CptS 487
Software Design and Architecture

Lesson 28

Architectural Patterns part 2

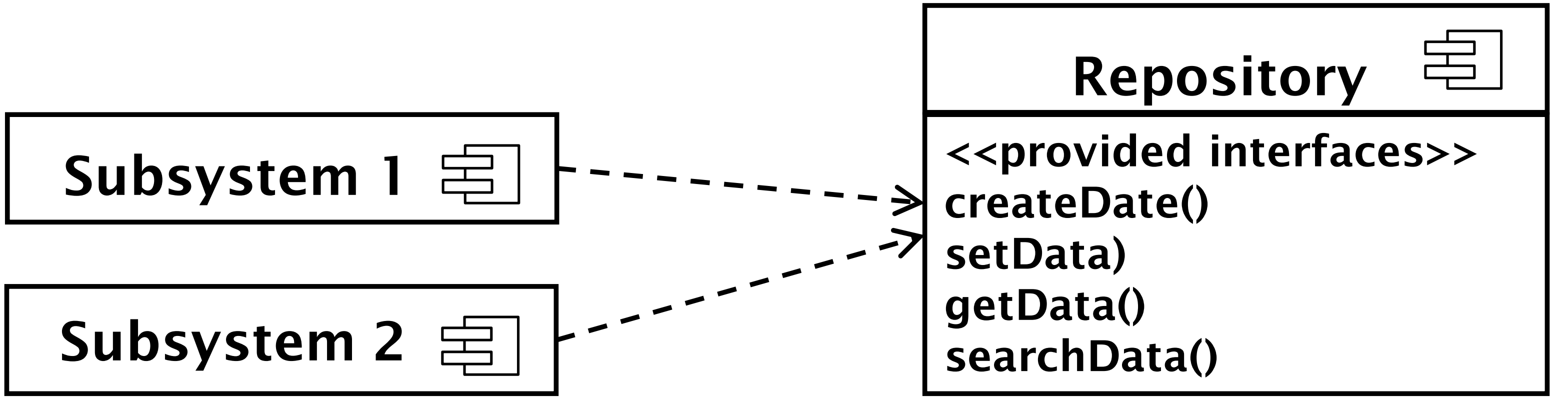
Instructor: Bolong Zeng

Overview

- Other common architectural patterns:
  - Repository
  - Client-Server/Peer-to-peer
  - Broker
  - Pipe-and-filter

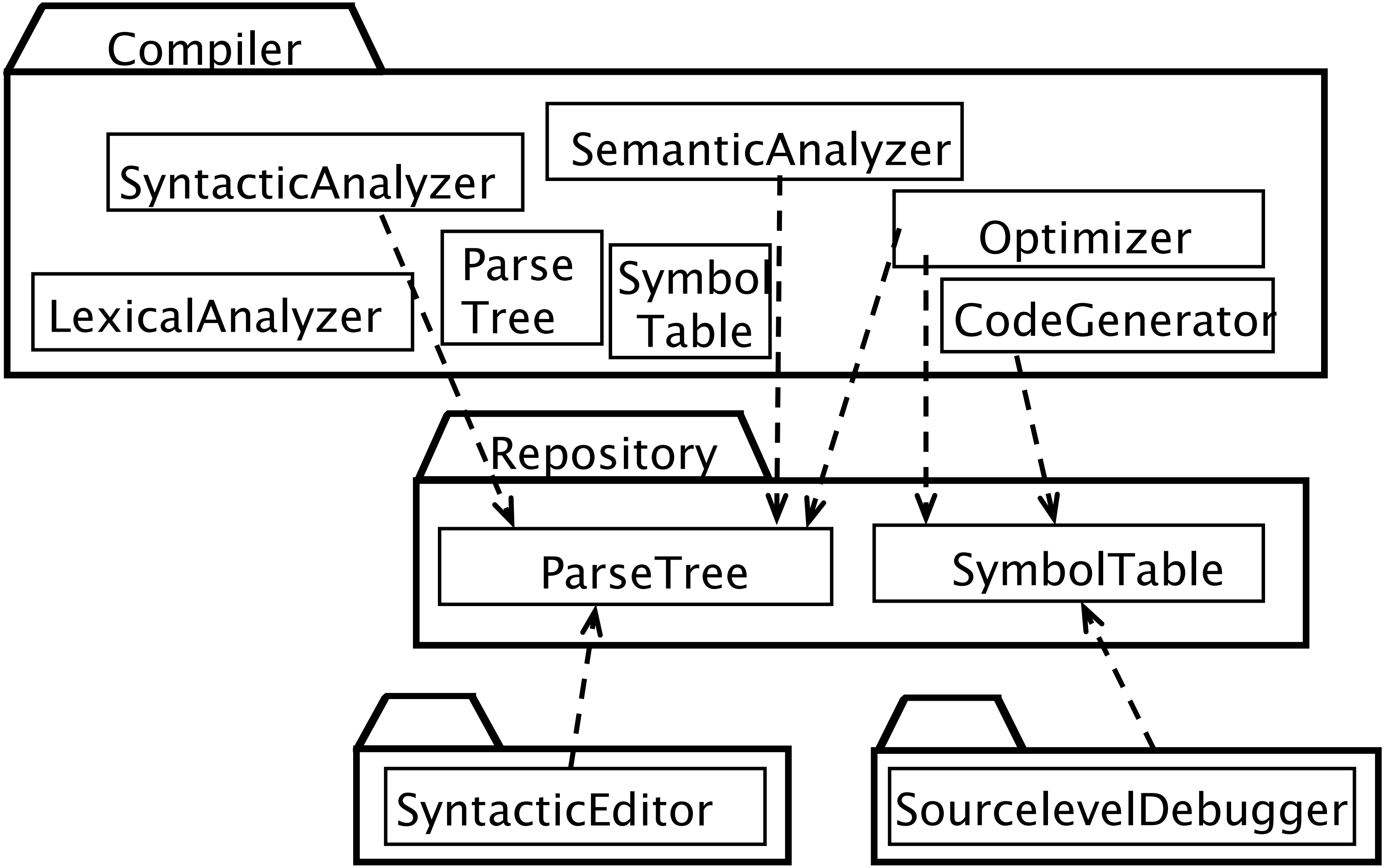
Repository Architectural Pattern

- Subsystems access and modify data from a single data structure called the repository
- Subsystems are loosely coupled (interact only through the repository)
- Control flow is dictated by the repository through triggers or by the subsystems through locks and synchronization primitives



Repository Architectural Pattern (UML Component Diagram)

Repository Architecture Example: Incremental Development Environment (IDE)



Repository Architectural Pattern and Design Principles

1. Divide and conquer: The subsystems and the repository can be designed independently.

2. Increase cohesion: Well-designed subsystems have high cohesion.

3. Reduce coupling: Subsystems interact only through repository. However, the coupling between the repository and the subsystems is high.

6. Increase reuse: The repository may be reused.

7. Design for flexibility: Well suited for constantly changing applications. Once the repository is well-defined it is easy to add new subsystems. However, changes in the repository can impact all subsystems.

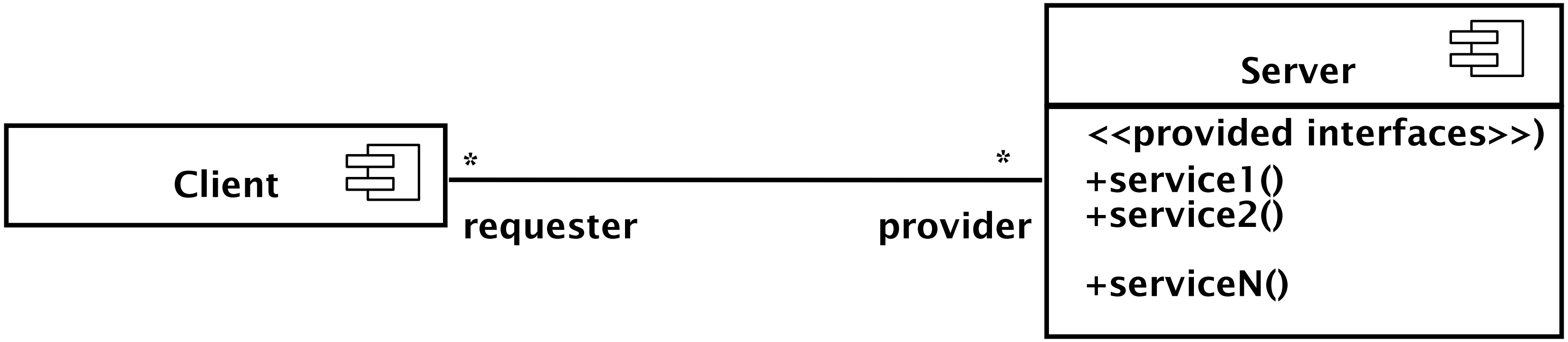
10. Design for testability: You can test repositories and subsystems independently.

The Client-Server Architectural Pattern

- There is at least one subsystem that has the role of server, waiting for and then handling connections.
- There is at least one subsystem that has the role of client, initiating connections in order to obtain some service.
- A further extension is the Peer-to-Peer pattern.
  - A system composed of various software subsystems that are distributed over several hosts (will cover next).

The Client-Server Architectural Pattern

- Each client calls on the server, which performs some service and returns the result
  - The clients know the interface of the server
  - The server does not need to know the interface of the client
- The response in general is immediate
- End users interact only with the client.



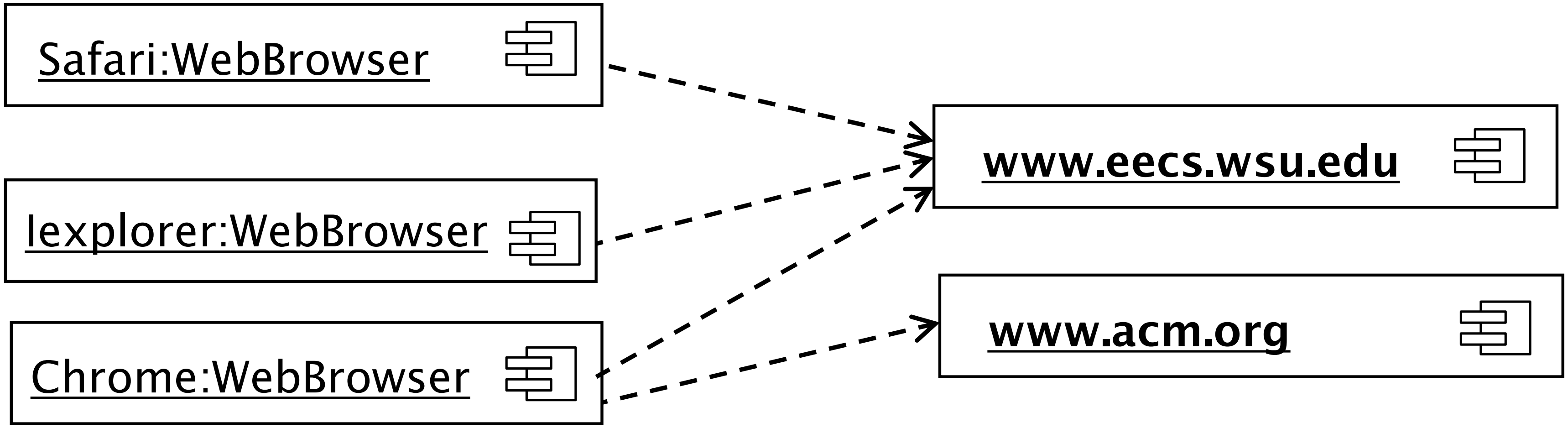
Client/server Architectural Design (UML Component Diagram)

The Client-Server Architectural Pattern

- Often used in the design of database systems
  - Front-end: User application (client)
  - Back end: Database access and manipulation (server)
  - Functions performed by client:
    - Input from the user (Customized user interface)
    - Front-end processing of input data
  - Functions performed by the database server:
    - Centralized data management
    - Data integrity and database consistency
    - Database security

The Client-Server Examples

- Web Server (IIS) – Web Browser (Safari)
- FTP Server (ftpd) – FTPClient (FileZilla)
- Email server (Microsoft Exchange) – Email client (Outlook)
- SQL Server – SQL Server Management Studio



The Web (UML Deployment Diagram)

The Client-Server Architectural Pattern and Design Principles

1. Divide and conquer: Dividing the system into client and server processes is a strong way to divide the system.
   - Each can be separately developed.
2. Increase cohesion: The server provides a cohesive service to clients.
3. Reduce coupling: There is usually only one communication channel exchanging simple messages.
4. Increase abstraction: Separate distributed components are often good abstractions.
6. Increase reuse: It might be possible to find suitable frameworks on which to build client-server systems
   - However, client-server systems are often very application specific.

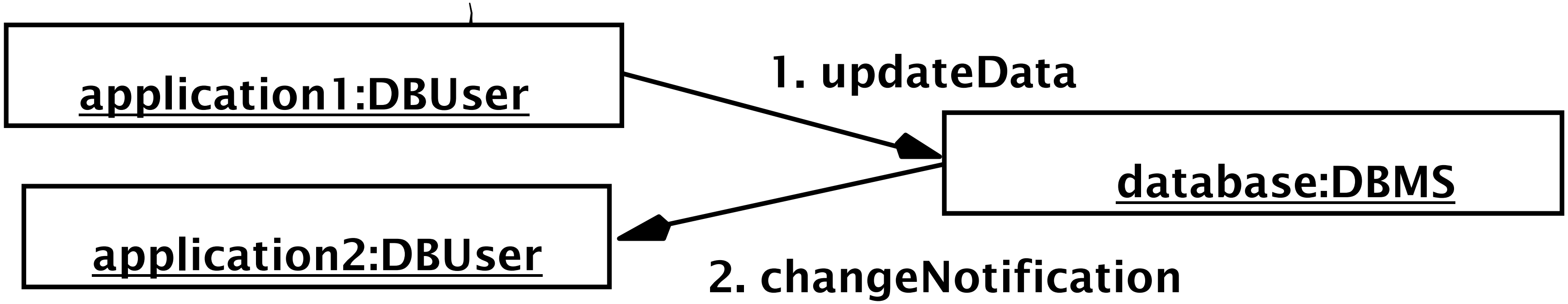
The Client-Server Architectural Pattern and Design Principles

7. Increase flexibility: User interface of client supports a variety of end devices (PDA, Handy, laptop, wearable computer)
9. Design for portability: Server runs on many operating systems and many networking environments
10. Design for testability: You can test clients and servers independently.

- Location-Transparency: Server might itself be distributed, but provides a single "logical" service to the user
- High Performance: Client optimized for interactive display-intensive tasks; Server optimized for CPU-intensive operations

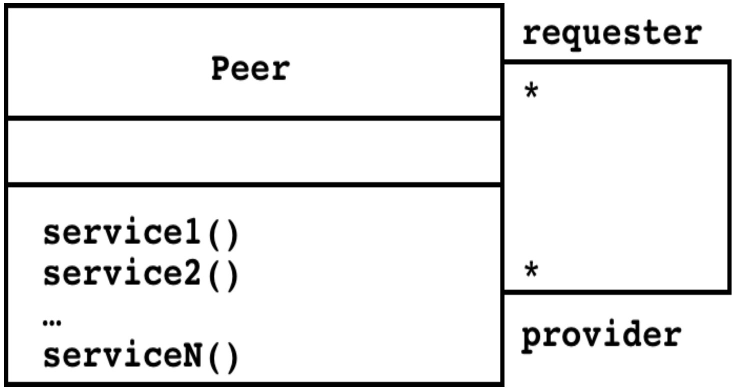
Limitation of Client/Server Architectures

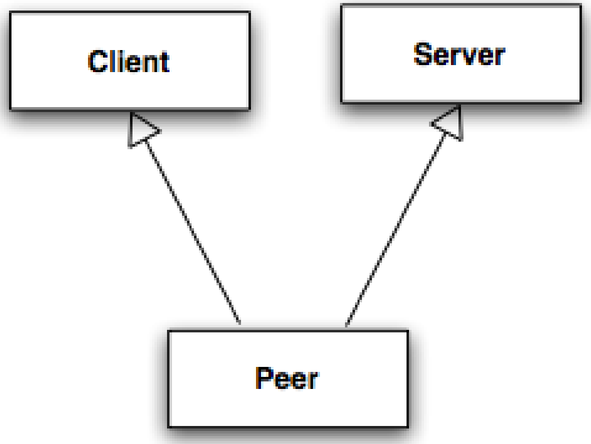
- Client/Server systems do not provide peer-to-peer communication
- Peer-to-peer communication is often needed
- Example:
  - Database must process queries from application and should be able to send notifications to the application when data have changed



Peer-to-Peer Architectural Pattern

- Generalization of Client/Server Architectural Pattern
  - "Clients can be servers and servers can be clients"
- Introduction a new abstraction: Peer
  - "Clients and servers can be both peers"

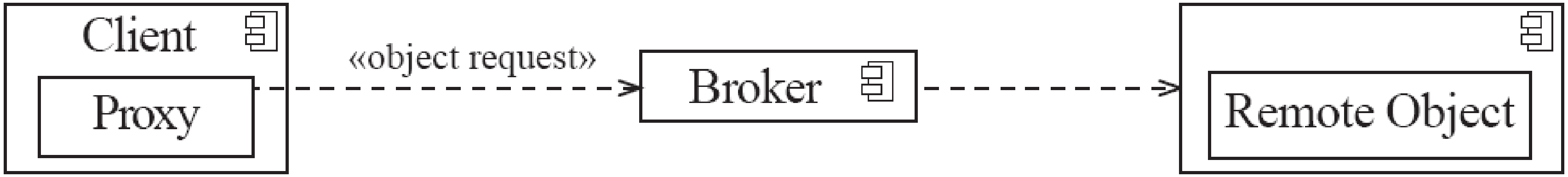




"A peer can be a client as well as a server".

The Broker Architectural Pattern

- Transparently distribute aspects of the software system to different nodes
  - An object can call methods of another object without knowing that this object is remotely located.
  - CORBA is a well-known open standard that allows you to build this kind of architecture.
  - Example:



The Broker Architectural Pattern (UML Deployment Diagram)

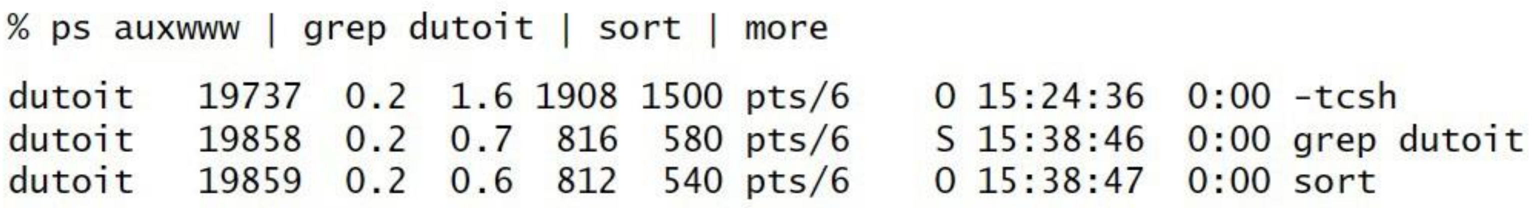
The Broker architecture and design principles

1. Divide and conquer: The remote objects can be independently designed.
5. Increase reusability: It is often possible to design the remote objects so that other systems can use them too.
6. Increase reuse: You may be able to reuse remote objects that others have created.
7. Design for flexibility: The brokers can be updated as required, or the proxy can communicate with a different remote object.
9. Design for portability: You can write clients for new platforms while still accessing brokers and remote objects on other platforms.

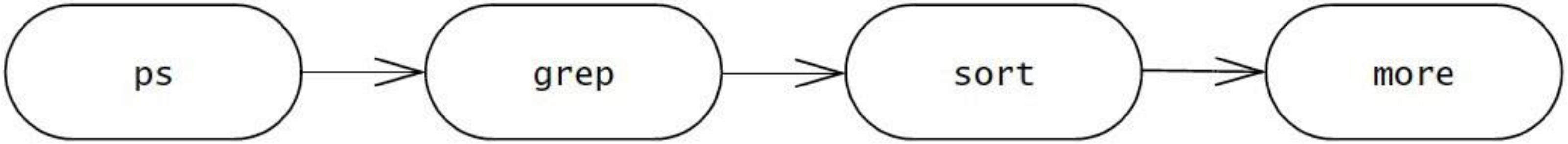
The Pipe-and-Filter Architectural Pattern

- A pipeline consists of a chain of processing elements (processes, threads, etc.), arranged so that the output of one element is the input to the next element
  - A stream of data, in a relatively simple format, is passed through these series of processes
  - Each of which transforms it in some way.
  - Data is constantly fed into the pipeline.
  - The processes work concurrently.
  - Example: Unix

    Unix shell command:



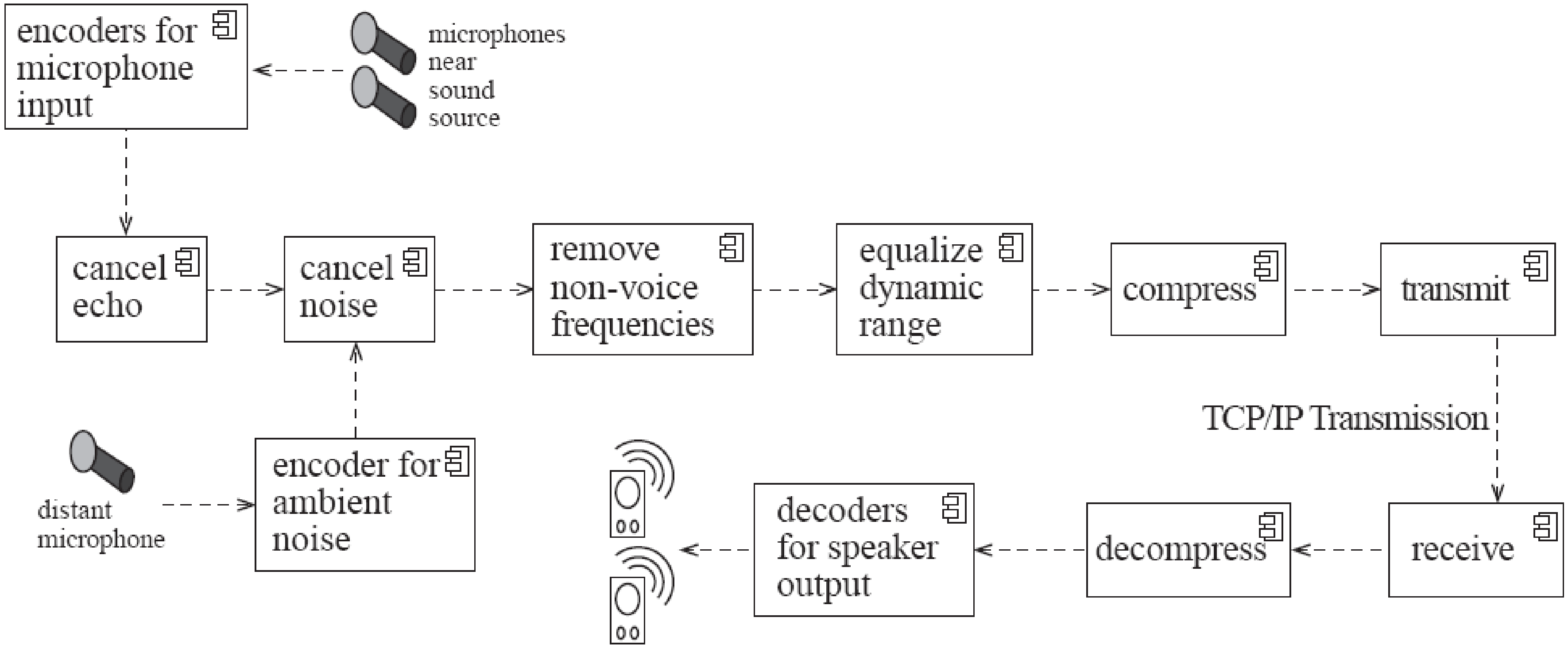
```
% ps auxwww | grep dutoit | sort | more

dutoit   19737  0.2  1.6 1908 1500 pts/6    O 15:24:36  0:00 -tcsh
dutoit   19858  0.2  0.7  816  580 pts/6    S 15:38:46  0:00 grep dutoit
dutoit   19859  0.2  0.6  812  540 pts/6    O 15:38:47  0:00 sort
```



The Pipe-and-Filter Architectural Pattern

- It consists of two subsystems called pipes and filters
  - **Filter:** A subsystem that does a processing step
  - **Pipe:** A Pipe is a connection between two processing steps
- Each filter has an input pipe and an output pipe.
  - The data from the input pipe are processed by the filter and then moved to the output pipe
- The architecture is very flexible.
  - Almost all the components could be removed.
  - Components could be replaced.
  - New components could be inserted.
  - Certain components could be reordered.

Example of a Pipe-and-Filter System


encoders for microphone input
microphones near sound source
cancel echo
cancel noise
remove non-voice frequencies
equalize dynamic range
compress
transmit
TCP/IP Transmission
distant microphone
encoder for ambient noise
decoders for speaker output
decompress
receive

The pipe-and-filter architecture and design principles

1. Divide and conquer: The separate processes can be independently designed.
2. Increase cohesion: The processes have functional cohesion.
3. Reduce coupling: The processes have only one input and one output.
4. Increase abstraction: The pipeline components are often good abstractions, hiding their internal details.
5. Increase reusability: The processes can often be used in many different contexts.
6. Increase reuse: It is often possible to find reusable components to insert into a pipeline.
7. Design for flexibility: There are several ways in which system is flexible.

10. Design for testability: It is normally easy to test the individual processes.

Summary of Architecture vs Design Principles

	1	2	3	4	5	6	7	8	9	10
Multi-layers	■	■	■	■	■	■	■	■	■	■
Repository	■	■	■			■	■			■
Client-server	■	■	■	■		■	■		■	■
Broker	■				■	■	■		■	
Pipe-and-filter	■	■	■	■	■	■	■			■
MVC	■	■	■			■	■			■