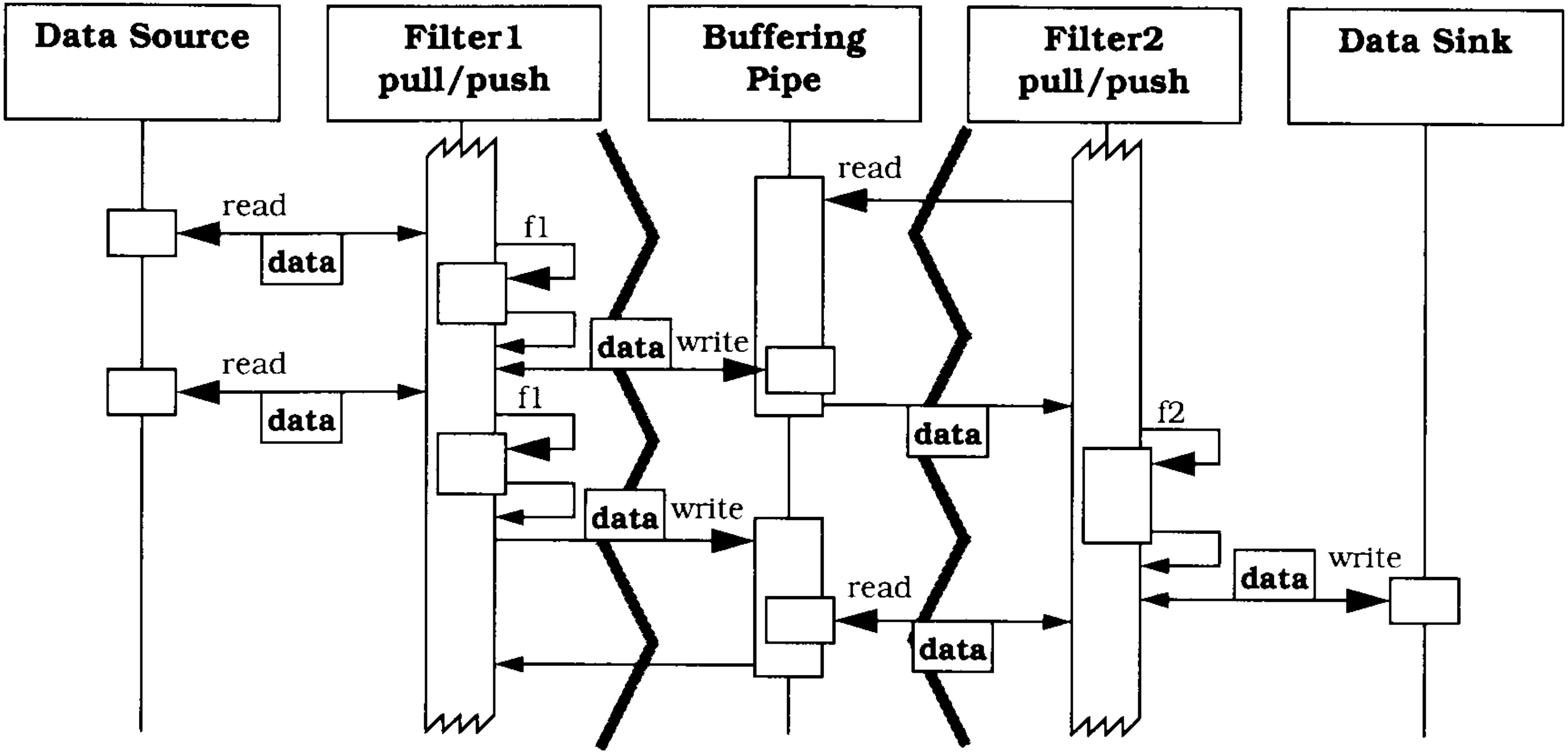
Pipes and Filters: pull pipeline



Pipes and Filters: push-pull pipeline



Pipes and Filters : Threaded Filters



Pipes and Filters: Known Uses

- Unix
- CMS Pipelines (extension IBM mainframes)
- LASSPTools (Numerical Analysis)
  - Graphical input devices (knobs or sliders)
  - Filters for numerical analysis and data extraction
  - Data sinks to produce animation from numerical data streams…
- Khoros : Image recognition…
- WEB !! Servlet !!

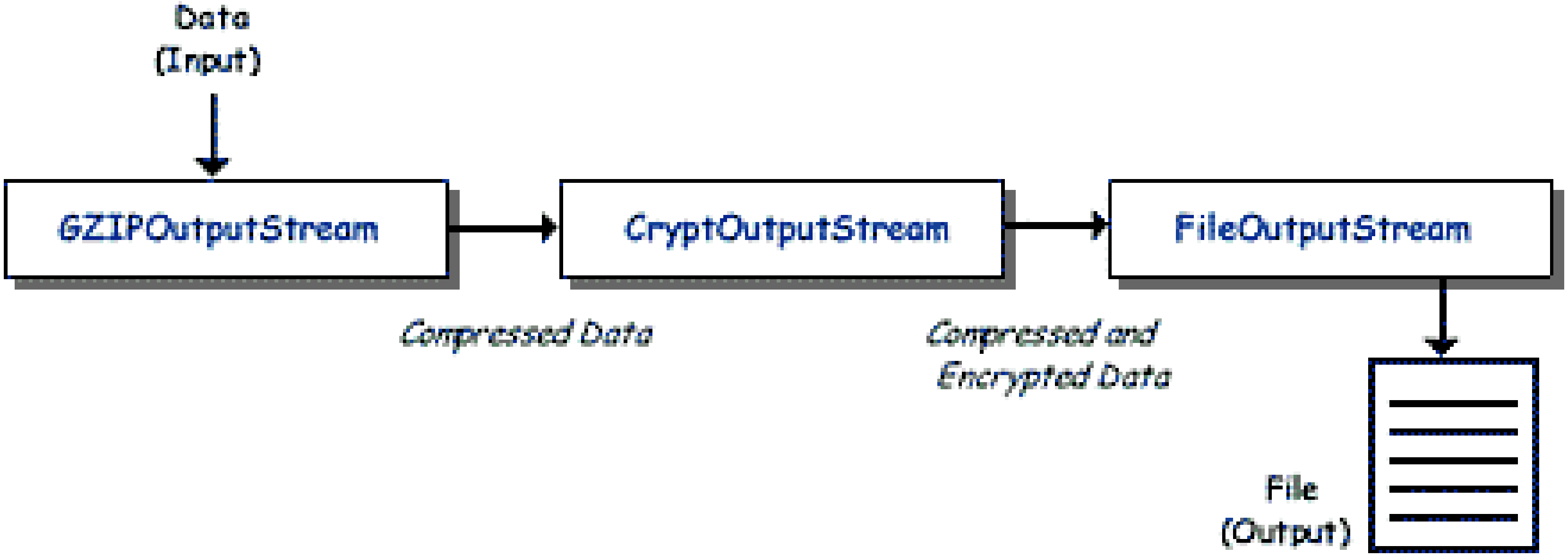
Pipes and Filters benefits

- No intermediate file necessary (but possible)
- Flexibility by filter exchange
- Flexibility by recombination
- Reuse of filter components
- Rapid prototyping of pipeline
- Efficiency by parallel processing

Pipes and Filters Liabilities

- Sharing state information is expensive or inflexible
- Efficiency gain by parallel processing is often an illusion
  - Cost of data transfer, filters that consume all data before one output, context switch on one computer, synchronization of filters via pipes
- Data transformation overhead
- Error Handling

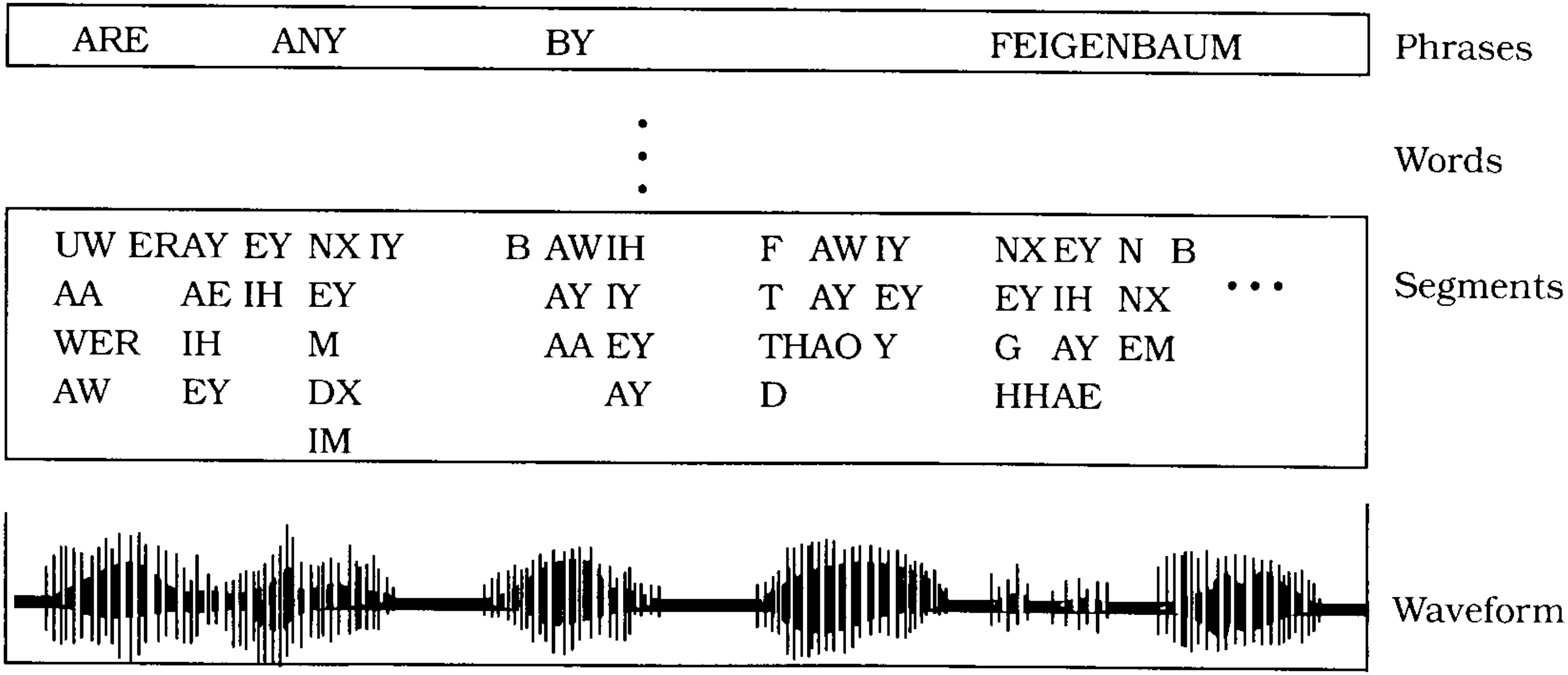
[Sun Developpers]



Blackboard

The Blackboard architectural pattern is 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.

Blackboard Example



ARE ANY BY FEIGENBAUM

Phrases

Words

UW ERAY EY NX IY B AW IH F AW IY NX EY N B

AA AE IH EY AY IY T AY EY EY IH NX . . .

WER IH M AA EY THAO Y G AY EM

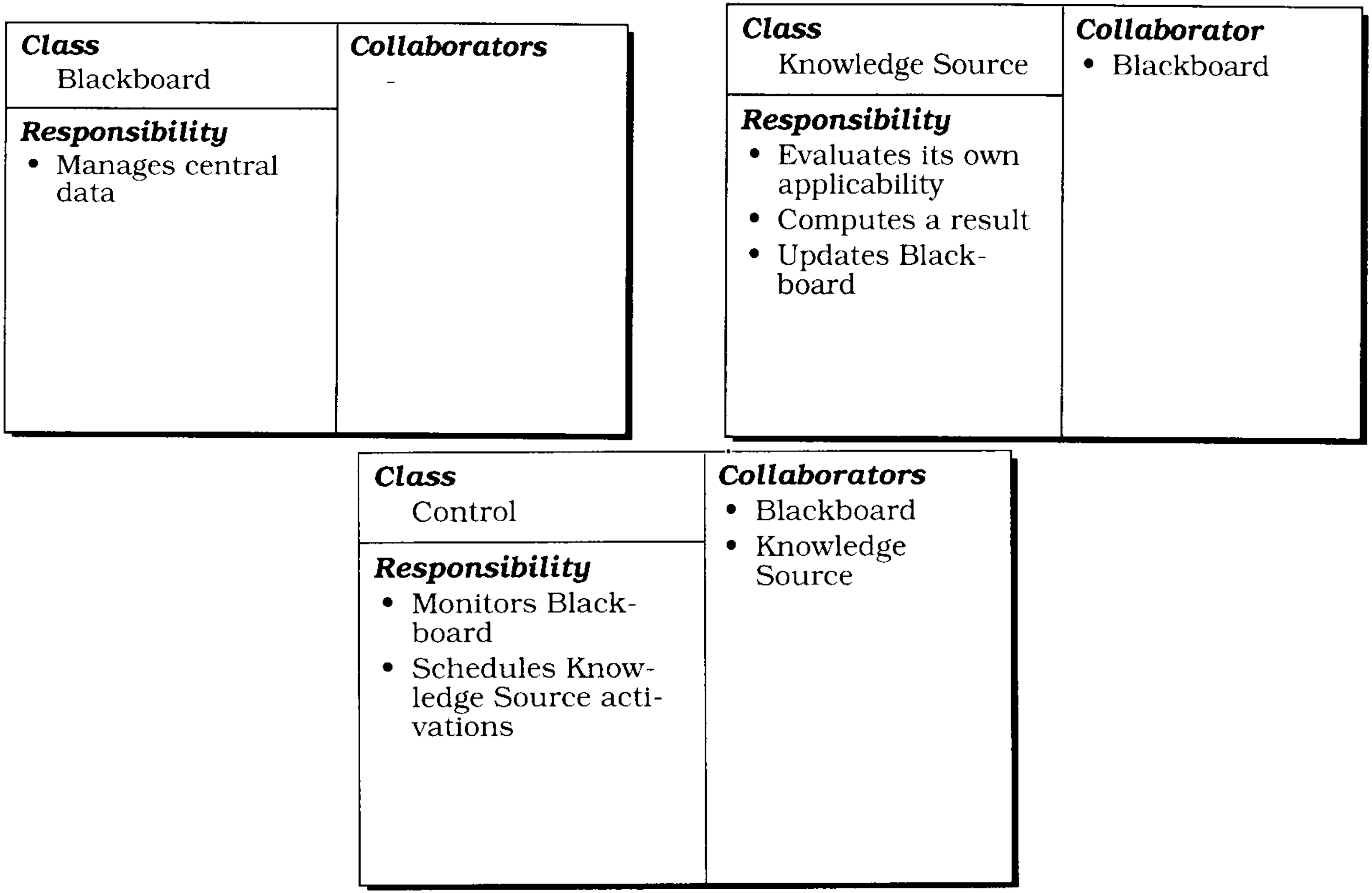
AW EY DX AY D HHAE

IM

Segments

Waveform

Blackboard Structure

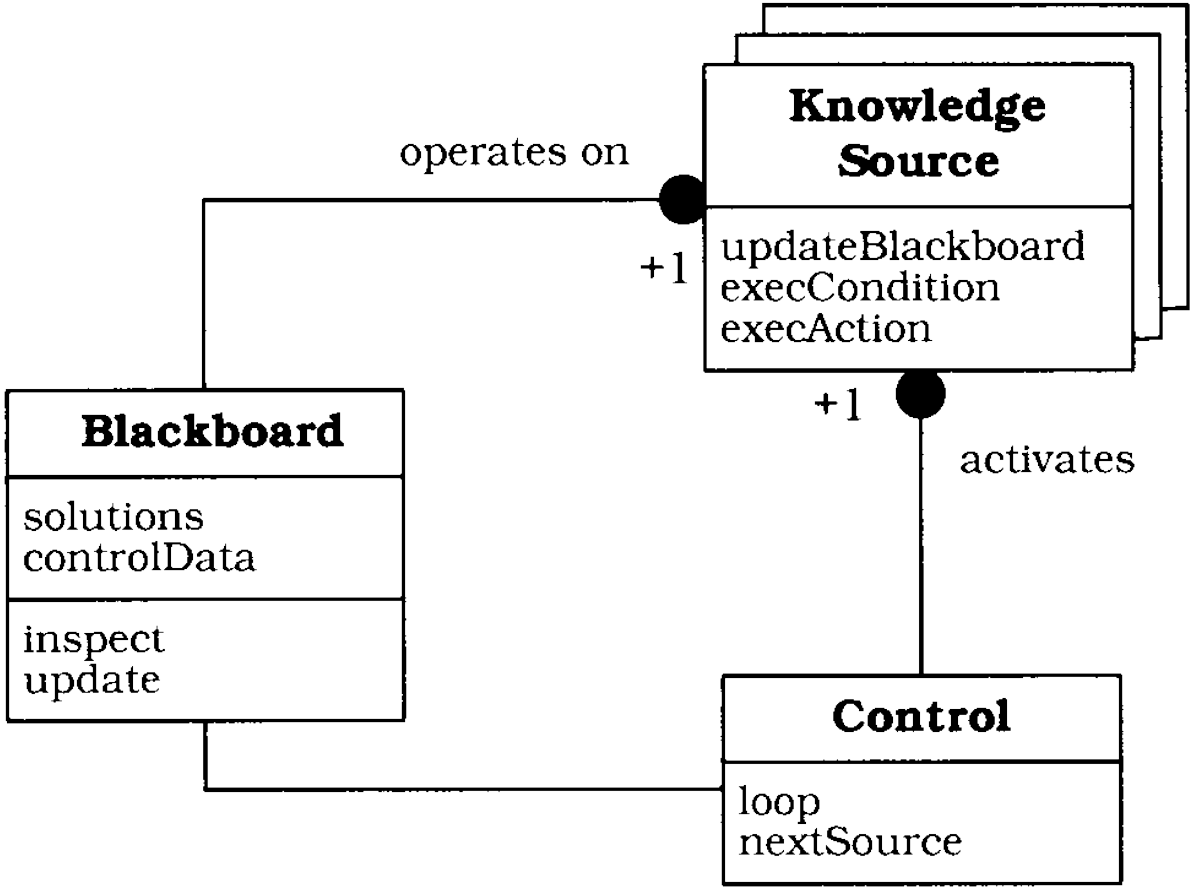


Class Blackboard	***Collaborators*** -
Responsibility • Manages central data	

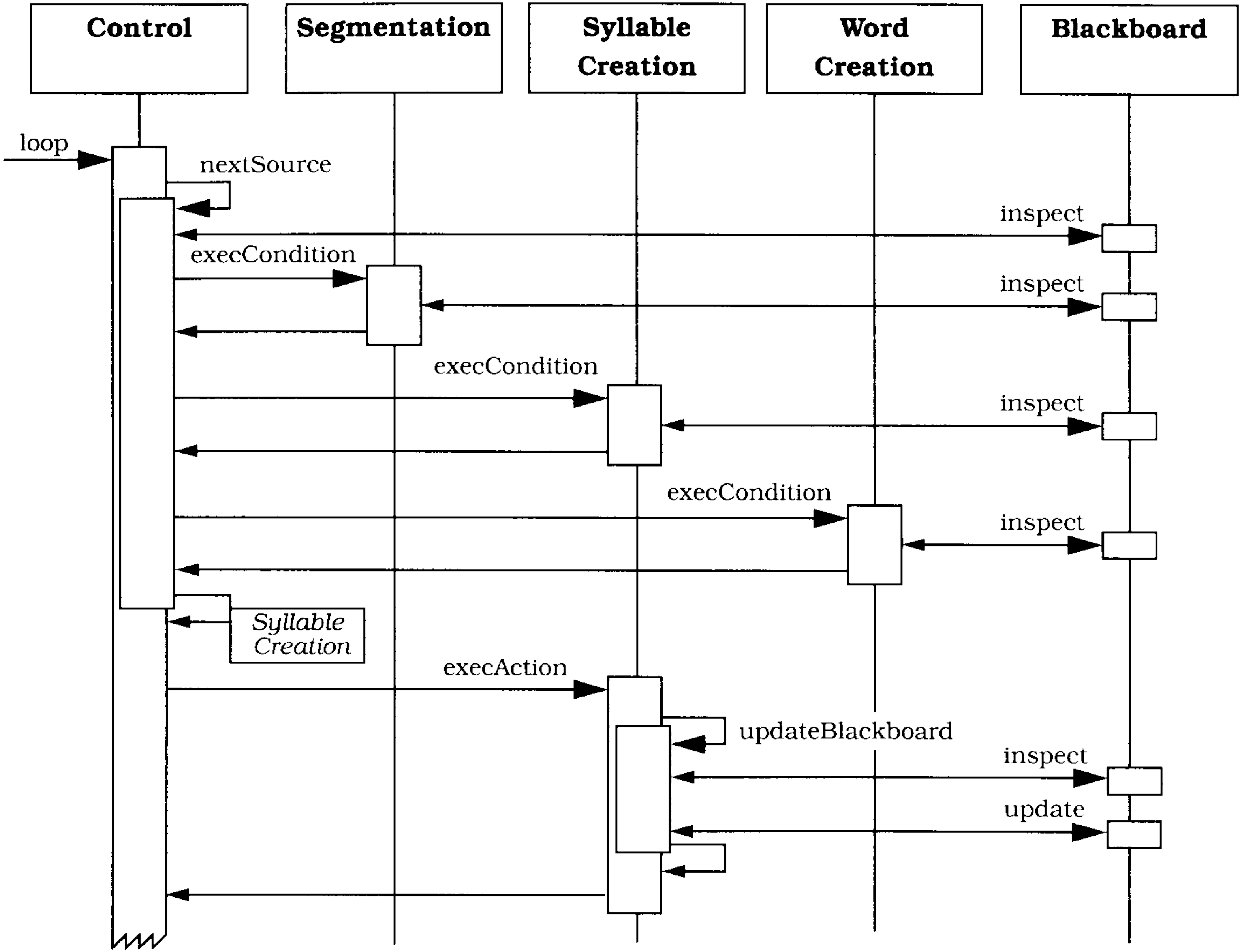
Class Knowledge Source	***Collaborator*** • Blackboard
Responsibility • Evaluates its own applicability • Computes a result • Updates Blackboard	

Class Control	***Collaborators*** • Blackboard • Knowledge Source
Responsibility • Monitors Blackboard • Schedules Knowledge Source activations	

Blackboard Structure



Blackboard Structure



Blackboard Variants

- Production System (OPS Language)
  - – Blackboard : working memory
  - – Knowledge source: Condition-action rules
  - – Control: conflict resolution module.
- Repository:
  - – blackboard: Data,
  - – Application program: knowledge source.
  - – Control: user input, external program

Blackboard known uses

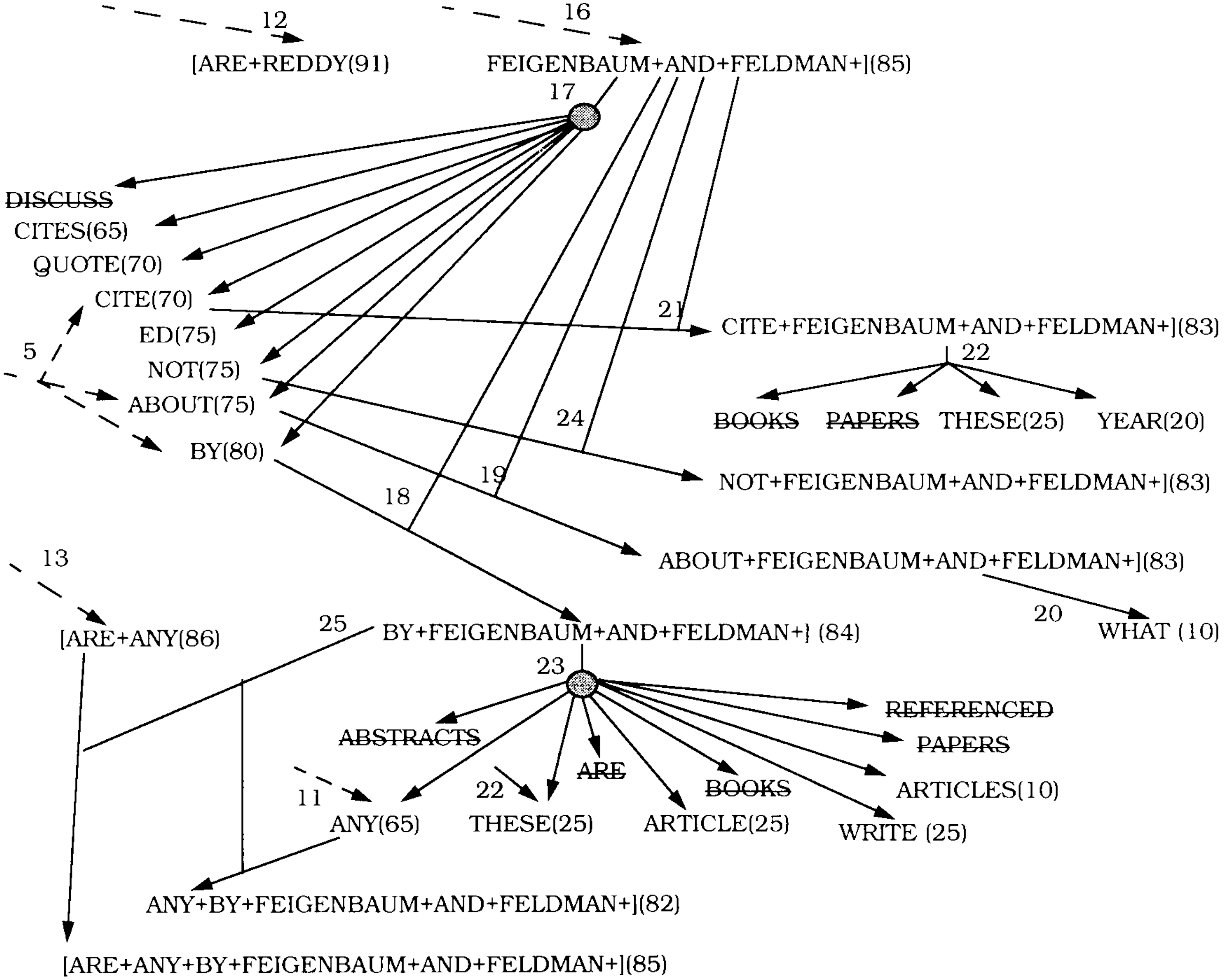
- HEARSAY-II: Speech recognition
- HASP/SIAP: detect enemy submarine
- Crysalis: infer three-dimensional structure of protein molecule from X-Ray diffraction Data.
- Tricero: Aircraft activities. Extend blackboard to distributed computing

Blackboard benefits

- Experimentation: different algo, different control heuristics
- Changeability and maintainability: separation data/control.
- Reusable knowledge source
- Support for Fault tolerance and robustness: Tolerance of noisy data…

Blackboard Liabilities

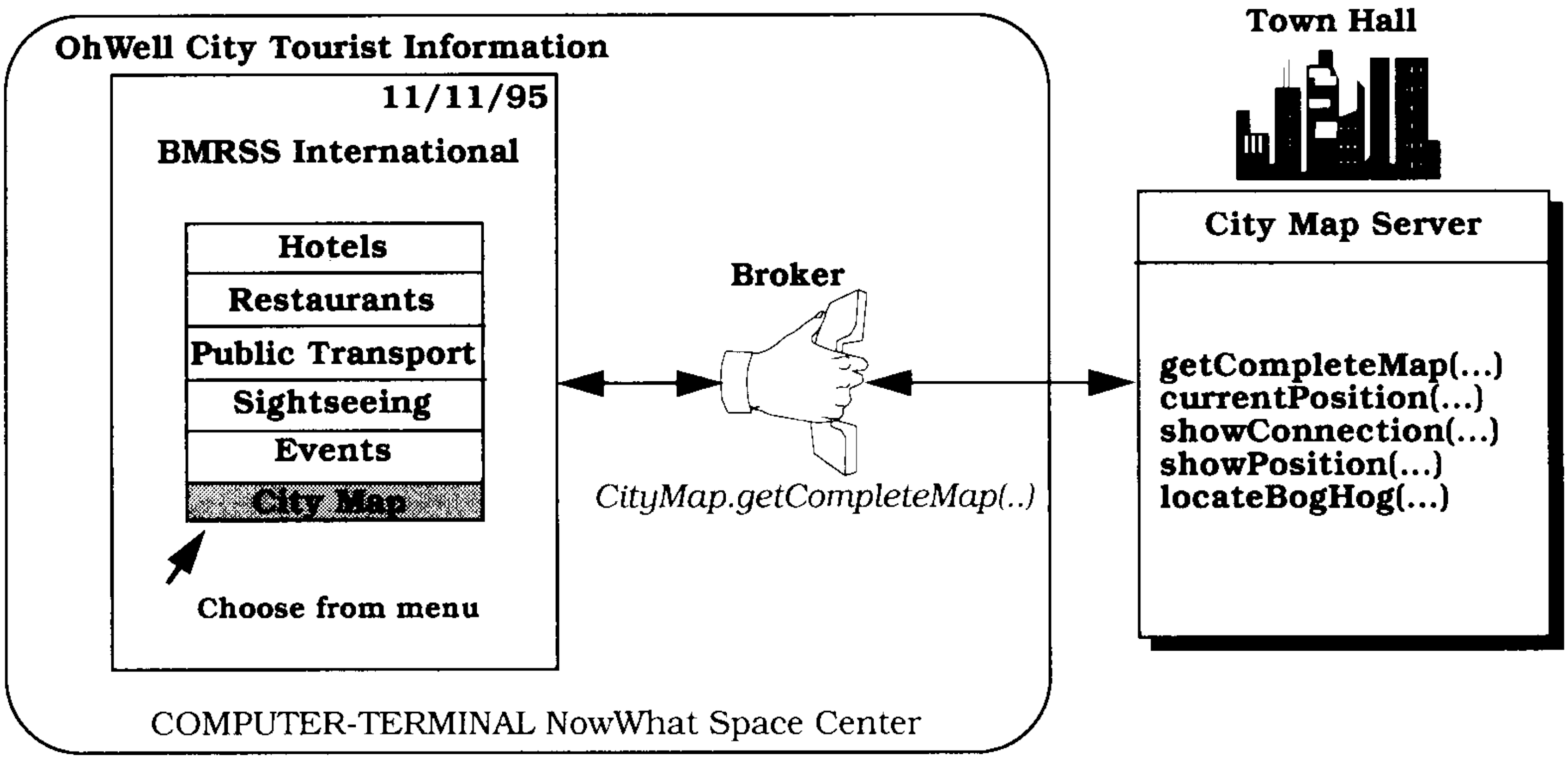
- Difficulty of testing: no deterministic algo
- No good solution is guaranteed.
- Difficulty of establishing a good control strategy
- Low efficiency: (rejecting wrong hypothesis)
- High development effort : trial-and-error programming
- No support for parallelism


12
[ARE+REDDY(91)
16
FEIGENBAUM+AND+FELDMAN+](85)
17
~~DISCUSS~~
CITES(65)
QUOTE(70)
CITE(70)
ED(75)
5
NOT(75)
ABOUT(75)
BY(80)
21
CITE+FEIGENBAUM+AND+FELDMAN+](83)
22
~~BOOKS~~
~~PAPERS~~
THESE(25)
YEAR(20)
24
NOT+FEIGENBAUM+AND+FELDMAN+](83)
19
18
ABOUT+FEIGENBAUM+AND+FELDMAN+](83)
20
WHAT (10)
13
[ARE+ANY(86)
25
BY+FEIGENBAUM+AND+FELDMAN+} (84)
23
~~REFERENCED~~
~~PAPERS~~
~~ABSTRACTS~~
~~ARE~~
~~BOOKS~~
ARTICLES(10)
11
22
ANY(65)
THESE(25)
ARTICLE(25)
WRITE (25)
ANY+BY+FEIGENBAUM+AND+FELDMAN+](82)
[ARE+ANY+BY+FEIGENBAUM+AND+FELDMAN+](85)

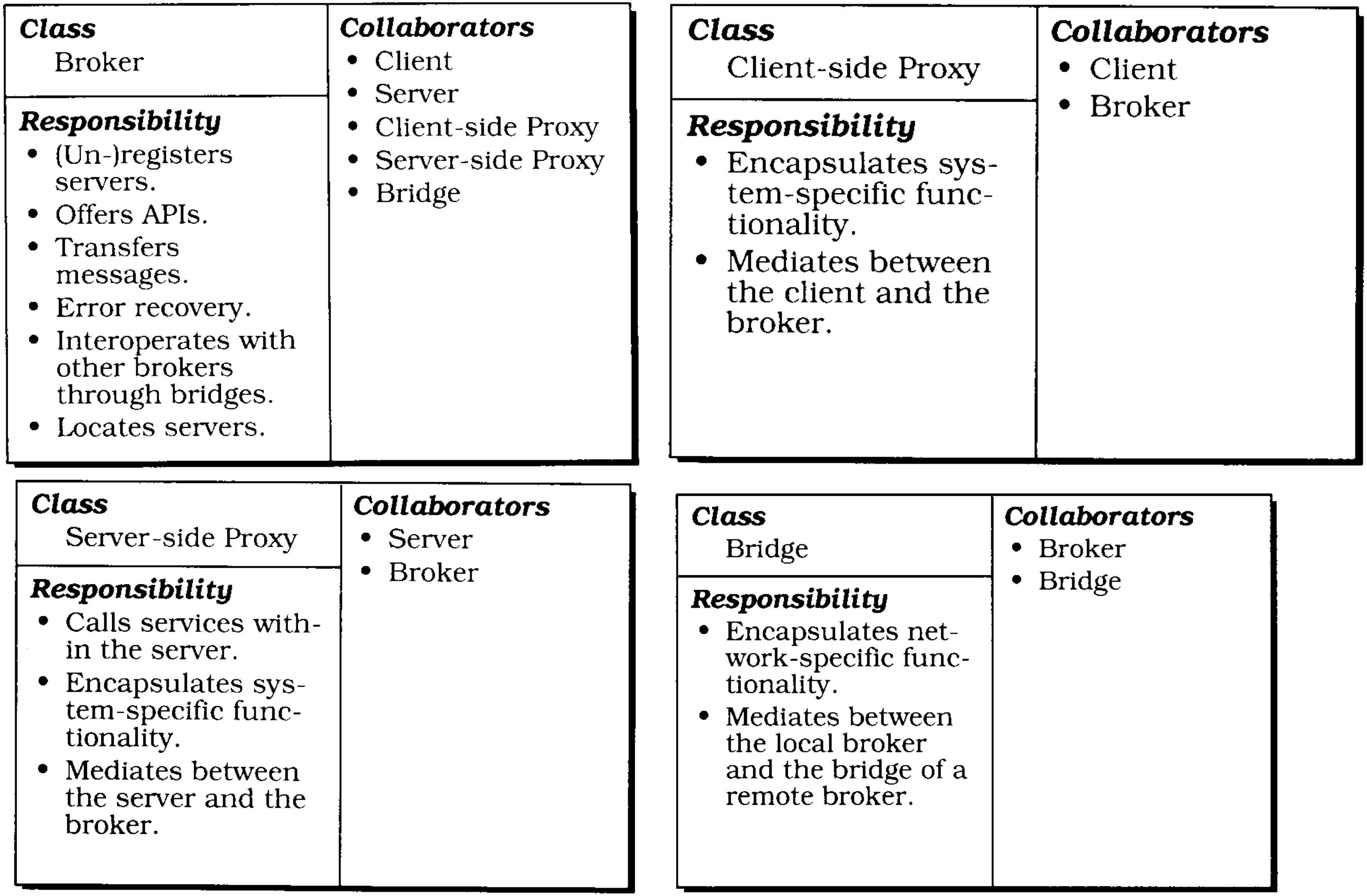
Broker

- Used to structure distributed software systems with decoupled components that interact by remote service invocation.
- A broker component is responsible for coordinating communication, such as forwarding request, as well as for transmitting result and exception.

Broker example



Broker structure



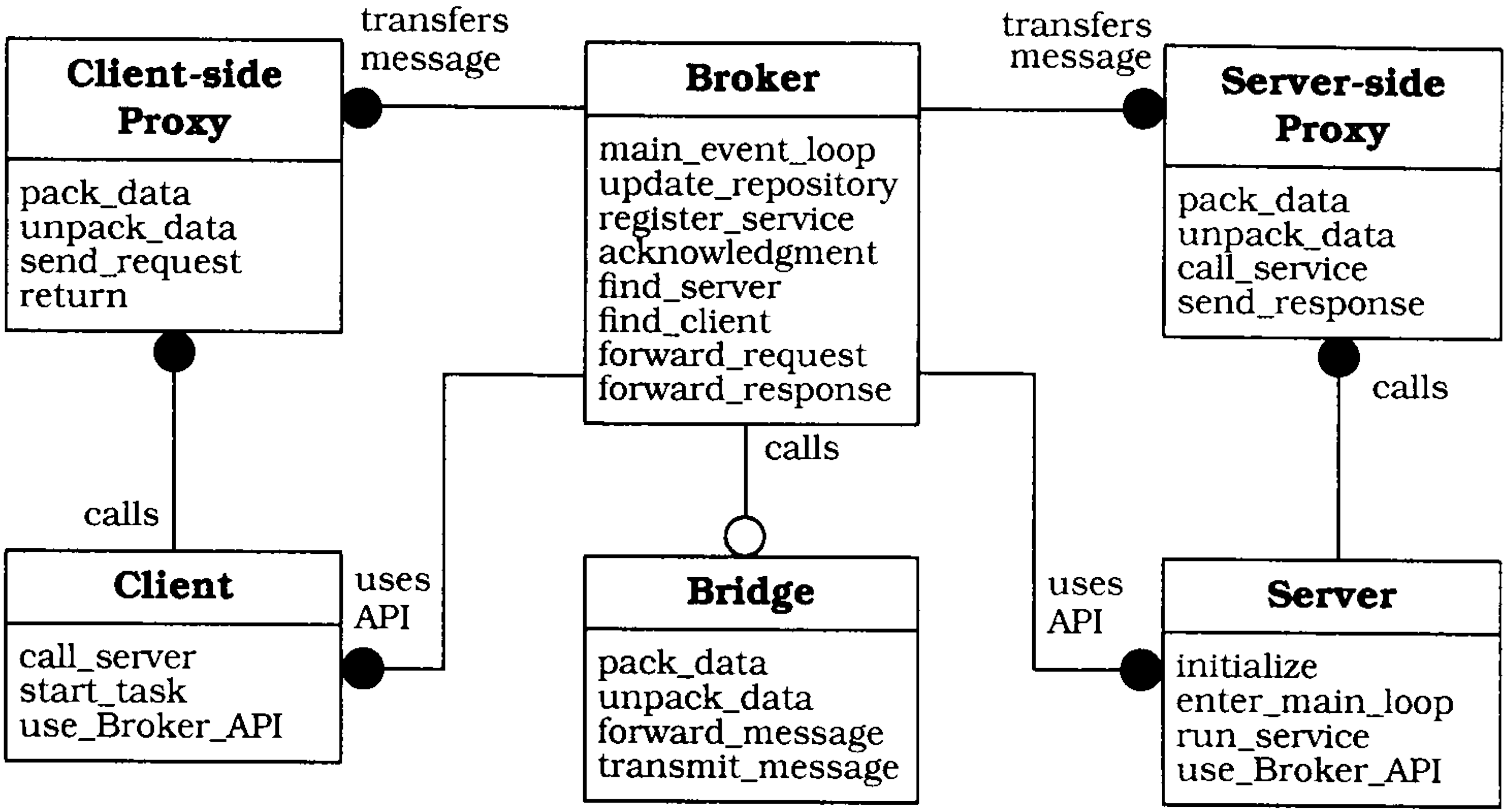
Class Broker	***Collaborators***
Responsibility • (Un-)registers servers. • Offers APIs. • Transfers messages. • Error recovery. • Interoperates with other brokers through bridges. • Locates servers.	• Client • Server • Client-side Proxy • Server-side Proxy • Bridge

Class Client-side Proxy	***Collaborators***
Responsibility • Encapsulates system-specific functionality. • Mediates between the client and the broker.	• Client • Broker

Class Server-side Proxy	***Collaborators***
Responsibility • Calls services within the server. • Encapsulates system-specific functionality. • Mediates between the server and the broker.	• Server • Broker

Class Bridge	***Collaborators***
Responsibility • Encapsulates network-specific functionality. • Mediates between the local broker and the bridge of a remote broker.	• Broker • Bridge

Broker Structure



transfers message

transfers message

Client-side Proxy

pack_data
unpack_data
send_request
return

Broker

main_event_loop
update_repository
register_service
acknowledgment
find_server
find_client
forward_request
forward_response

Server-side Proxy

pack_data
unpack_data
call_service
send_response

calls

calls

calls

uses API

uses API

Client

call_server
start_task
use_Broker_API

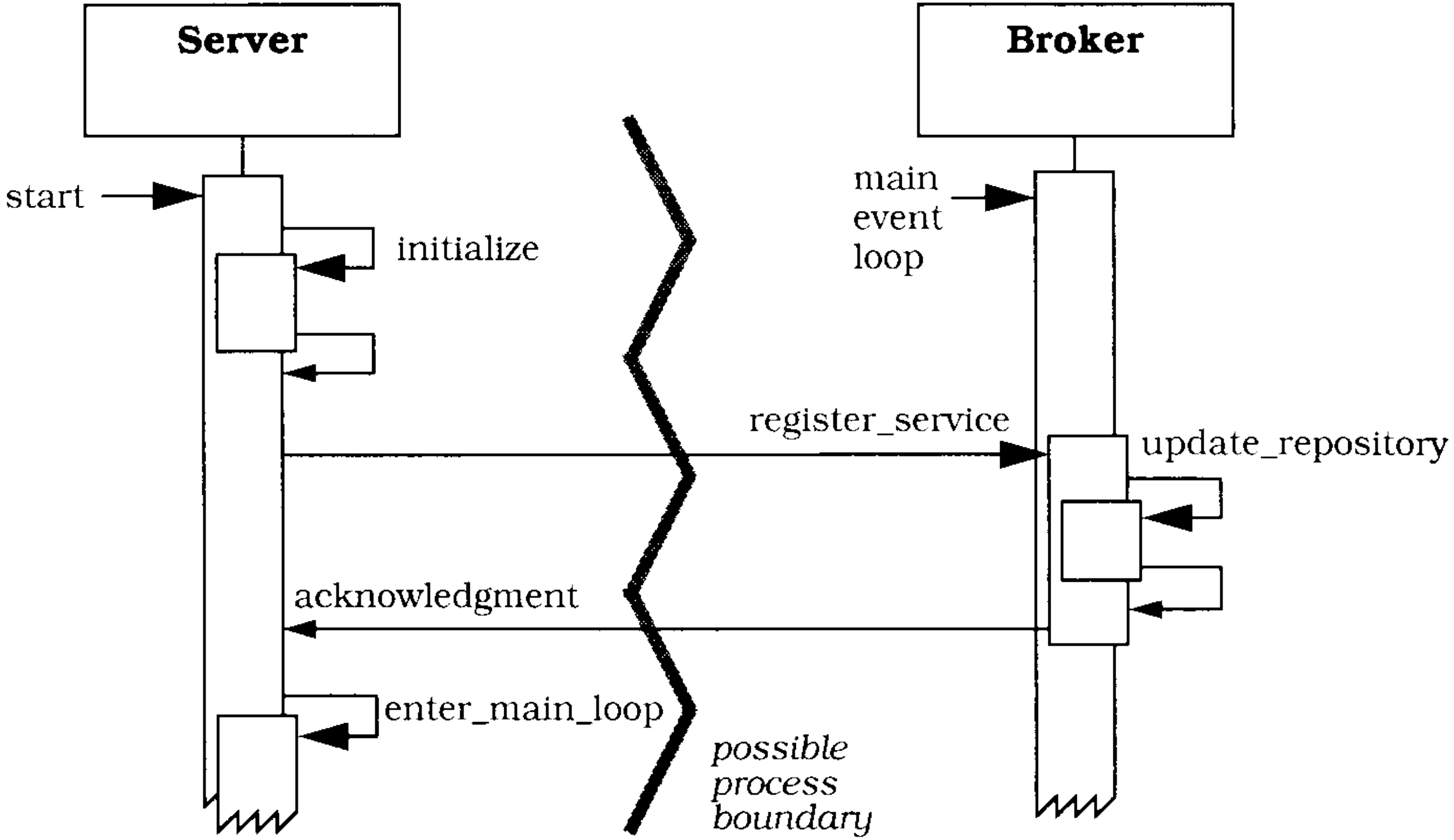
Bridge

pack_data
unpack_data
forward_message
transmit_message

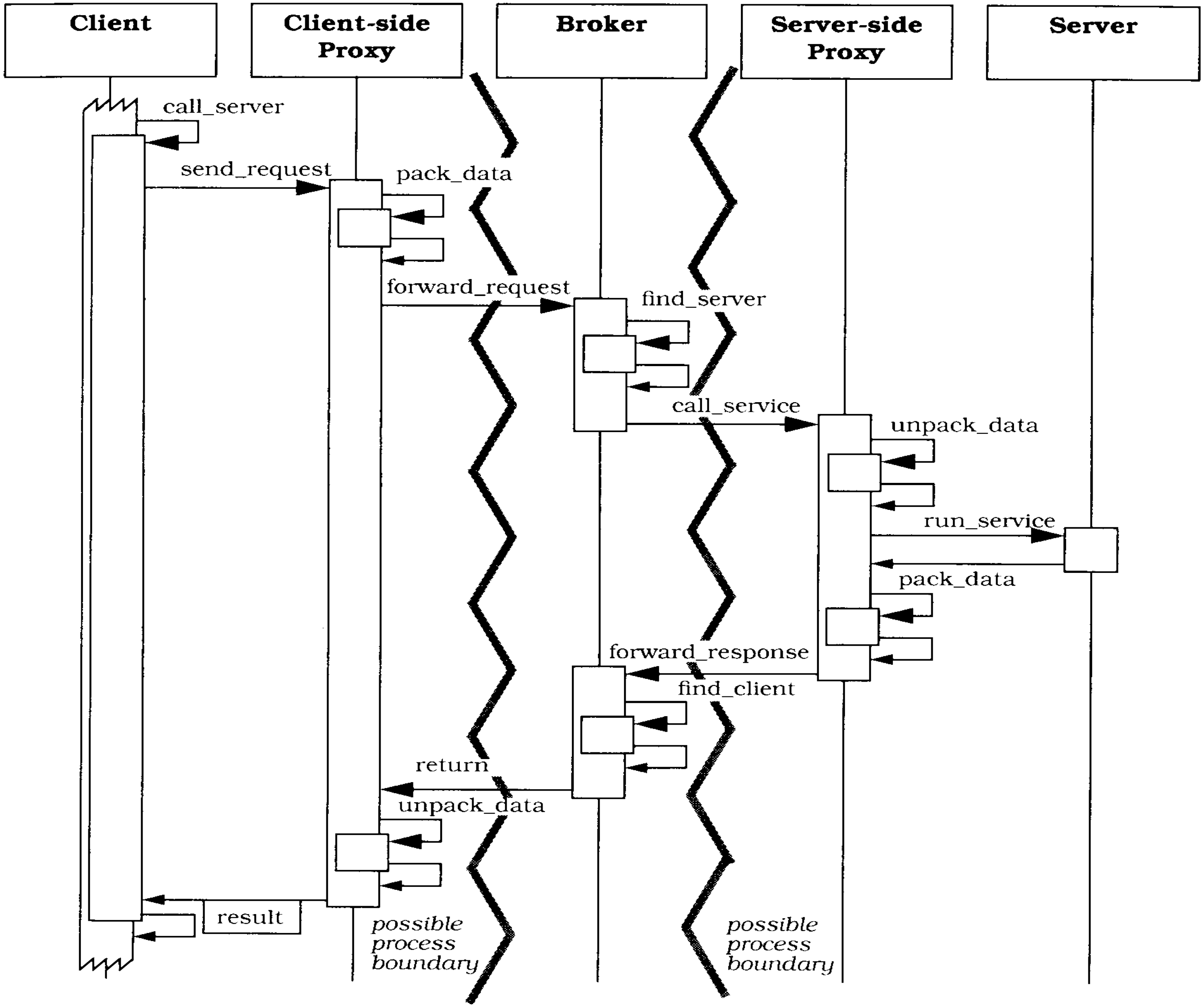
Server

initialize
enter_main_loop
run_service
use_Broker_API

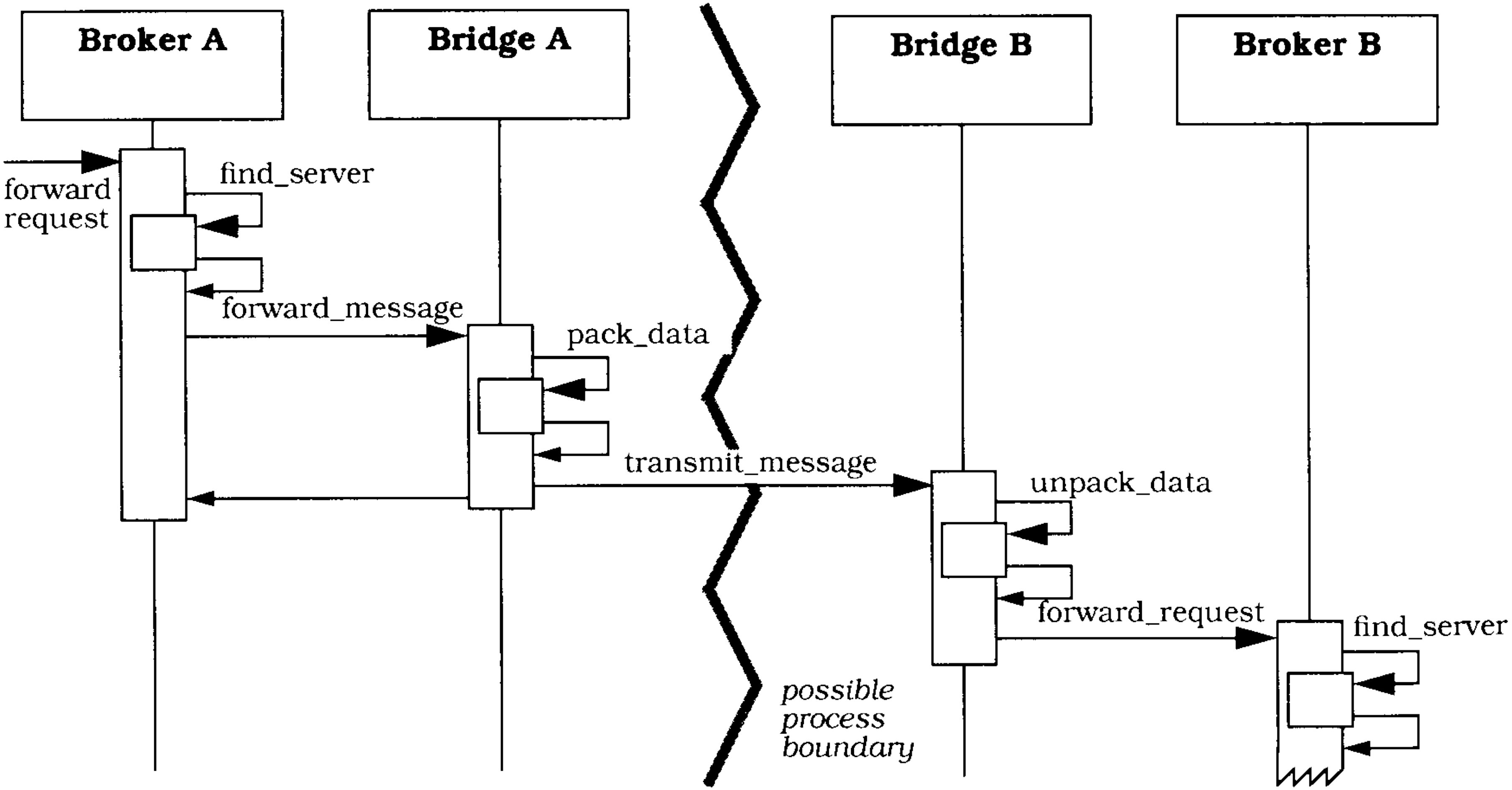
Broker Structure



Broker Structure



Broker Structure



Broker Variants

- Direct Communication Broker System:
  - Direct link to server
- Message Passing Broker System
  - Focus on transmission of data. Type of the message determine the behavior of the broker…
- Trader System :
  - service identifiers are used to access server functionality. Request can be forwarded to more than one server…
- Callback broker system: event driven…

Known Uses

- CORBA
- IBM SOM/DSOM
- Microsoft Ole 2.x
- WWW
- ATM-P: Message passing broker. Telecommunication switching system based on ATM.

Broker benefits

- Location transparency
- Changeability and extensibility of components
- Portability of a broker system (Layered)
- Interoperability between brokers (bridge)
- Reusability (of services)

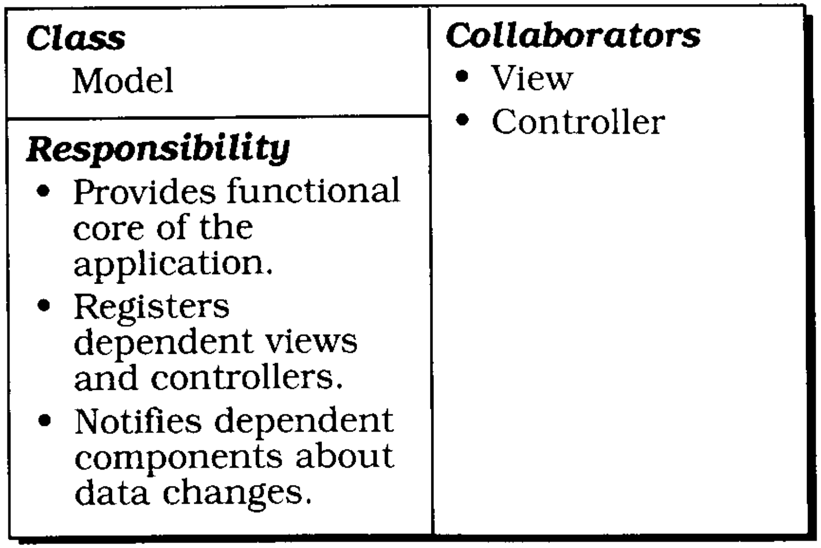
Broker Liabilities

- Restricted efficiency (indirection layer)
- Lower Fault tolerance: fault a broker or a server… replication of components…
- Testability:
  - Of components (benefits)
  - Of application (liabilities)

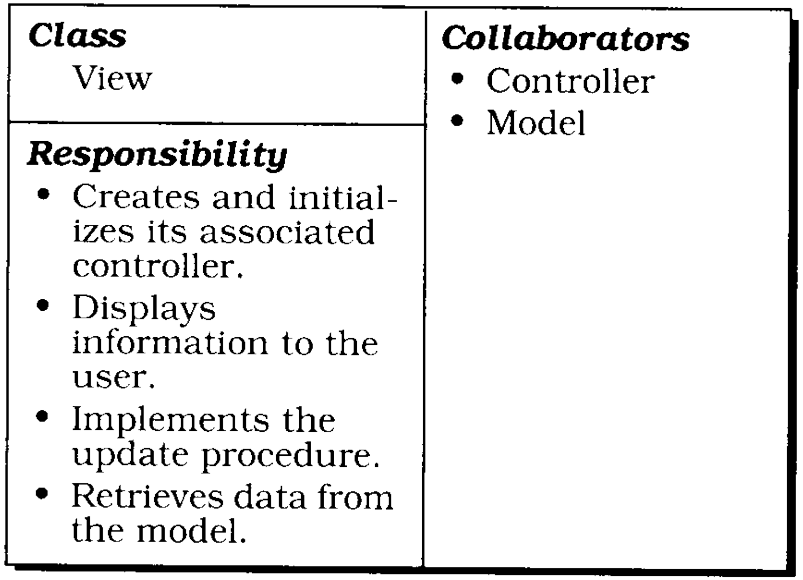
Model-View-Contoler (MVC)

- The model contains the core functionality and data?
- Views display information to the user.
- Controllers handle user input.
- A change propagation mechanism ensure consistency between user interface and the model.

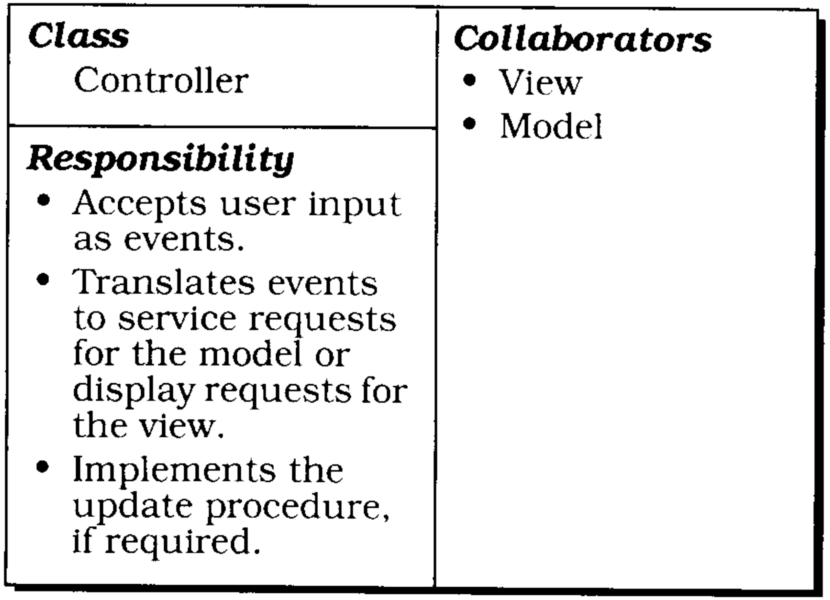
MVC



Class Model	***Collaborators*** • View • Controller
Responsibility • Provides functional core of the application. • Registers dependent views and controllers. • Notifies dependent components about data changes.	

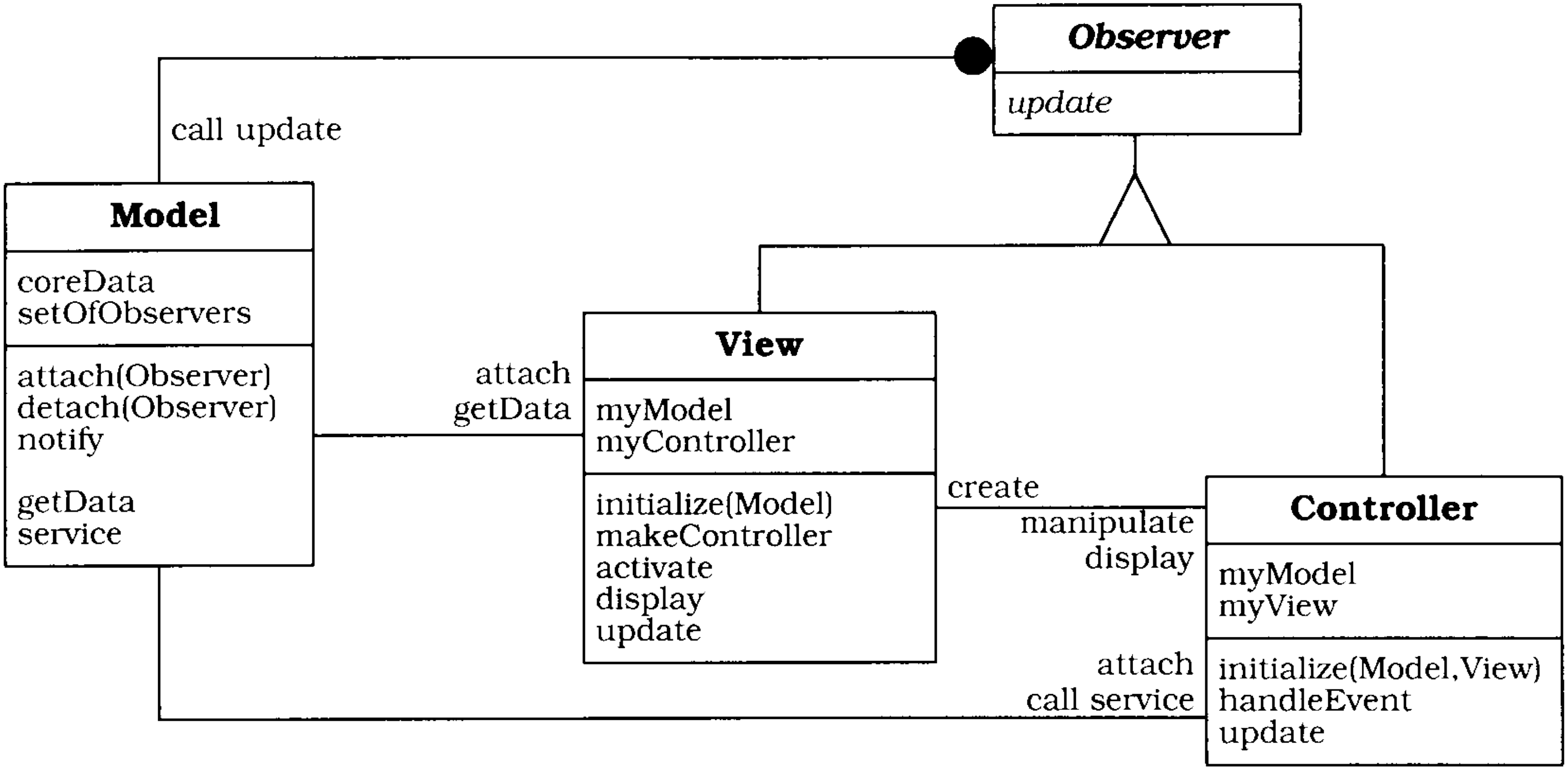


Class View	***Collaborators*** • Controller • Model
Responsibility • Creates and initializes its associated controller. • Displays information to the user. • Implements the update procedure. • Retrieves data from the model.	

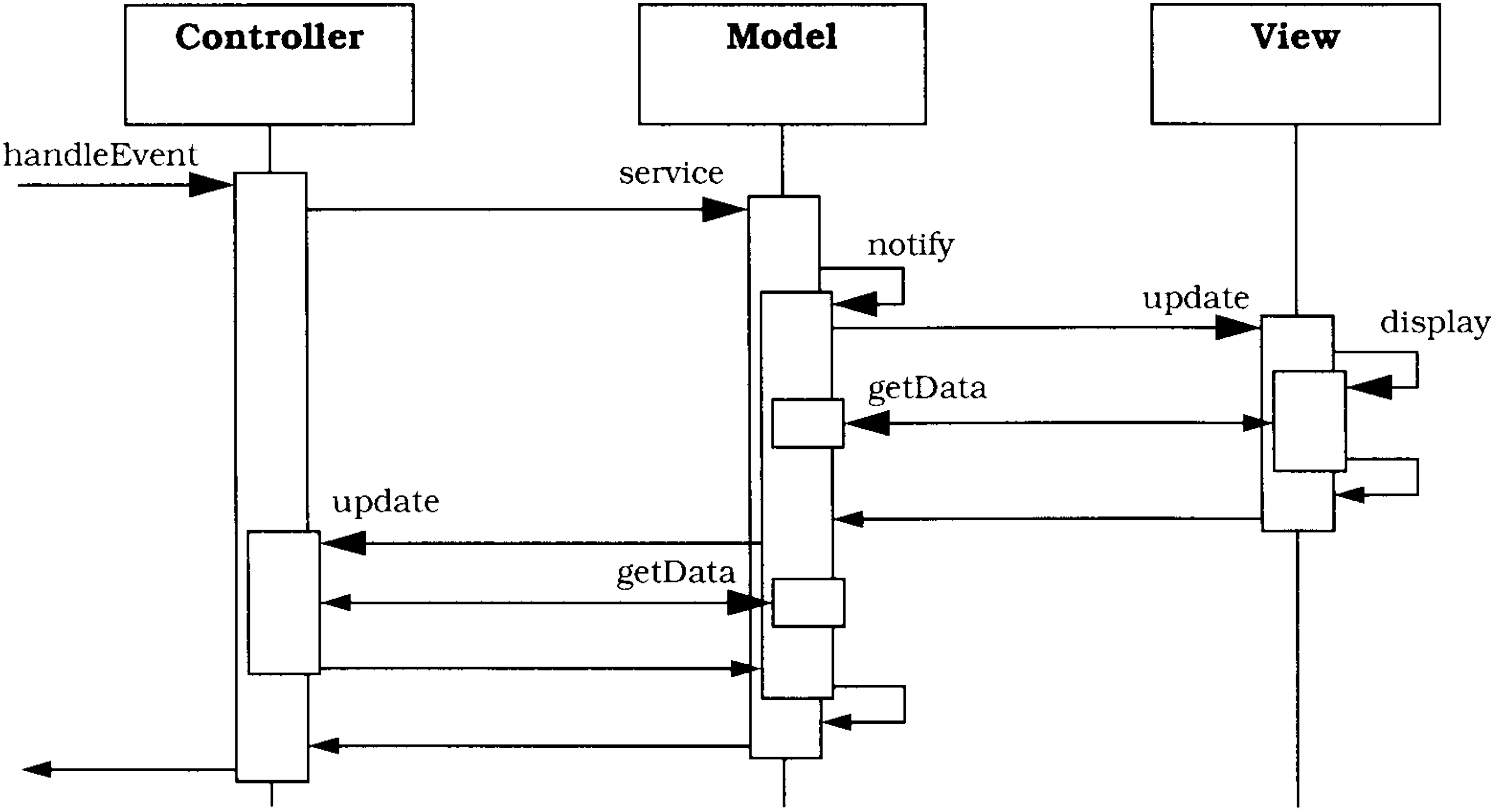


Class Controller	***Collaborators*** • View • Model
Responsibility • Accepts user input as events. • Translates events to service requests for the model or display requests for the view. • Implements the update procedure, if required.	

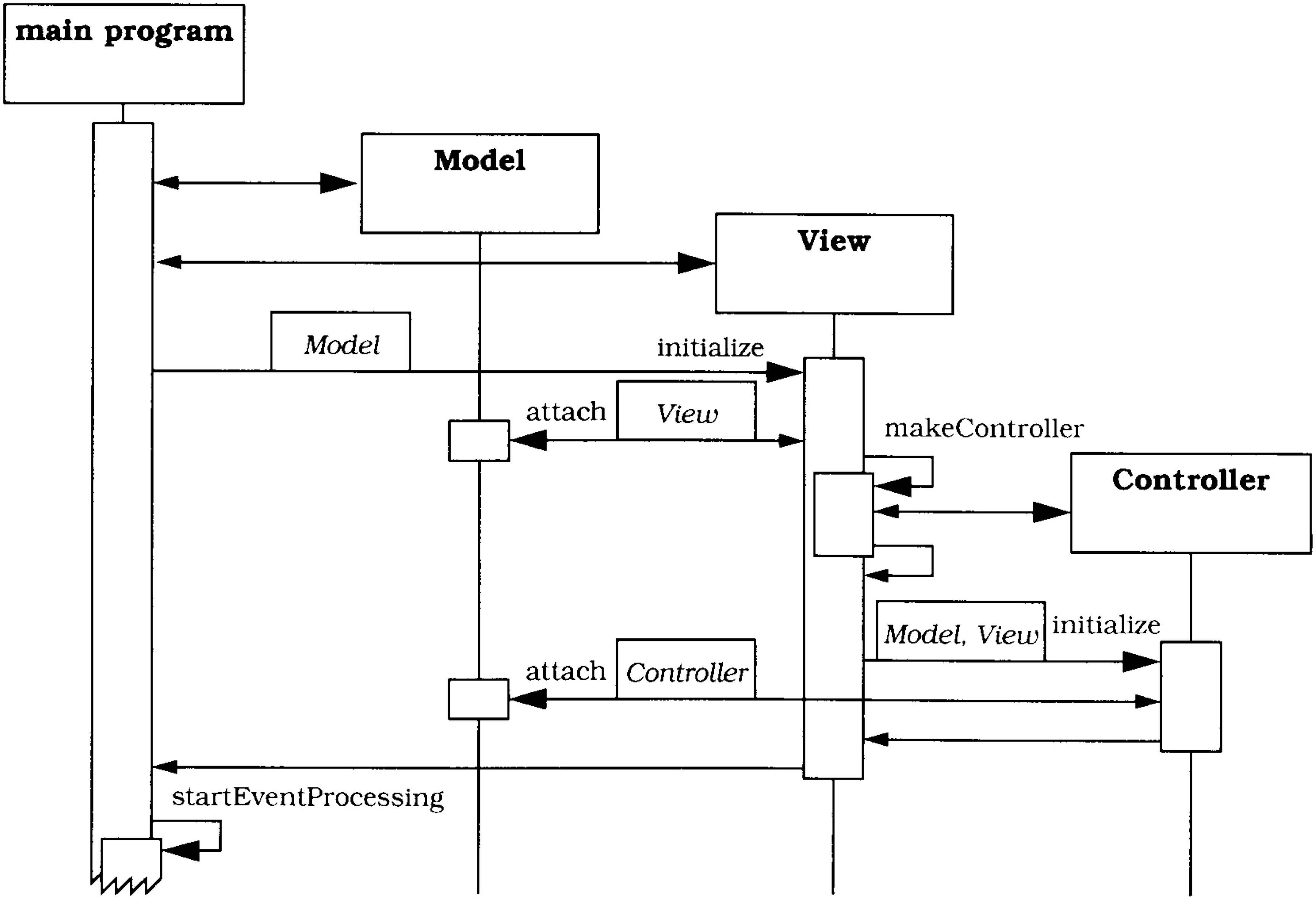
MVC Structure



MVC Structure



MVC Structure



MVC Known Uses

- Smalltalk
- MFC
- ET++: application Framework
- Java/Swing

MVC benefits

- Multiple views of the same model
- Synchronized views: change propagation
- Pluggable views and controllers
- Exchangeability of ‘look and feel’
- Framework potential

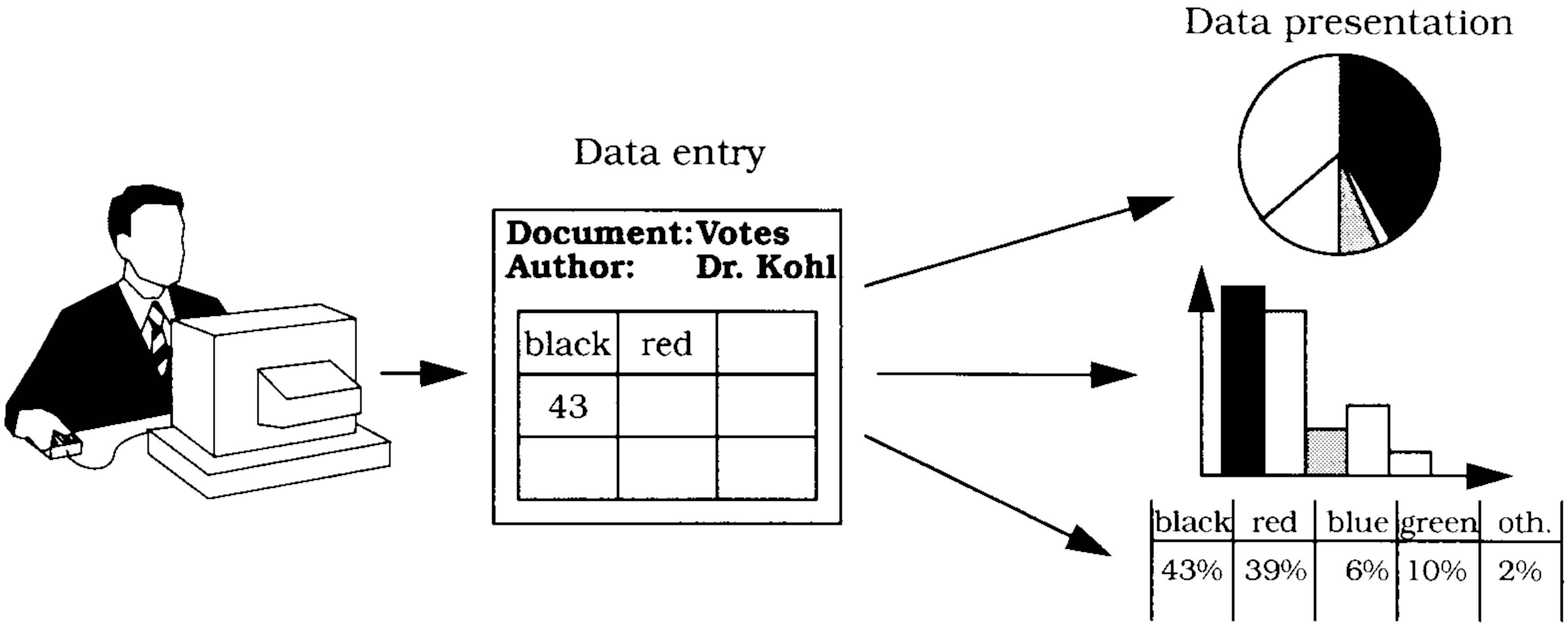
MVC Liabilities

- Increased complexity
- Potential for excessive number of updates
- Intimate connection between view and controller
- Close coupling of views and controllers to a model
- Inefficiency of data access in view
- Inevitability of change to view and controller when porting
- Difficulty of using MVC with modern user-interface tools

Presentation-Abstraction-Control

- PAC define a hierarchy of cooperating agents.
- Each agent consists of three components: presentation, abstraction, control.
- Separates human computer interaction from its functional core and its communication with other agents…

PAC Example



Data entry

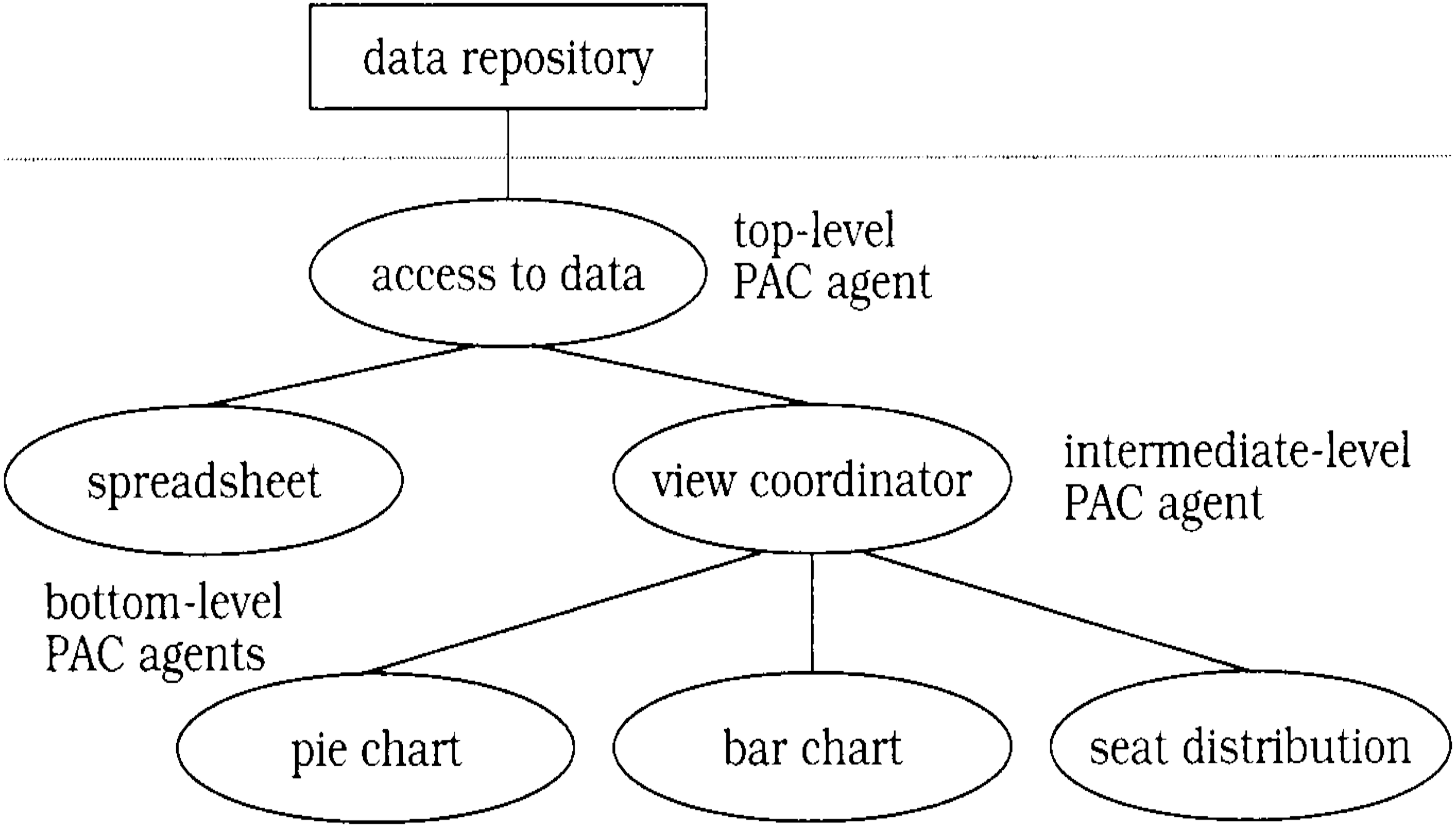
Document: Votes
Author: Dr. Kohl

black	red	
43		

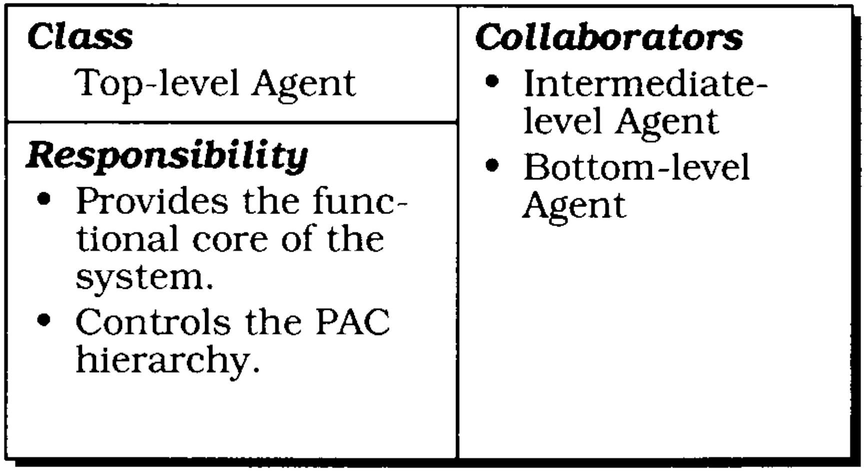
Data presentation

black	red	blue	green	oth.
43%	39%	6%	10%	2%

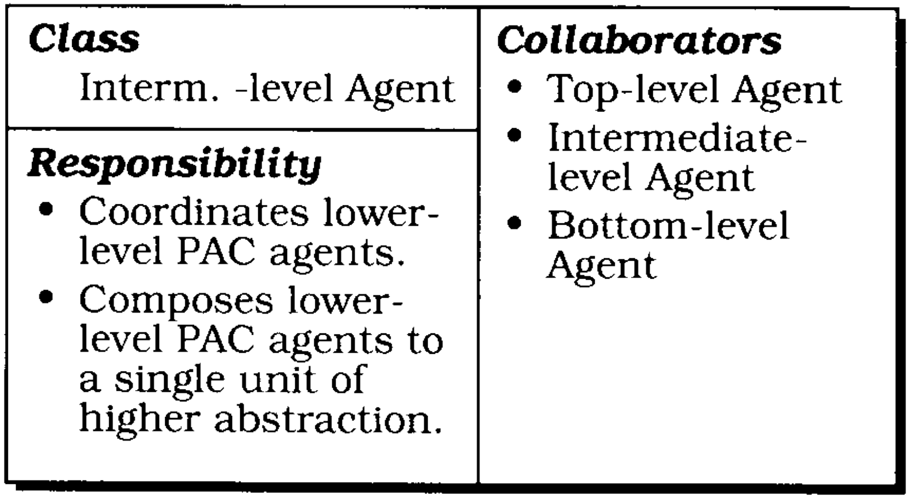
PAC Example


data repository
access to data
top-level PAC agent
spreadsheet
view coordinator
intermediate-level PAC agent
bottom-level PAC agents
pie chart
bar chart
seat distribution

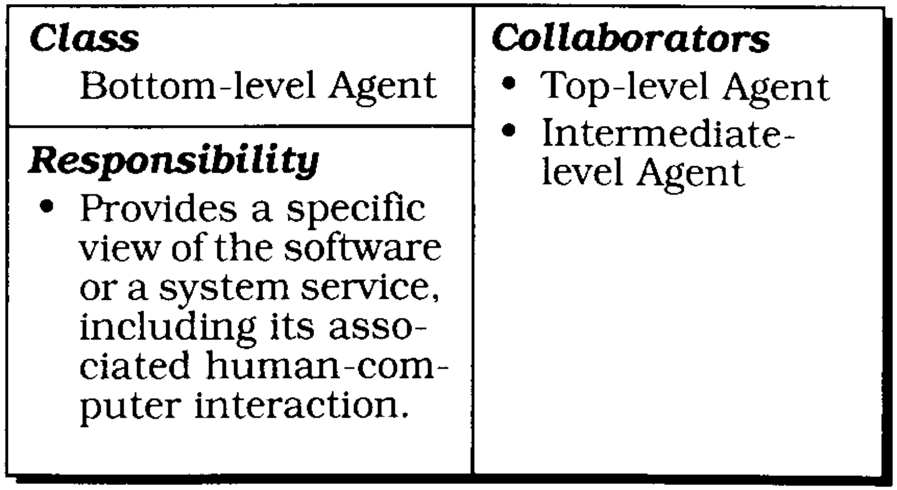
PAC Structure



Class Top-level Agent	***Collaborators*** • Intermediate-level Agent • Bottom-level Agent
Responsibility • Provides the functional core of the system. • Controls the PAC hierarchy.	



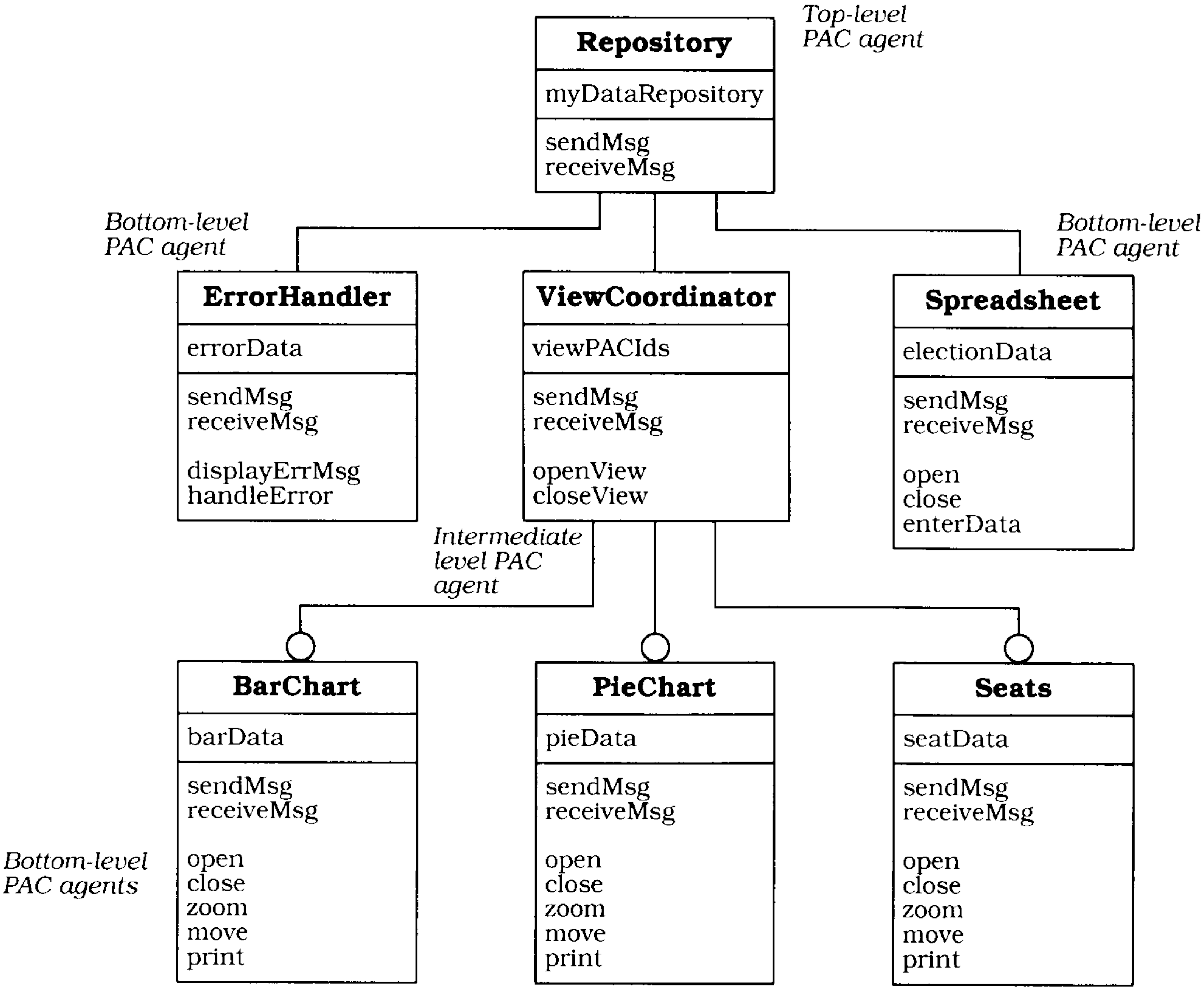
Class Interm. -level Agent	***Collaborators*** • Top-level Agent • Intermediate-level Agent • Bottom-level Agent
Responsibility • Coordinates lower-level PAC agents. • Composes lower-level PAC agents to a single unit of higher abstraction.	



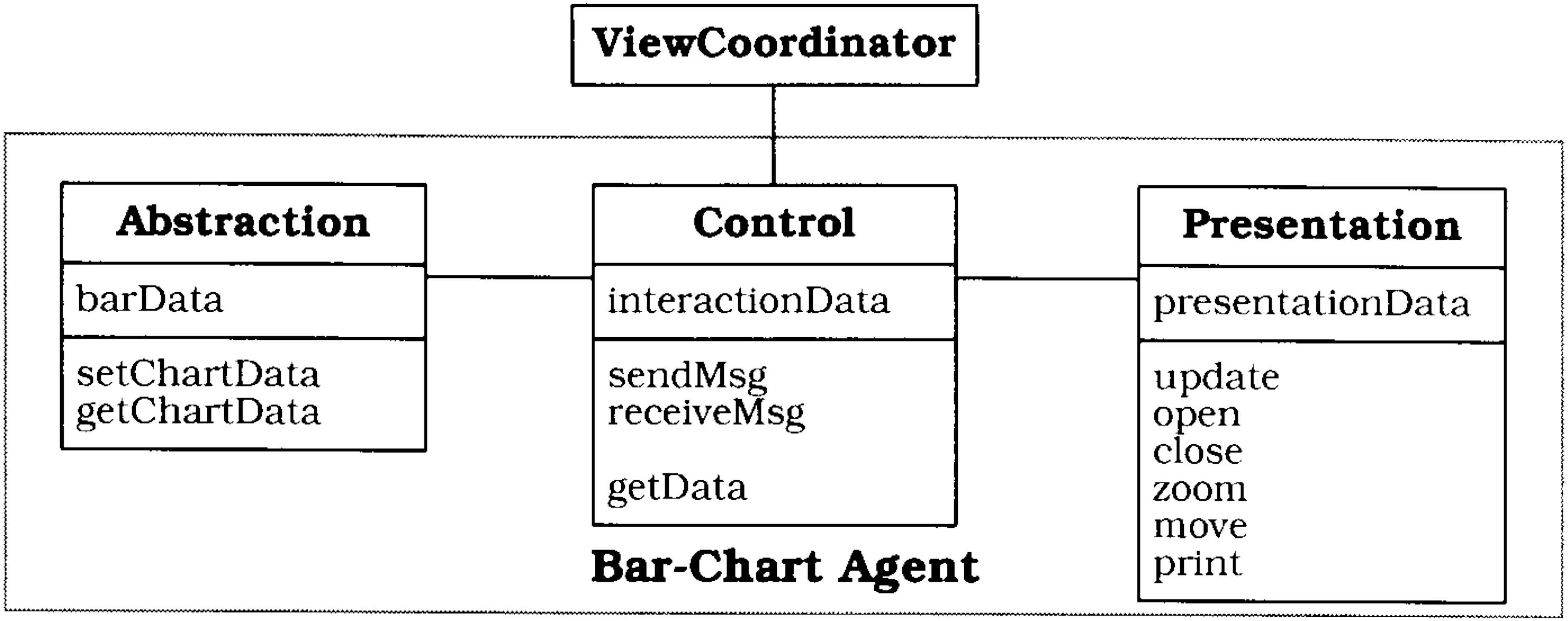
Class Bottom-level Agent	***Collaborators*** • Top-level Agent • Intermediate-level Agent
Responsibility • Provides a specific view of the software or a system service, including its associated human-computer interaction.	

Top Level PAC

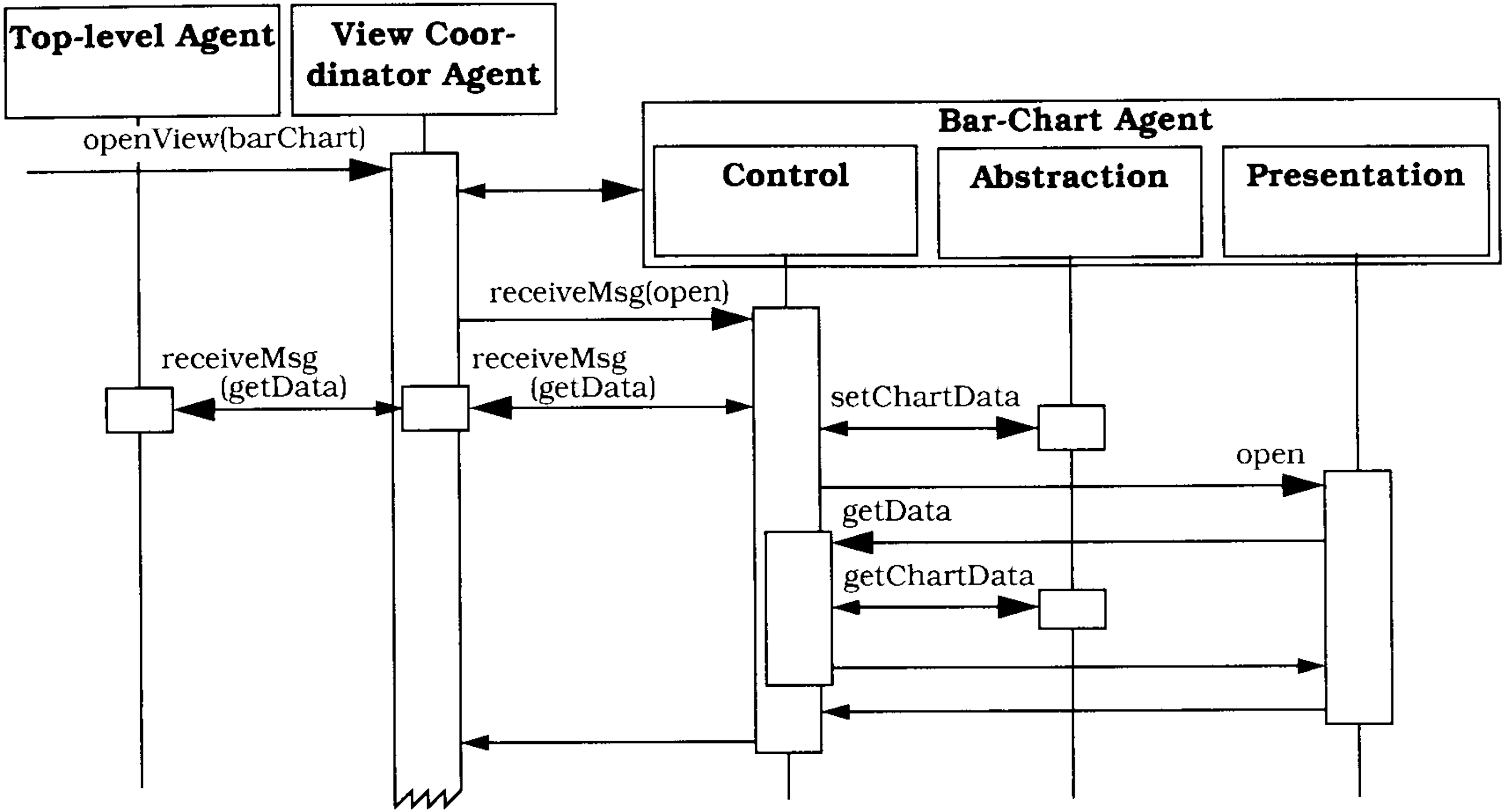
- Abstraction : Global Data model
- Presentation : Some Graphical elements
- Control:
  - Allow sub-agent to access abstraction
  - Manage hierarchy of PAC component
  - Manage info about interaction (log, check applicability of triggered application…


Top-level PAC agent
Repository
myDataRepository
sendMsg
receiveMsg
Bottom-level PAC agent
ErrorHandler
errorData
sendMsg
receiveMsg
displayErrMsg
handleError
ViewCoordinator
viewPACIds
sendMsg
receiveMsg
openView
closeView
Bottom-level PAC agent
Spreadsheet
electionData
sendMsg
receiveMsg
open
close
enterData
Intermediate level PAC agent
BarChart
barData
sendMsg
receiveMsg
open
close
zoom
move
print
PieChart
pieData
sendMsg
receiveMsg
open
close
zoom
move
print
Seats
seatData
sendMsg
receiveMsg
open
close
zoom
move
print
Bottom-level PAC agents

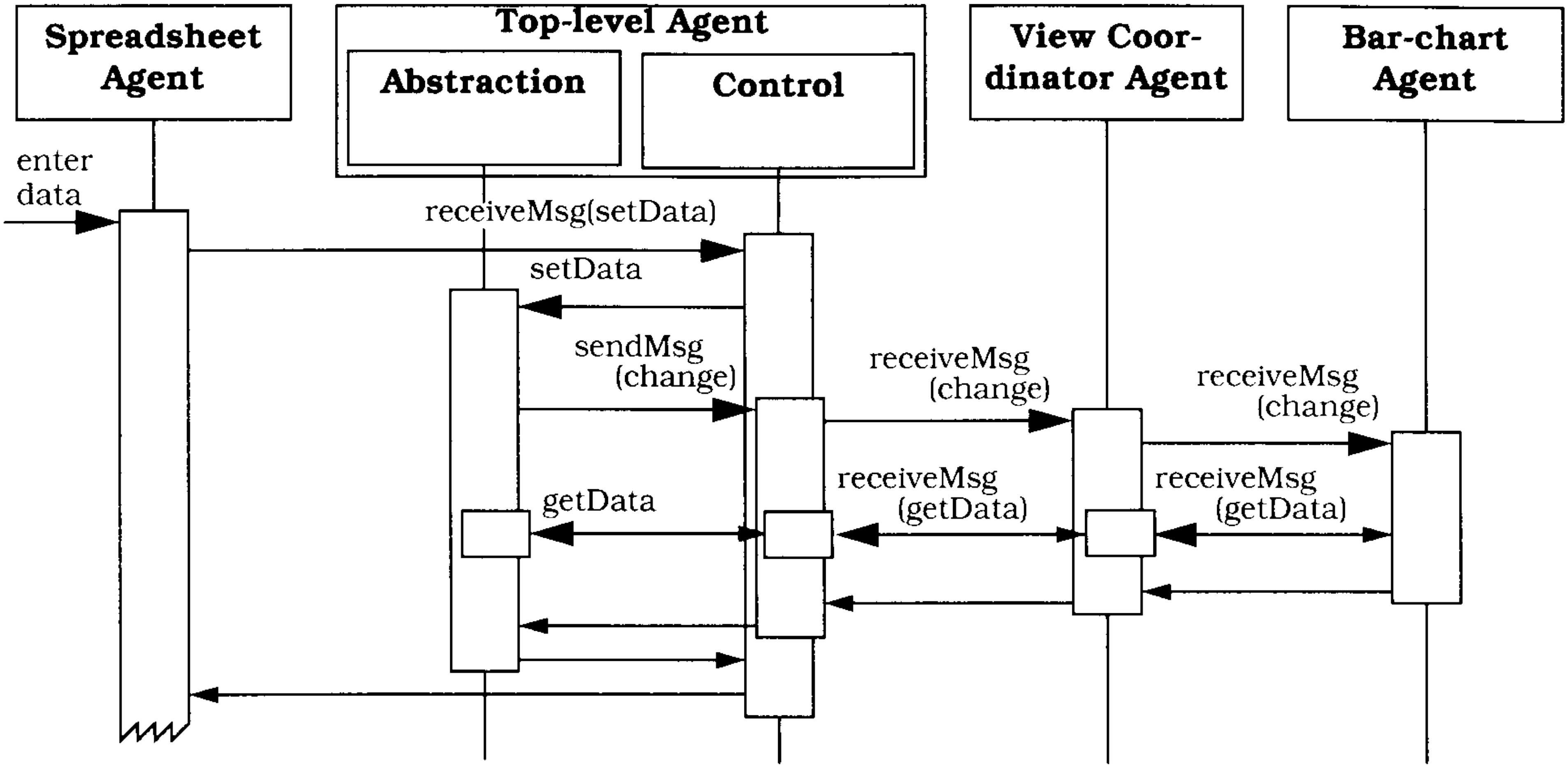
PAC Structure



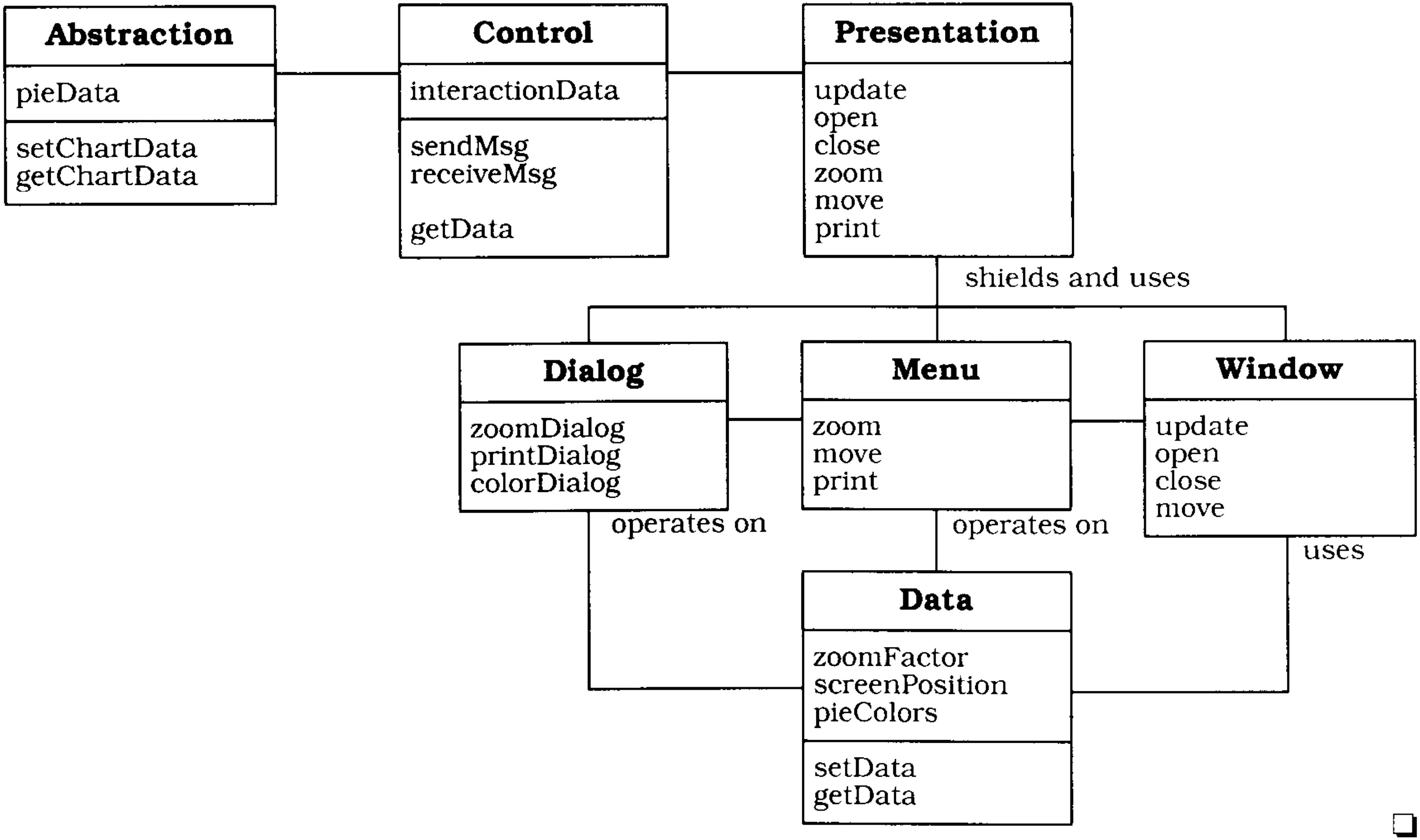
PAC Structure



PAC Structure



PAC Structure



PAC Known Uses

- Network Trafic Management (TS93)
  - Gathering traffic data
  - Threshold checking and generation exceptions
  - Logging and routing of network exception
  - Vizualisation of traffic flow and network exceptions
  - Displaying various user-configurable views of the whole network
  - Statistical evaluation of traffic data
  - Access to historic traffic data
  - System administration and configuration

PAC Benefits

- Separation of concerns: Agent and inside an agent
- Support for change and extension
- Support for multi-tasking: each PAC agent can run its own thread on a different computer…

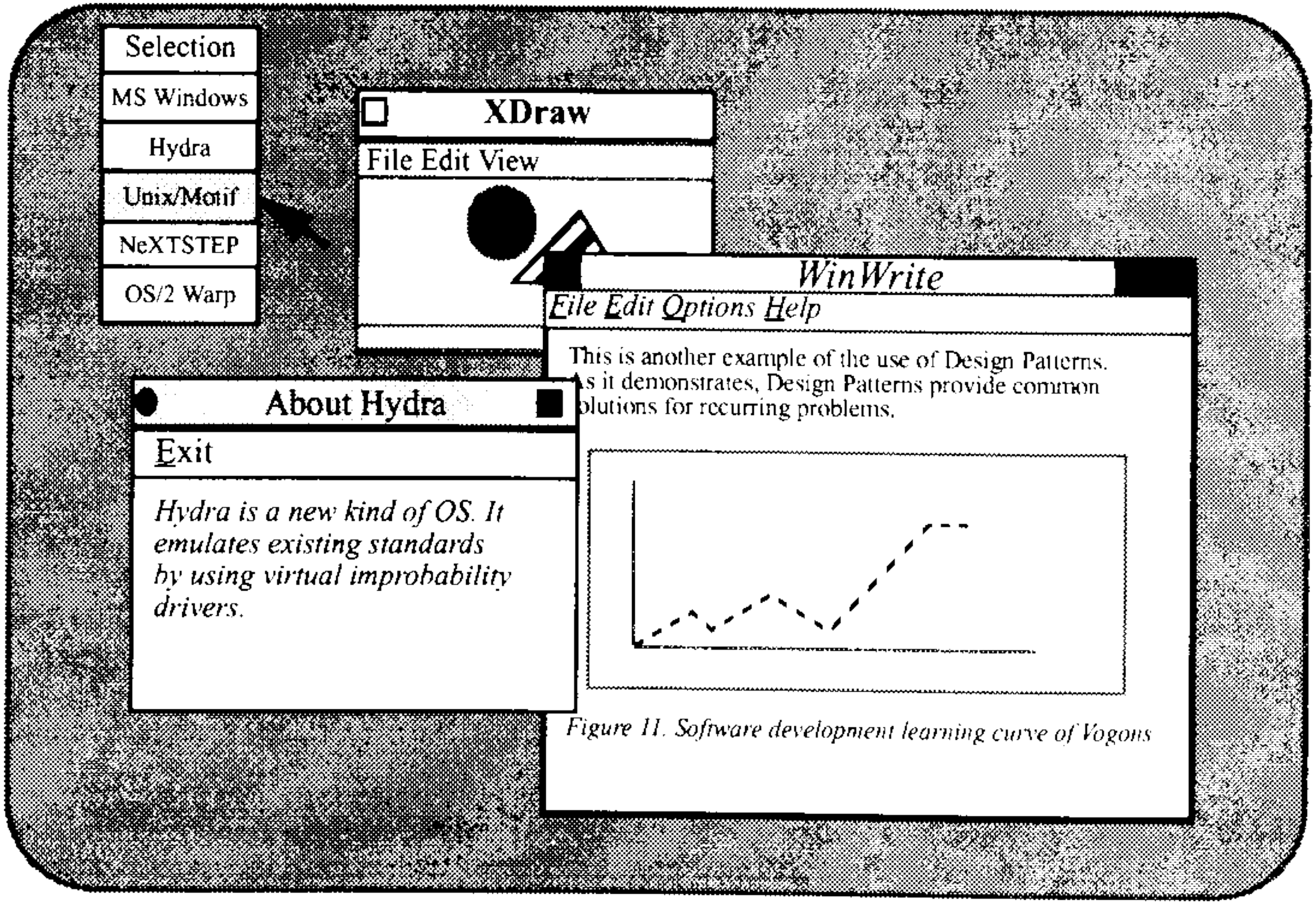
PAC Liabilities

- Increased system complexity: Coordination of agents…
- Complex control component: coordonate action inside agent and with other agents…
- Efficiency : data are propagated throught the tree…
- Applicability : Not a graphic editor where each object is a PAC agent…

Microkernel

- Applies to software systems that be able to adapt to changing system requirements.
- It separates a minimal functional core from extended functionality and customer specific parts.
- The Microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration.

Microkernel

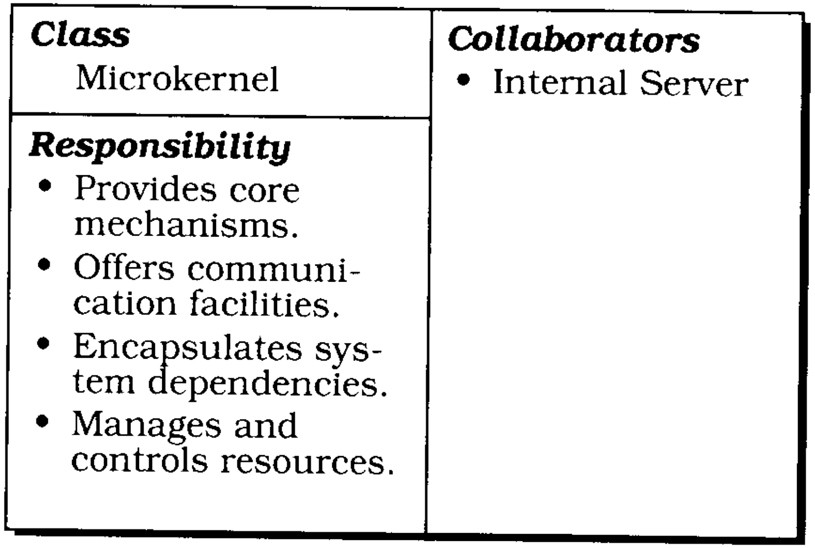


This is another example of the use of Design Patterns. As it demonstrates, Design Patterns provide common solutions for recurring problems.

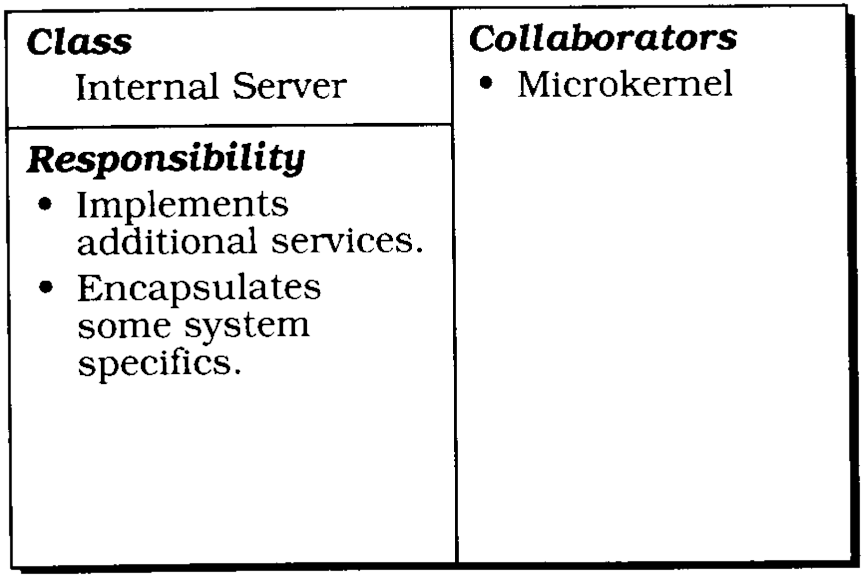
Figure 11. Software development learning curve of Vogons

Hydra is a new kind of OS. It emulates existing standards by using virtual improbability drivers.

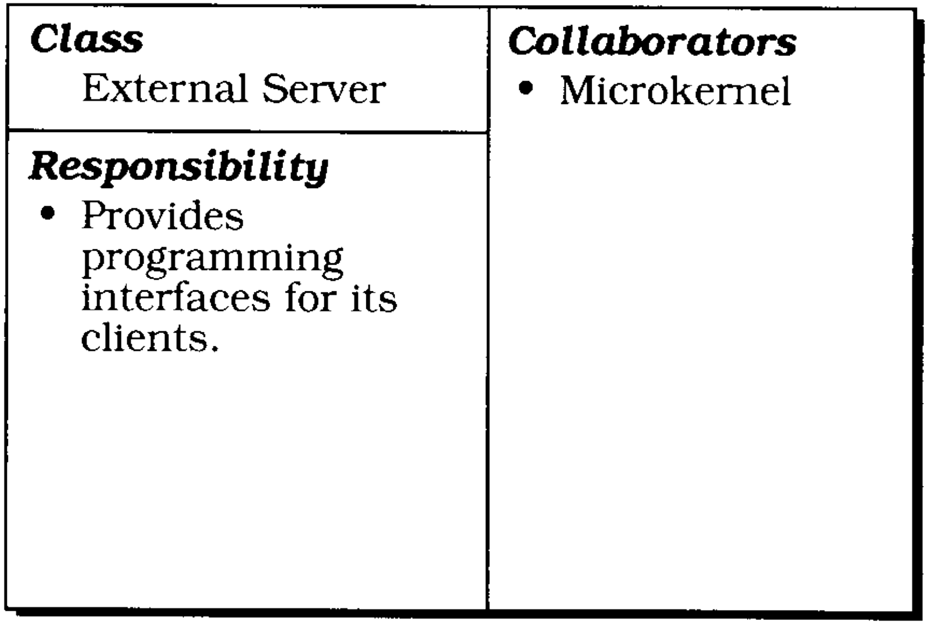
Microkernel Architecture



Class Microkernel	**Collaborators** • Internal Server
Responsibility • Provides core mechanisms. • Offers communication facilities. • Encapsulates system dependencies. • Manages and controls resources.	

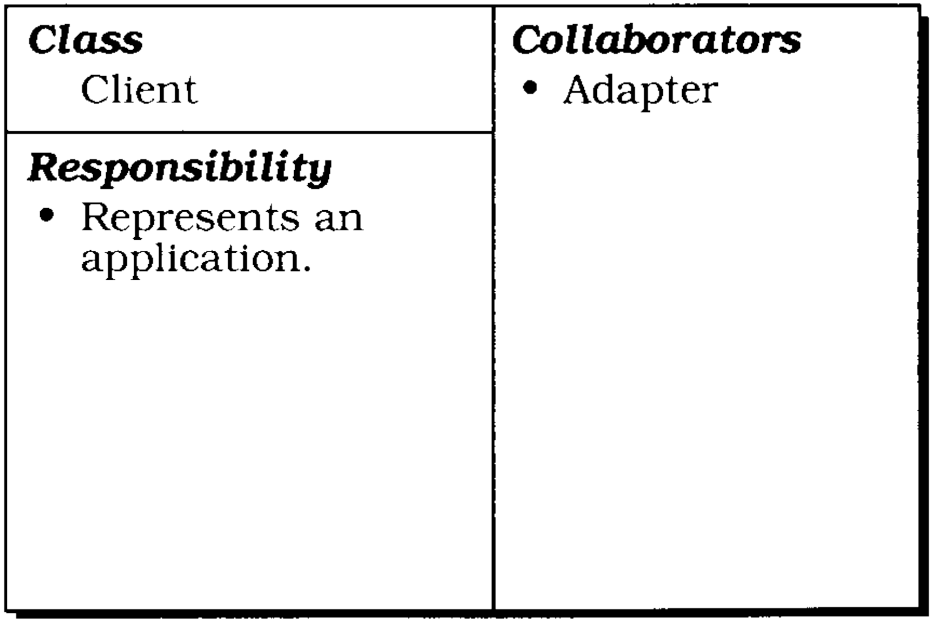


Class Internal Server	**Collaborators** • Microkernel
Responsibility • Implements additional services. • Encapsulates some system specifics.	

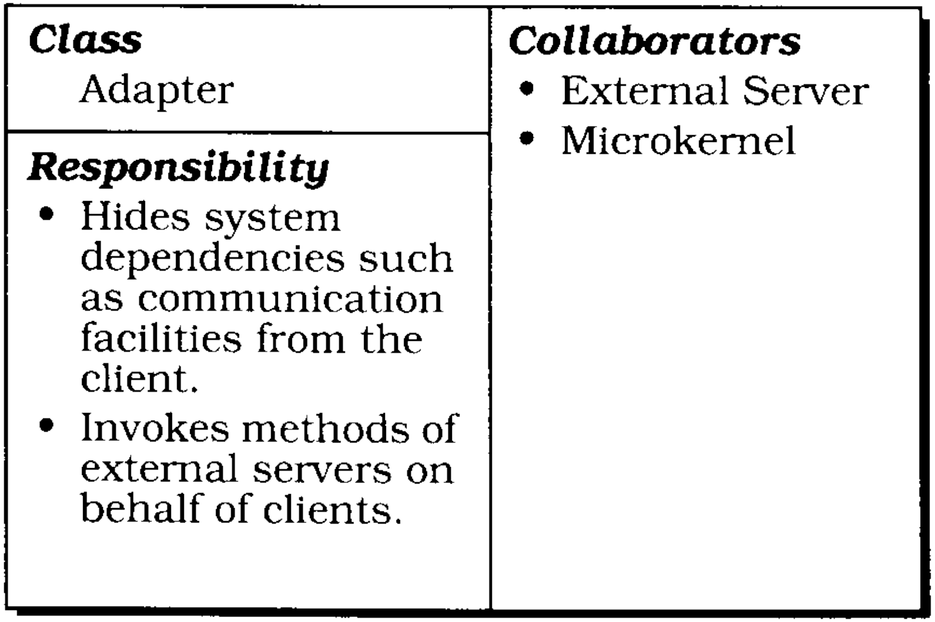


Class External Server	**Collaborators** • Microkernel
Responsibility • Provides programming interfaces for its clients.	

Microkernel Architecture

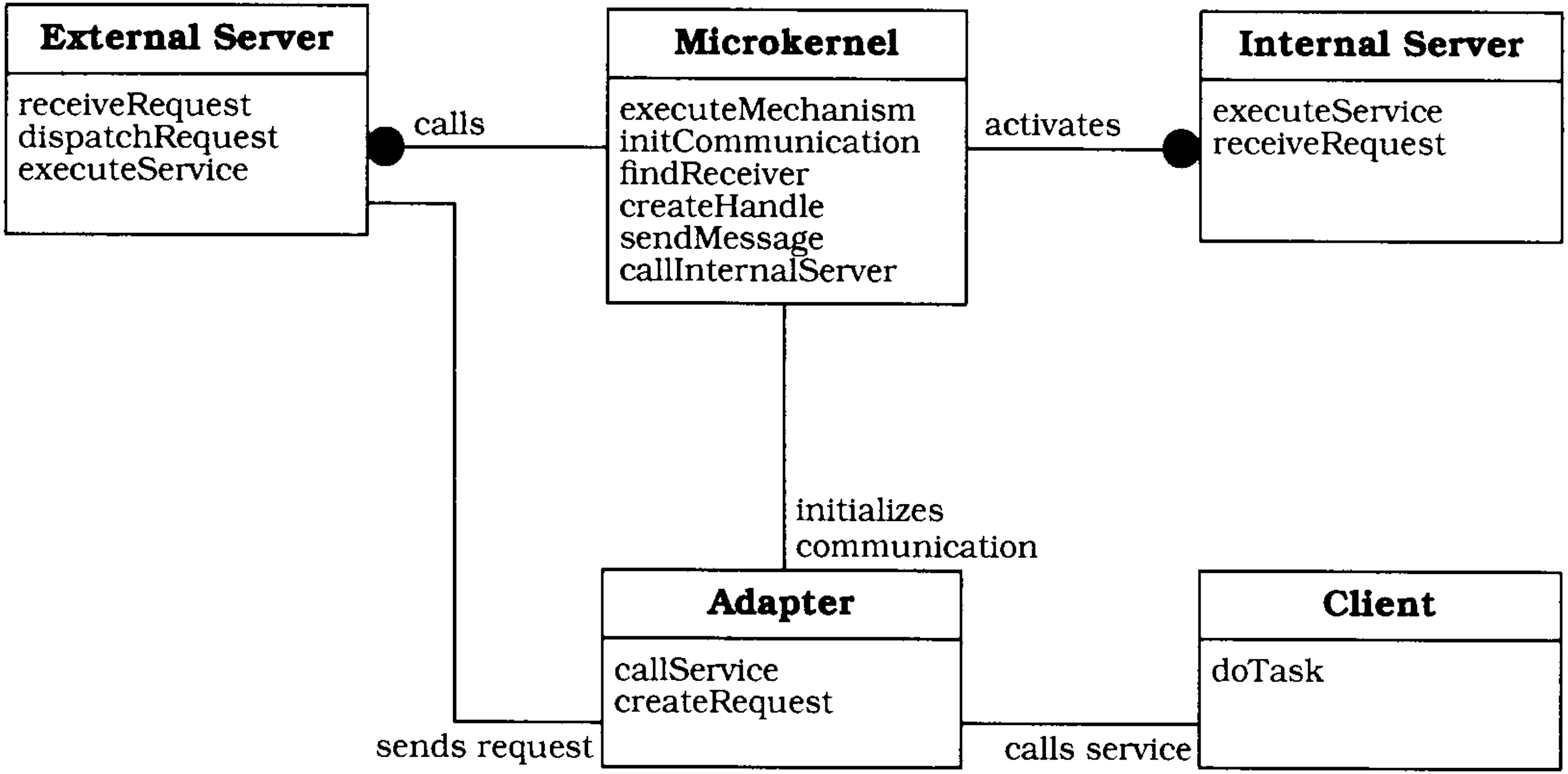


Class	***Collaborators***
Client	• Adapter
Responsibility	
• Represents an application.	

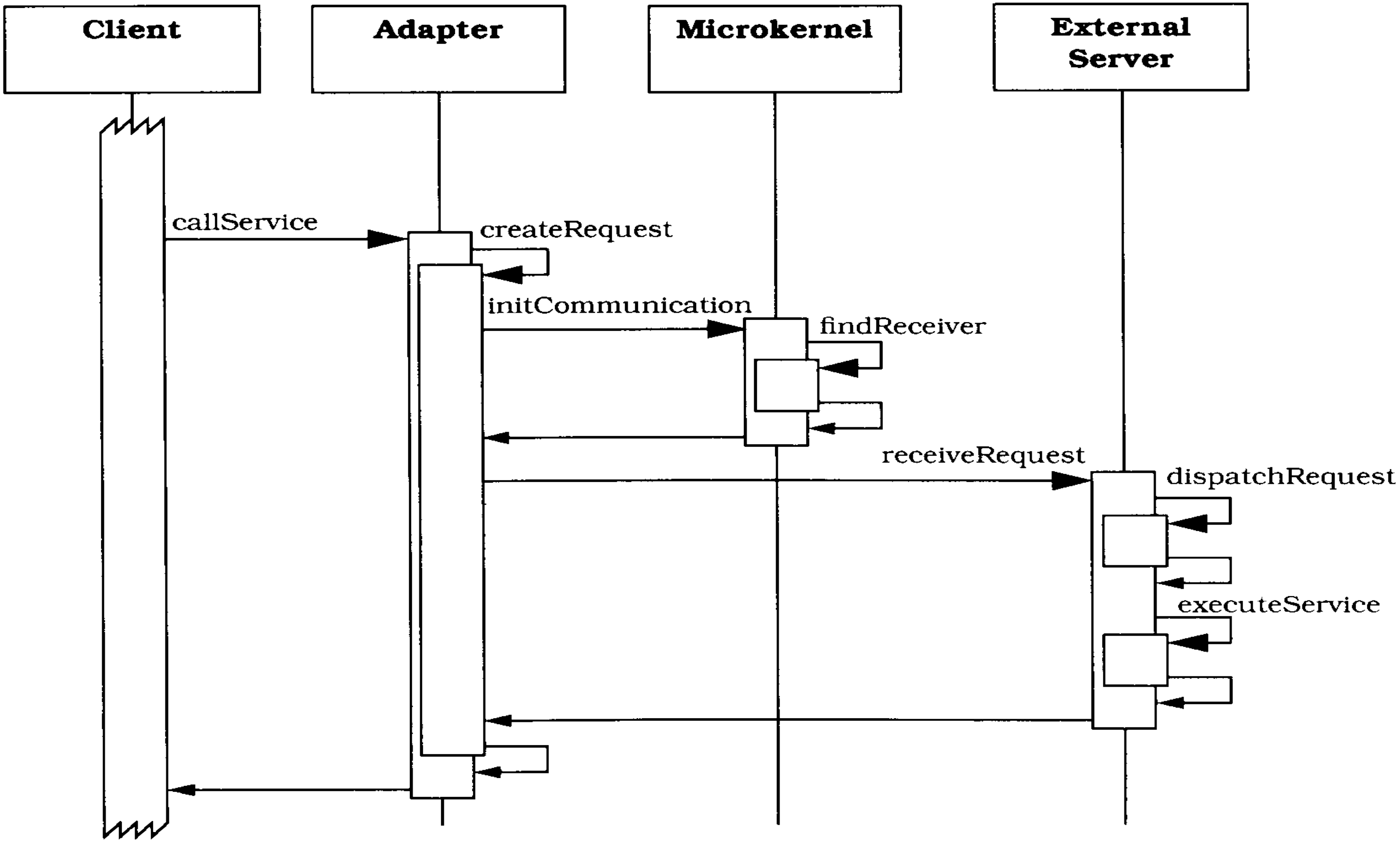


Class	***Collaborators***
Adapter	• External Server • Microkernel
Responsibility	
• Hides system dependencies such as communication facilities from the client. • Invokes methods of external servers on behalf of clients.	

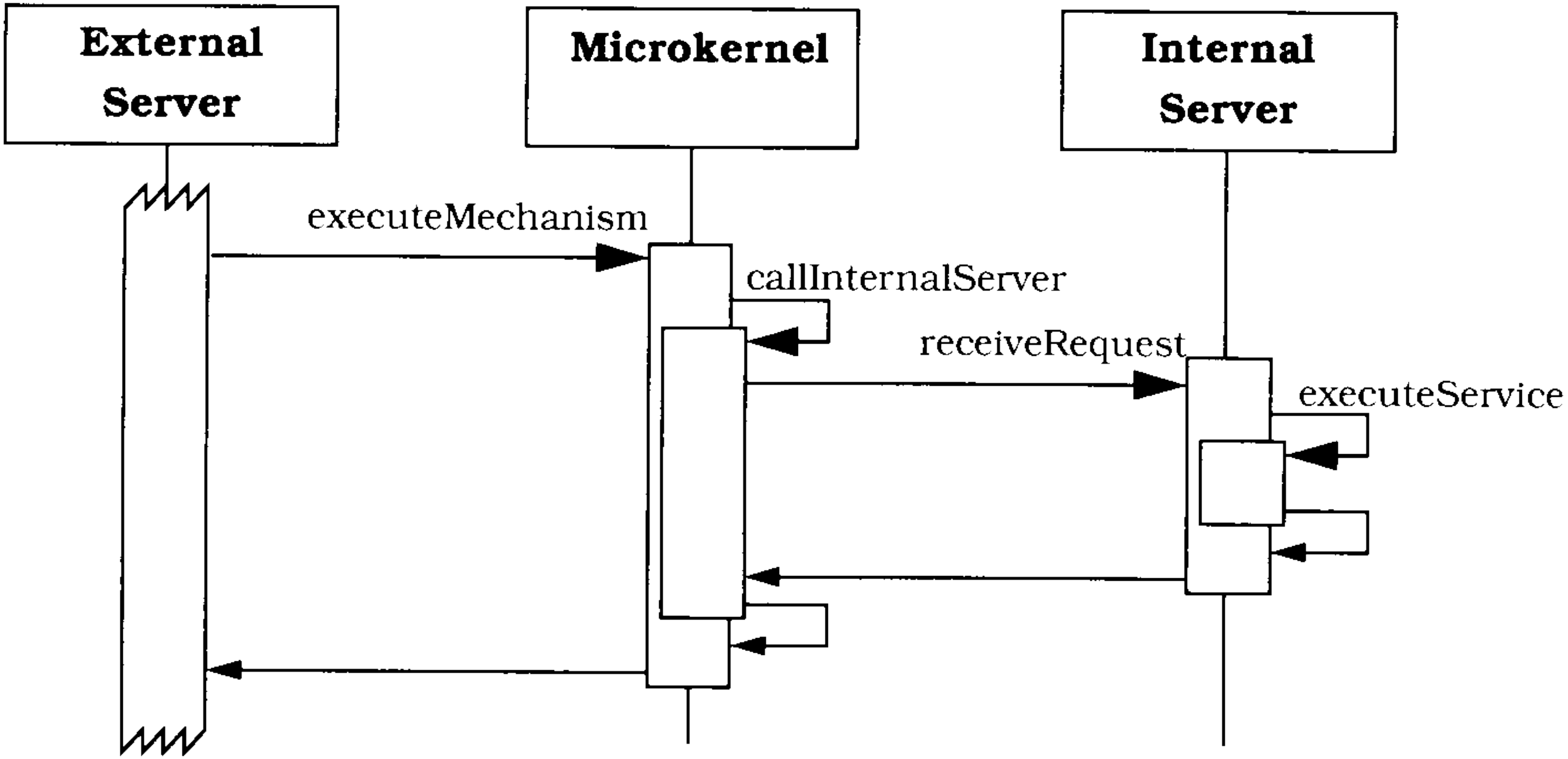
Microkernel Architecture



Microkernel Structure



Microkernel Structure



Microkernel variants

- Microkernel system with indirect Client-Server connections. MK establish channel of communication between client and external servers.

Microkernel known Uses

- Mach (92): Emulate other operating system (NeXTSTEP)
- Amoeba (92):
  - Kernel: process, threads system memory, communication, IO
  - Services not in the kernel are internal servers..

Known uses

- Chorus
- WINDOWS NT:
  - External servers: OS/2.1.X, posix server and win32 server
- MKDE: Microkernel Databank Engine
  - External server : Data model of SQL database

Microkernel Benefits

- Portability : no need to port external servers…
- Flexibility and extensibility
- Separation of policy and mechanism:
  - Mechanism in kernel, policy in external servers
- Scalability
- Reliability: Distributed Microkernel… :-/
- Transparency : Microkernel ~ broker…

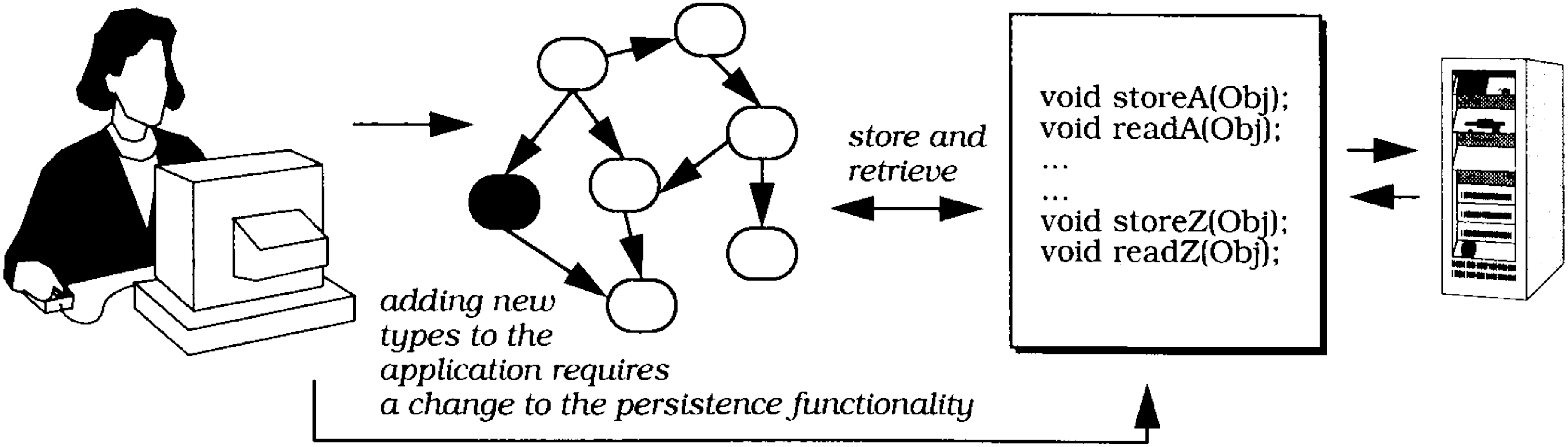
Microkernel Liabilities

- Performance
- Complexity of design and implementation.
  - Basic functionalities of the micro-kernel ??
  - Separation mechanism/policy => deep knowledge of domain.

Reflection

- Provides a mechanism for changing structure and behavior of software dynamically.
- Support modification of fundamental aspects: type structures and function call mechanism
- Meta-level makes the software self-aware
- Base-level includes application logic. Its implementation builds on the meta-level.

Reflection example



```
void storeA(Obj);
void readA(Obj);
...
...
void storeZ(Obj);
void readZ(Obj);
```

Class

Base Level

Responsibility

- Implements the application logic.
- Uses information provided by the meta level.

Collaborators

- Meta Level

Class

Meta Level

Responsibility

- Encapsulates system internals that may change.
- Provides an interface to facilitate modifications to the meta-level.

Collaborators

- Base Level

Class

Metaobject Protocol

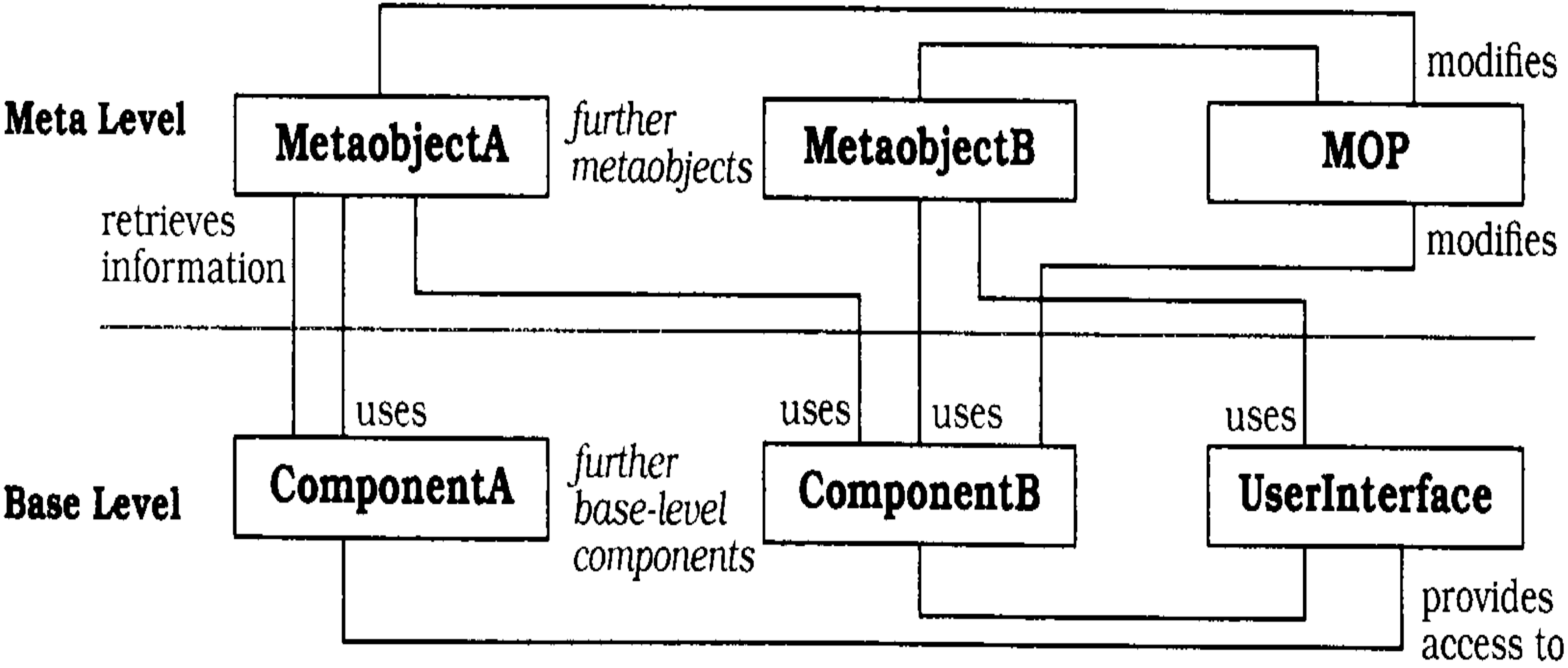
Responsibility

- Offers an interface for specifying changes to the meta level.
- Performs specified changes

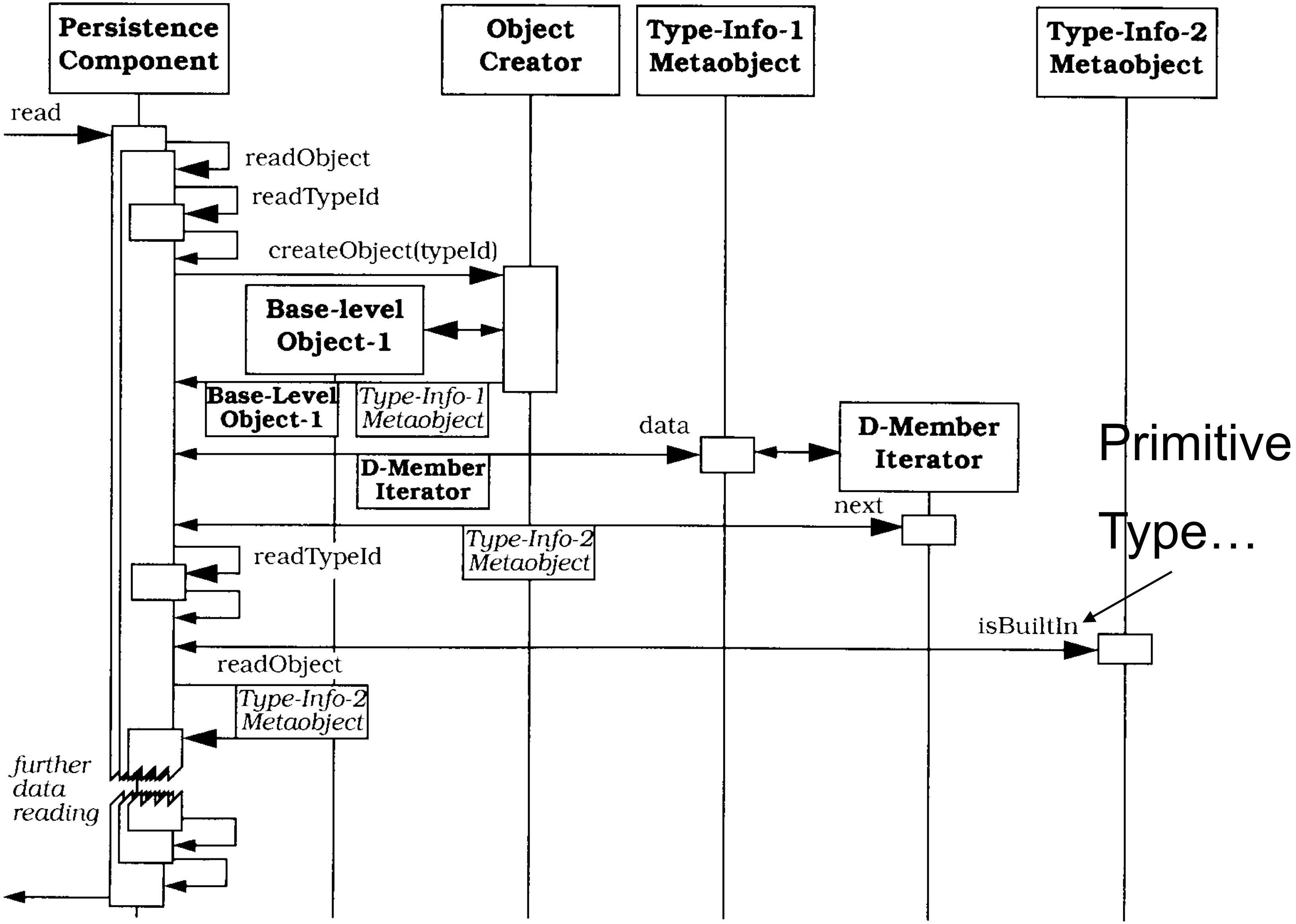
Collaborators

- Meta Level
- Base Level

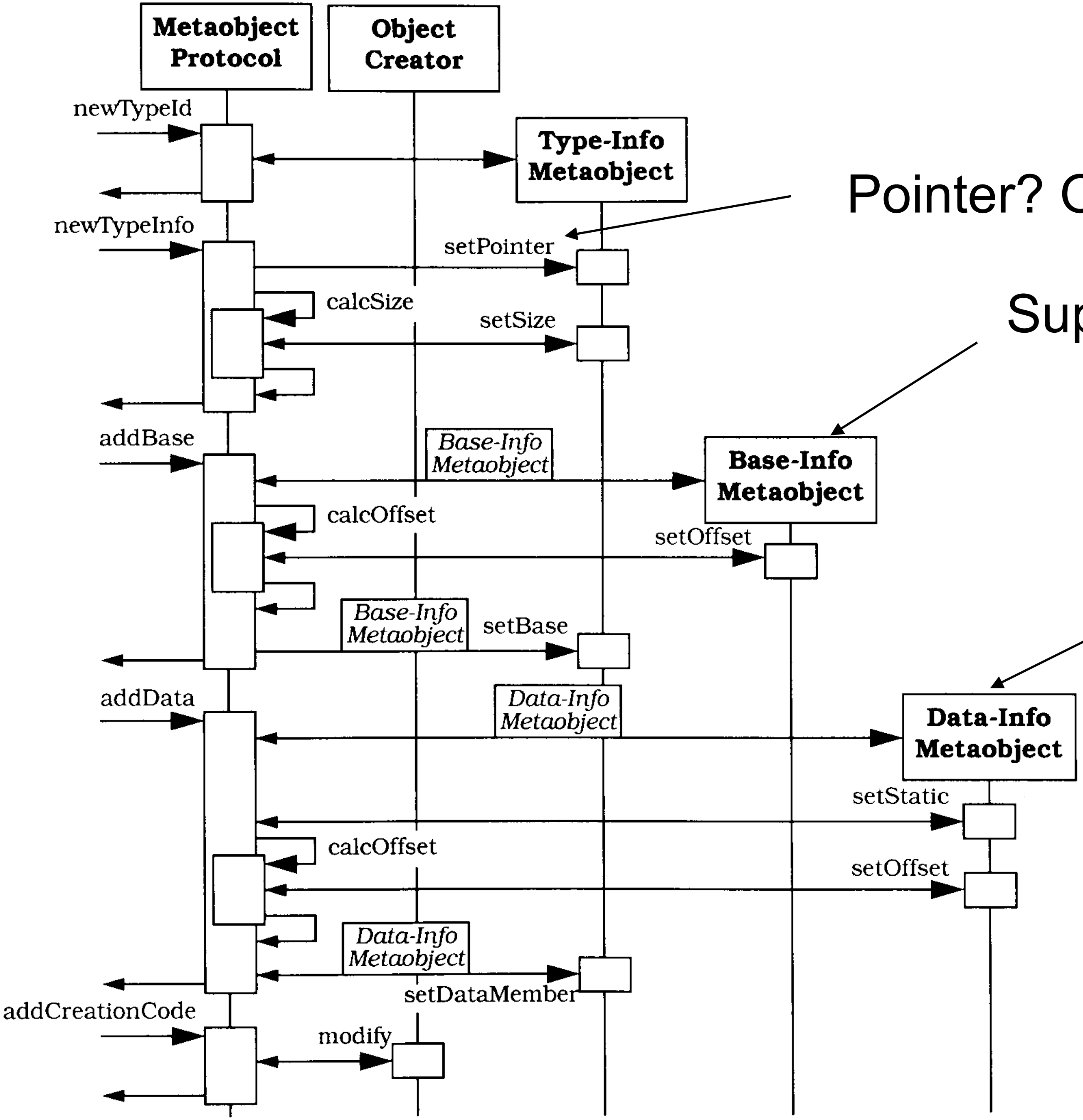
Reflection structure



Reflection example



Primitive Type…



Pointer? Or not

SuperType

Field…

Reflection known Uses

- CLOS : generic function and generic function invocation
- MIP: run-time type information system for C++
- Pgen: persistence component for C++ based on MIP
- Ole2.0, CORBA (dynamic invocation)…

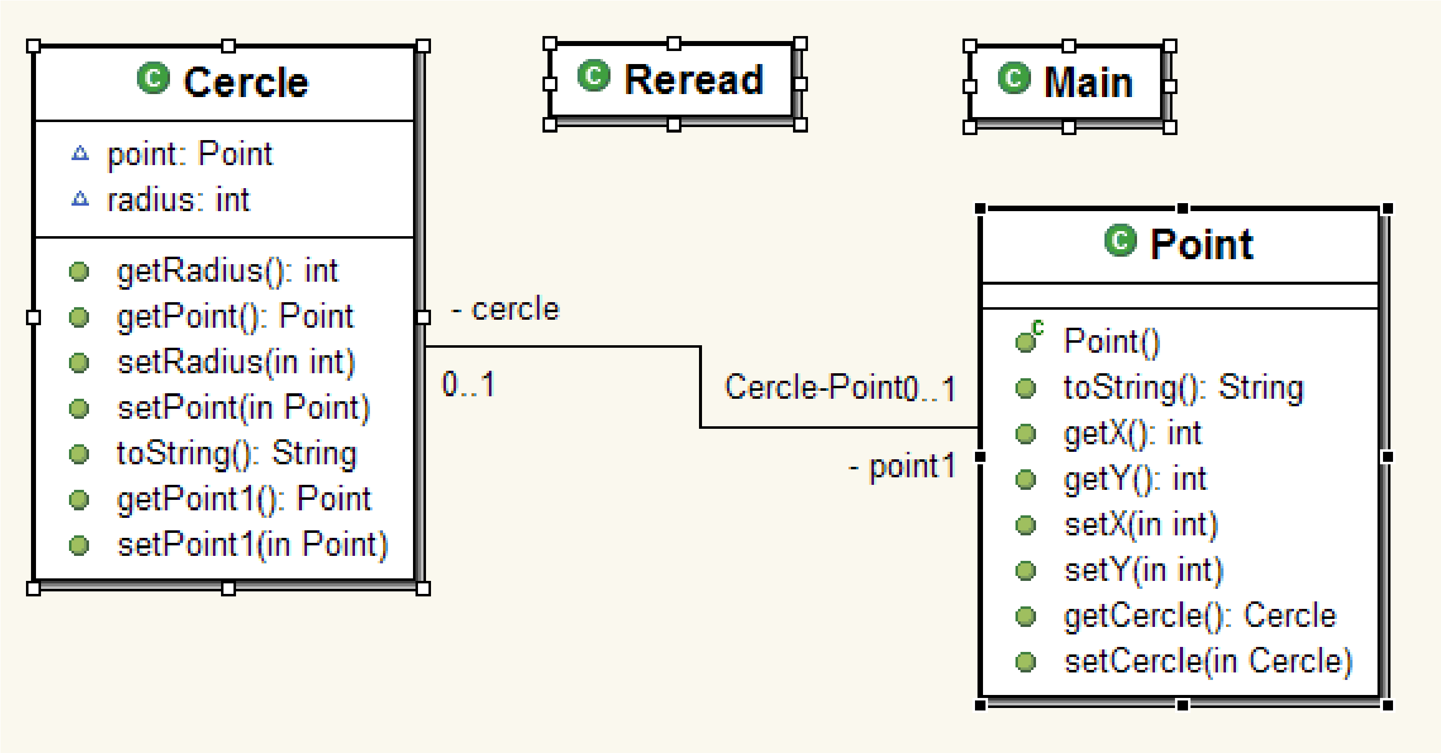
Reflection benefits

- No explicit modification of source code
- Changing a software is easy: no need for visitors, factories and strategies patterns
- Support for many kind of change

Reflection Liabilities

- Modification at the meta-level can cause damage.
- Increased number of component
- Lower efficiency
- Not all potential changes supported (only those supported by the MOP)
- Not all language support reflection

Reflection example



Reflection example

```
public class Main {
public static void main(String args[]) throws Exception {
Point p = new Point();
p.setX(3);
p.setY(4);
Cercle c = new Cercle();
c.setPoint(p);
c.setRadius(6);
XMLEncoder e = new XMLEncoder(new BufferedOutputStream(new
FileOutputStream(args[0])));
e.writeObject(c);
e.close();
System.out.println(c);
}
}
```

Reflection example

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.2_03" class="java.beans.XMLDecoder">
 <object class="Cercle">
  <void property="point">
   <object class="Point">
    <void property="x">
     <int>3</int>
    </void>
    <void property="y">
     <int>4</int>
    </void>
   </object>
  </void>
  <void property="radius">
   <int>6</int>
  </void>
 </object>
</java>
```

Reflection example

```
public class Reread {
public static void main(String args[])
throws Exception {
XMLDecoder d = new XMLDecoder(new
          BufferedInputStream(new
          FileInputStream(args[0])));
Cercle c = (Cercle)d.readObject();
d.close();

System.out.println(c);
}
}
```

Summary (C. Alexander)

- It is possible to make building by stringing together patterns, in a rather loose way. A building made like this, is an assembly of patterns. It is not dense. It is not profound. But it is also possible to put patterns together in such way that many patterns overlap in the same physical space: the building is very dense; it has many meanings captured in a small space; and through this density, it becomes profound.

Drawbacks of Patterns

- Patterns do not lead to direct code reuse.
- Individual Patterns are deceptively simple.
- Composition of different patterns can be very complex.
- Teams may suffer from pattern overload.
- Patterns are validated by experience and discussion rather than by automated testing.
- Integrating patterns into a software development process is a human-intensive activity.