The Client-Server Model – part 1

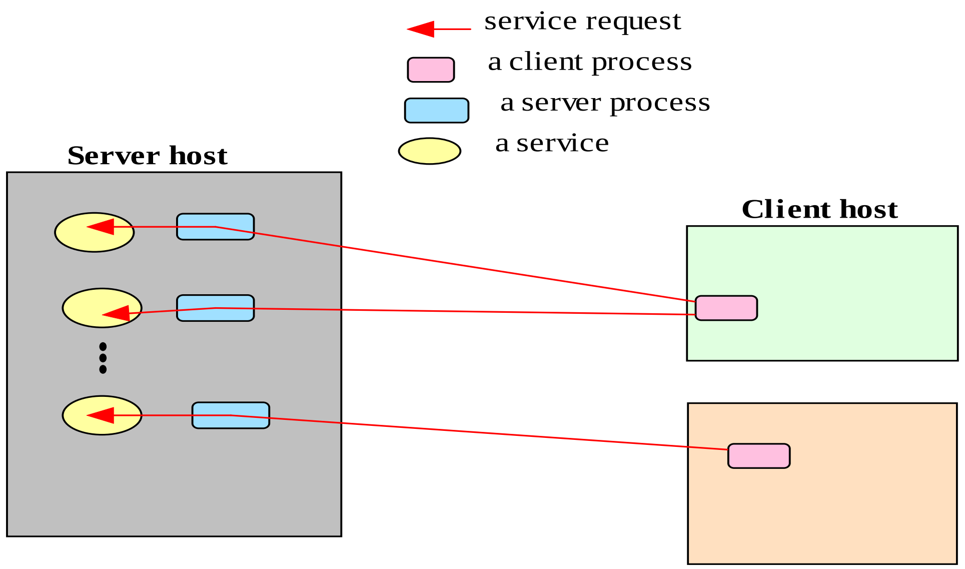
M. L. Liu

Introduction

- The Client-Server paradigm is the most prevalent model for distributed computing protocols.
- It is the basis of all distributed computing paradigms at a higher level of abstraction.
- It is **service-**oriented, and employs a **request-response protocol**.

The Client-Server Paradigm

- A server process, running on a server host, provides access to a service.
- A client process, running on a client host, accesses the service via the server process.
- The interaction of the process proceeds according to a protocol.

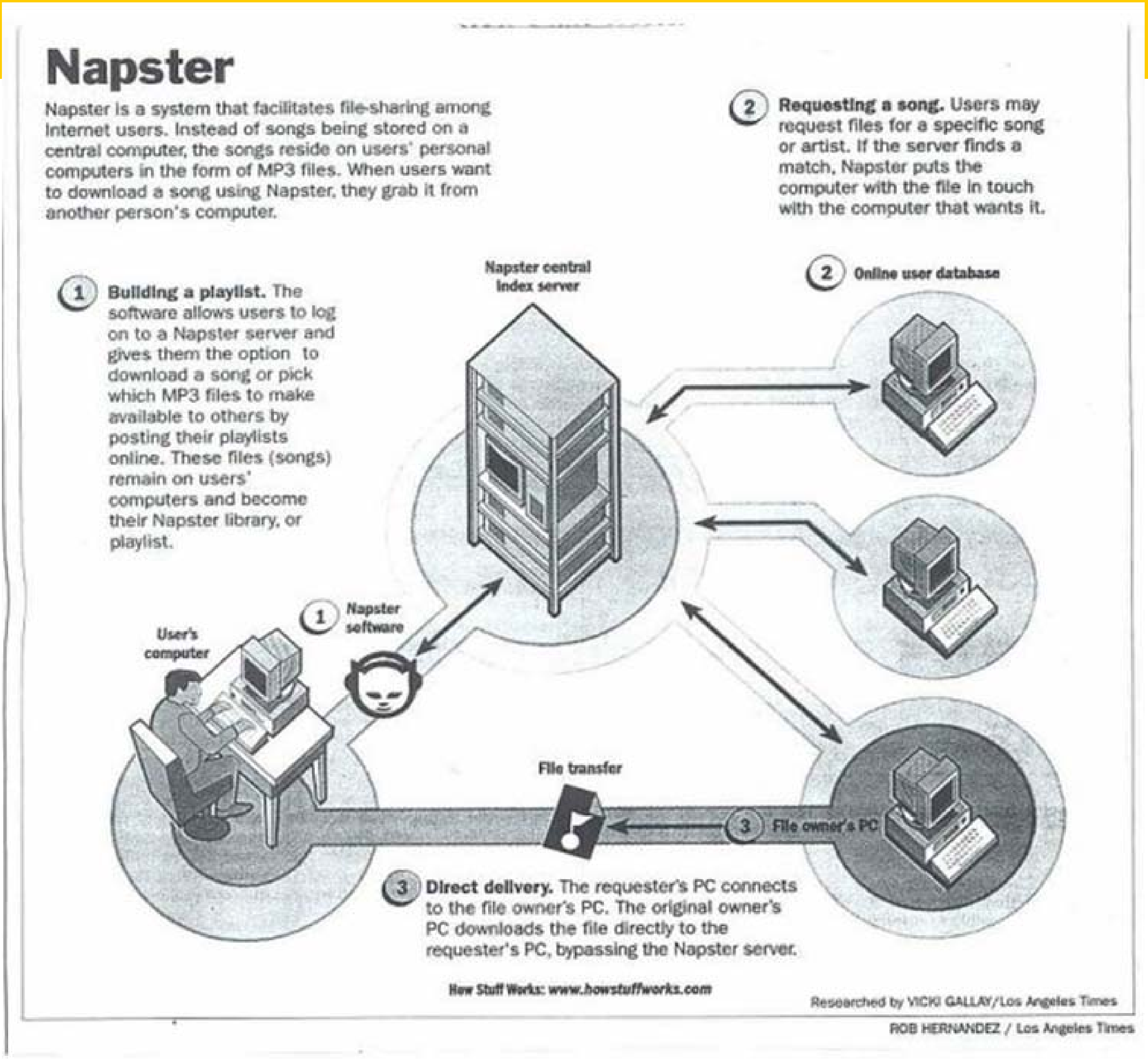


The Client-Server Paradigm, conceptual

Client-server applications and services

- An application based on the client-server paradigm is a client-server application.
- On the Internet, many services are Client-server applications. These services are often known by the protocol that the application implements.
- Well known Internet services include HTTP, FTP, DNS, finger, gopher, etc.
- User applications may also be built using the client-server paradigm.

A Sample Client-Server Application



Napster

Napster is a system that facilitates file-sharing among Internet users. Instead of songs being stored on a central computer, the songs reside on users' personal computers in the form of MP3 files. When users want to download a song using Napster, they grab it from another person's computer.

(1) **Building a playlist.** The software allows users to log on to a Napster server and gives them the option to download a song or pick which MP3 files to make available to others by posting their playlists online. These files (songs) remain on users' computers and become their Napster library, or playlist.

(2) **Requesting a song.** Users may request files for a specific song or artist. If the server finds a match, Napster puts the computer with the file in touch with the computer that wants it.

(3) **Direct delivery.** The requester's PC connects to the file owner's PC. The original owner's PC downloads the file directly to the requester's PC, bypassing the Napster server.

How Stuff Works: www.howstuffworks.com

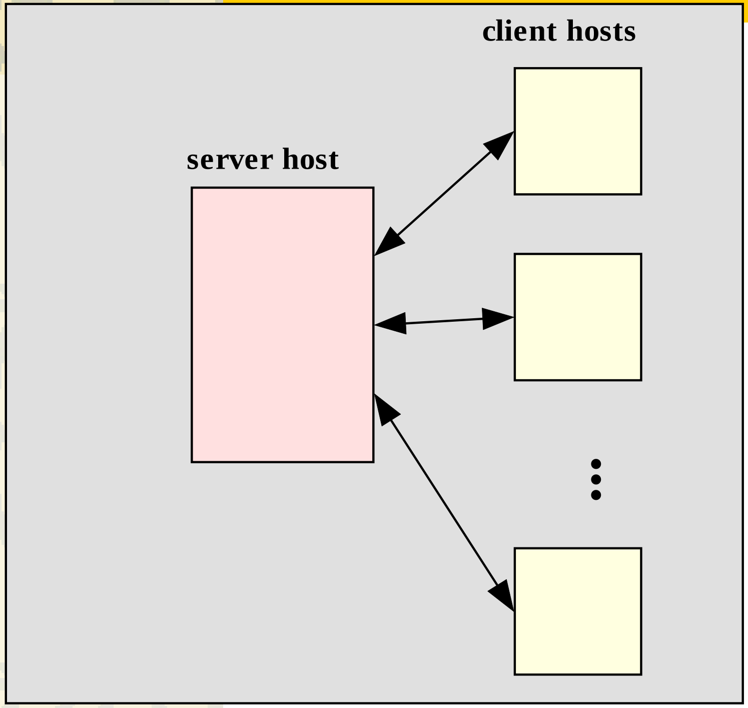
Researched by VICKI GALLAY/Los Angeles Times

ROB HERNANDEZ / Los Angeles Times

Client-server system architecture
vs.
Client-server distributed computing

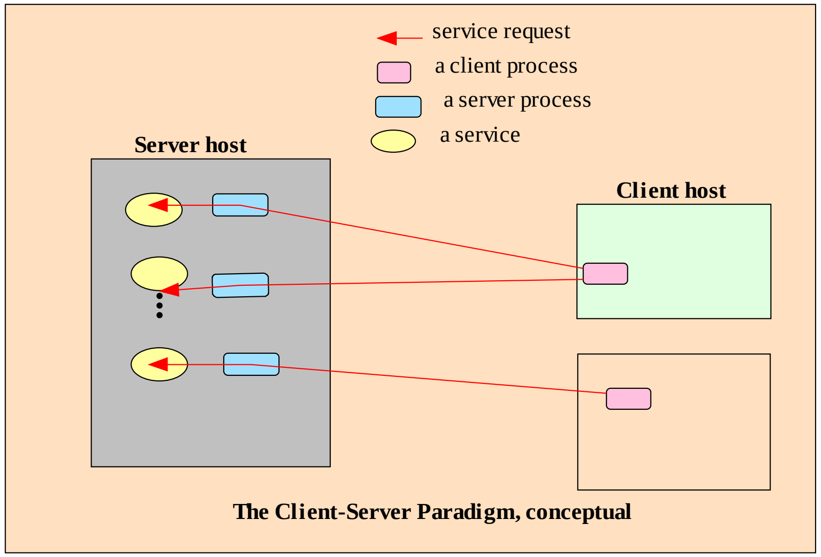
In the client-server system architecture, the terms clients and servers refer to computers, while in the client-server distributed computing paradigm, the terms refer to processes.

Client-server, an overloaded term



Client-Server System Architecture

Client hosts make use of services provided on a server host.



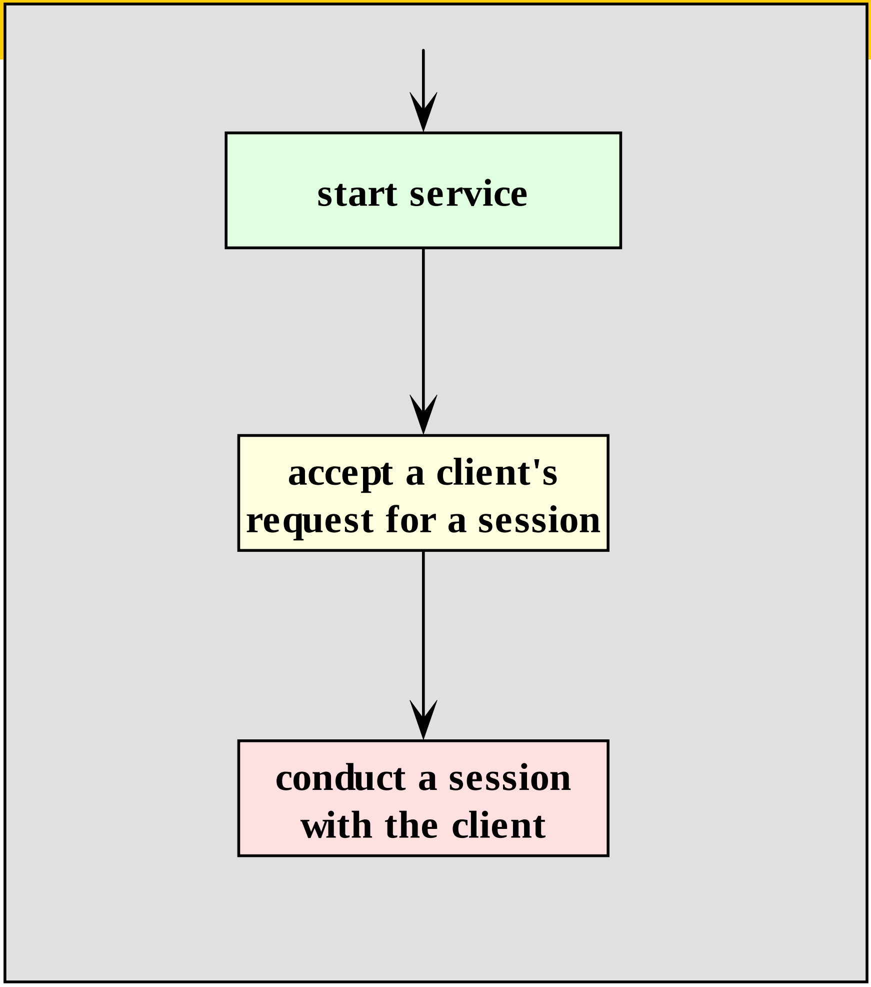
Client-Server Computing Paradigm

Client processes (objects) make use of a service provided by a server process (object) running on a server host.

A protocol/service session

In the context of the client-server model, we will use the term **session** to refer to the interaction between the server and one client. The service managed by a server may be accessed by multiple clients who desire the service, sometimes concurrently. Each client, when serviced by the server, engages in a separate session with the server, during which it conducts a dialog with the server until the client has obtained the service it required

A service session


start service
accept a client's request for a session
conduct a session with the client

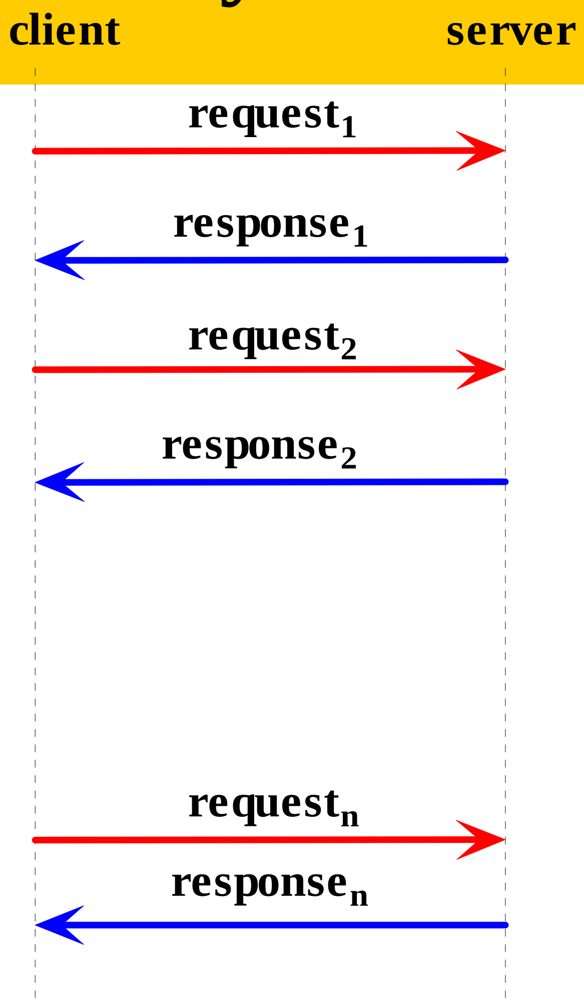
The Protocol for a Network Service

- A protocol is needed to specify the rules that must be observed by the client and the server during the conductin of a service. Such rules include specifications on matters such as (i) how the service is to be located, (ii) the sequence of interprocess communication, and (iii) the representation and interpretation of data exchanged with each IPC.
- On the Internet, such protocols are specified in the RFCs.

Locating the service

- A mechanism must be available to allow a client process to locate a server for a given service.
- A service can be located through the address of the server process, in terms of the host name and protocol port number assigned to the server process. This is the scheme for Internet services. Each Internet service is assigned to a specific port number. In particular, a well-known service such as ftp, HTTP, or telnet is assigned a default port number reserved on each Internet host for that service.
- At a higher level of abstraction, a service may be identified using a logical name registered with a registry, the logical name will need to be mapped to the physical location of the server process. If the mapping is performed at runtime (that is, when a client process is run), then it is possible for the service's location to be dynamic, or moveable

The interprocess communications and event synchronization

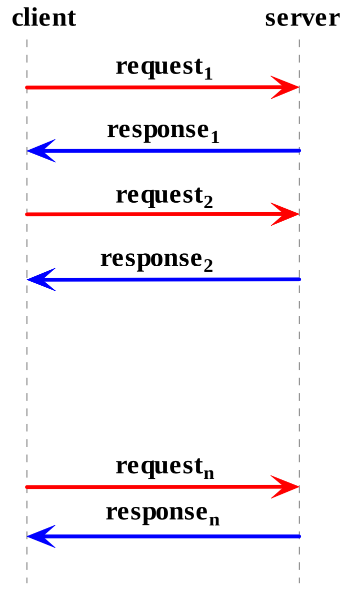


Implementation of a network service

Any implementation of the client or server program for this service is expected to adhere to the specification for the protocol, including how the dialogs of each session should proceed. Among other things, the specification defines (i) which side (client or server) should speak first, (ii) the syntax and semantic of each request and response, and (iii) the action expected of each side upon receiving a particular request or response.

The interprocess communications and event synchronization

- Typically, the interaction of the client and server processes follows a request-response pattern.



Session IPC examples

The dialog in each session follows a pattern prescribed in the protocol specified for the service.

Daytime service [RFC867]:

- **Client**: Hello, <client address> here. May I have a timestamp please.
- **Server**: Here it is: (time stamp follows)

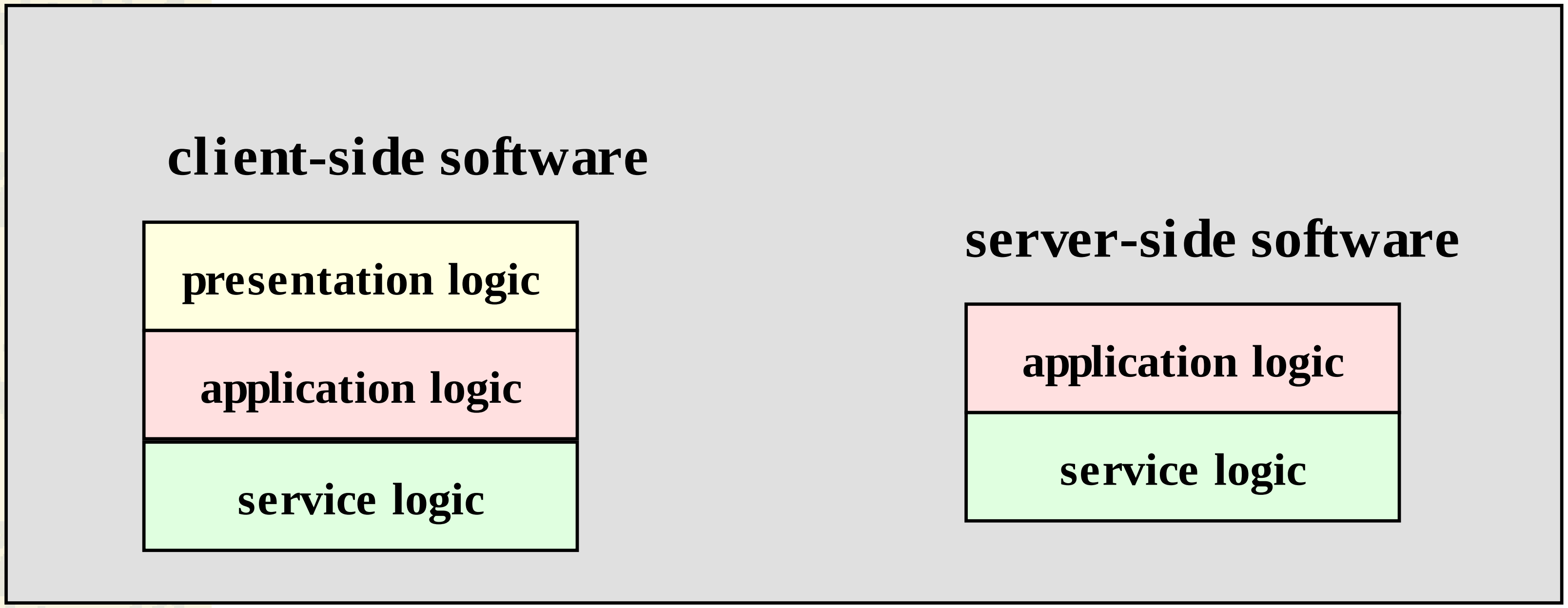
World Wide Web session:

- **Client**: Hello, <client address> here.
- **Server**: Okay. I am a web server and speaks protocol HTTP1.0.
- **Client**: Great, please get me the web page index.html at the root of your document tree.
- **Server**: Okay, here's what's in the page: (contents follows).

Client-server protocol data representation

- Part of the specification of a protocol is the syntax and semantics of each request and response.
- The choice of data representation depends on the nature and the needs of the protocol.
- Representing data using text (character strings) is common, as it facilitates data marshalling and allows the data to be readable by human.
- Most well known Internet protocols are client-server, request-response, and text-base.

Software Engineering for a Network Service



Daytime Client-server using Connectionless Datagram Socket

Client-side presentation logic

DaytimeClient1.java encapsulates the client-side presentation logic; that is, it provides the interface for a user of the client process. You will note that the code in this class is concerned with obtaining input (the server address) from the user, and displaying the output (the timestamp) to the user. To obtain the timestamp, a method call to a "helper" class, ***DaytimeClientHelper1.java***, is issued. This method hides the details of the application logic and the underlying service logic. In particular, the programmer of DaytimeClient1.java need not be aware of which socket types is used for the IPC.

Daytime Client-server using Connectionless Datagram Socket - continued

Client-side Application logic

The ***DaytimeClientHelper1.java*** class (Figure 6b) encapsulates the client-side application logic. This module performs the IPC for sending a request and receiving a response, using a specialized class of the ***DatagramSocket, myClientDatagramSocket***. Note that the details of using datagram sockets are hidden from this module. In particular, this module does not need to deal with the byte array for carrying the payload data.

Service logic

The *MyClientDatagram.java* (Figure 6c) class provides the details of the IPC service, in this case using the datagram socket API.

Advantages of separating the layers of logic

- It allows each module to be developed by people with special skills to focus on a module for which they have expertise. Software engineers who are skilled in user interface may concentrate on developing the modules for the presentation logic, while those specializing in application logic and the service logic may focus on developing the other modules.

- The separation allows modifications to be made to the logic at the presentation without requiring changes to be made at the lower layers. For example, the user interface can be changed from text-mode to graphical without requiring any change be made to the application logic or the service logic. Likewise, changes made in the application logic should be transparent to the presentation layer, as we will soon illustrate with an example client-server application.

Server-side software

Presentation logic

Typically, there is very little presentation logic on the server-side. In this case, the only user input is for the server port, which, for simplicity, is handled using a command-line argument.

Application logic

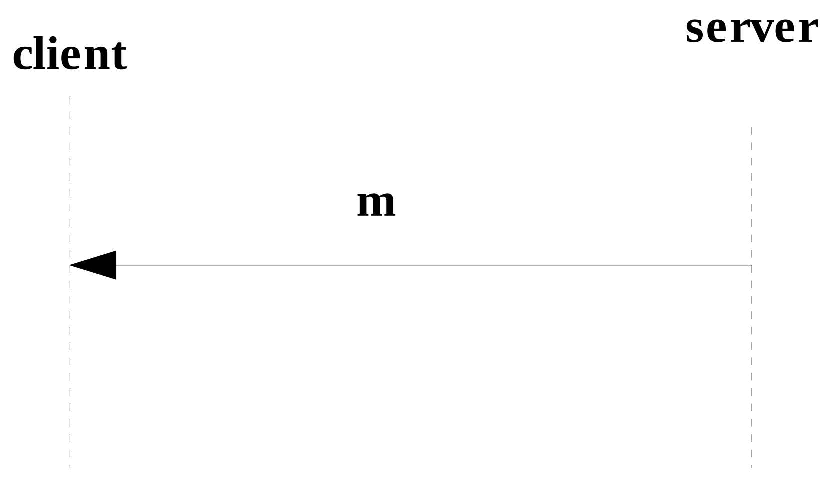
The ***DaytimeServer1.java*** class encapsulates the server-side application logic. This module executes in a forever loop, waiting for a request form a client and then conduct a service session for that client. The module performs the IPC for receiving a request and sending a response, using a specialized class of the ***DatagramSocket, myServerDatagramSocket***. Note that the details of using datagram sockets are hidden from this module. In particular, this module does not need to deal with the byte array for carrying the payload data.

Service logic

The ***MyServerDatagram.java*** class provides the details of the IPC service, in this case using the datagram socket API.

Example protocol: daytime

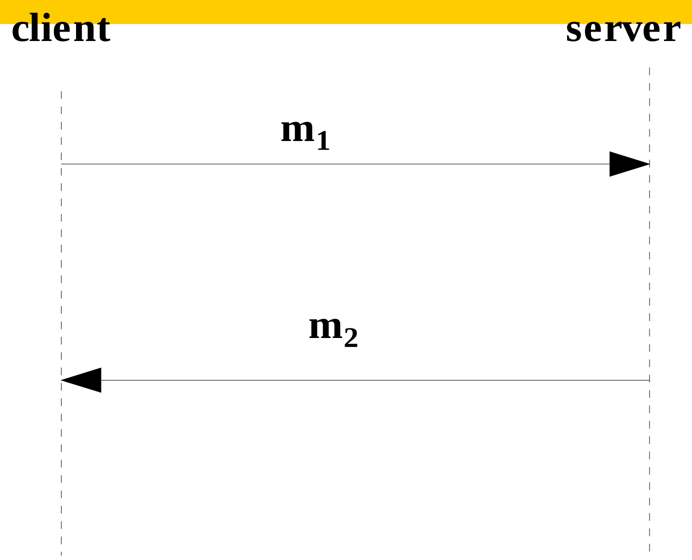
Defined in RFC867



sequence diagram

data representation: text (character strings)
data format:
m : contains a timestamp, in a format such as
Wed Jan 30 09:52:48 2002

Daytime Protocol



sequence diagram

data representation: text (character strings)
data format:
m1; a null message - contents will be ignored.
m2 : contains a timestamp, in a format such as
Wed Jan 30 09:52:48 2002

Daytime Protocol Implementation

Sample 1 – using connectionless sockets:

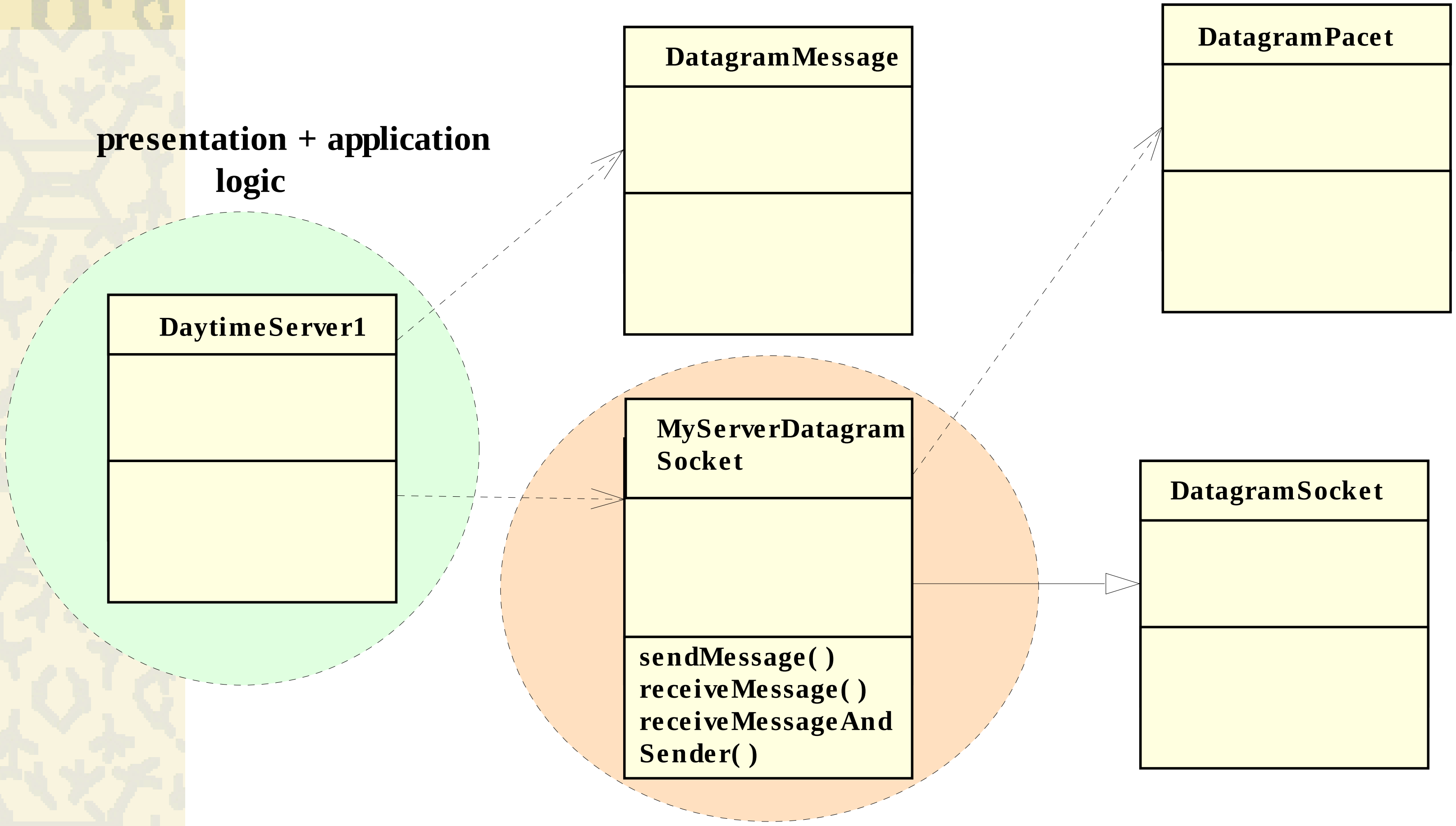
DaytimeServer1.java

DaytimeClient1.java

The getAddress and getPort Methods

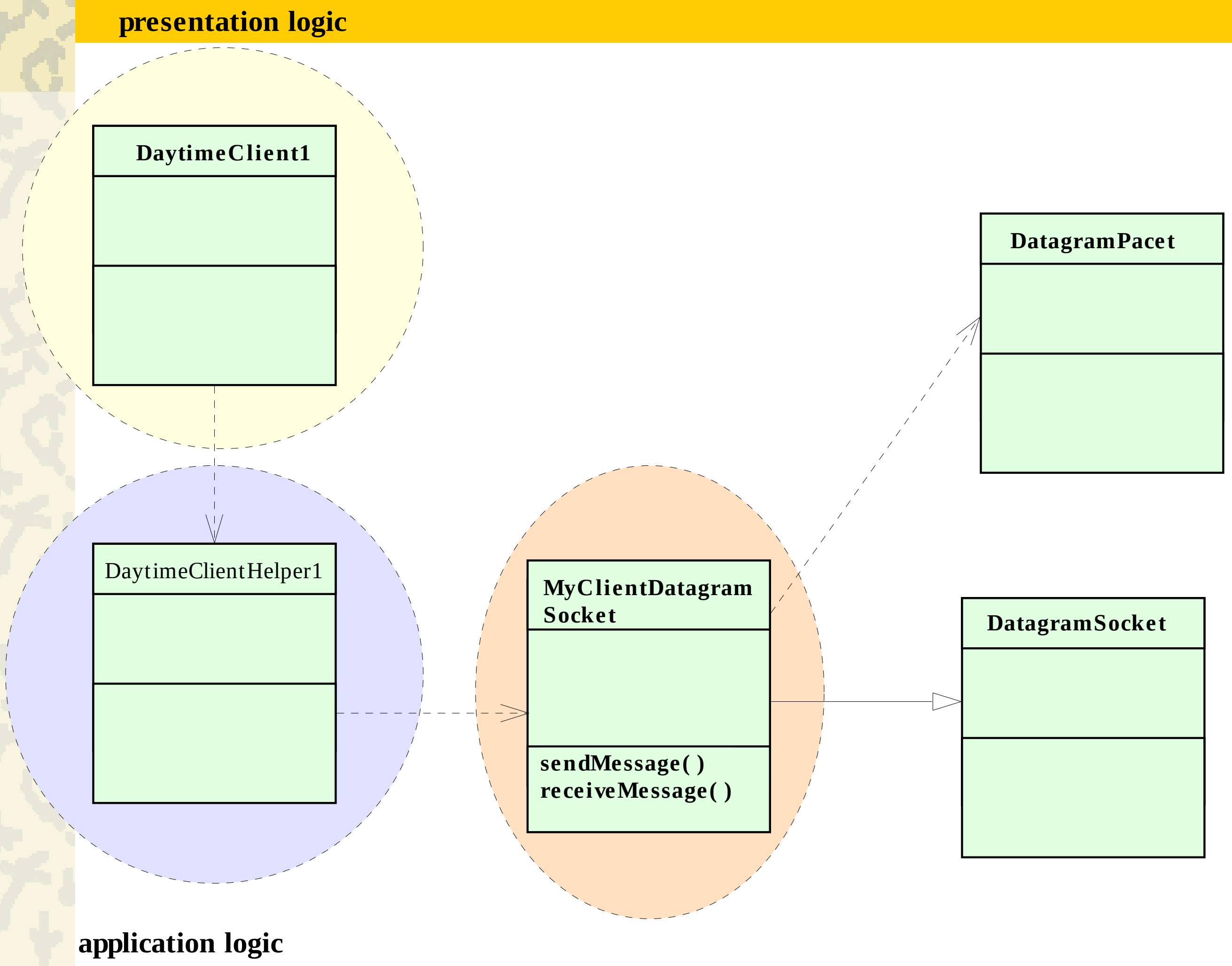
Method (of DatagramPacket class)	**Description**
public InetAddress **getAddress**( )	Return the IP address of the remote host from a socket of which the datagram was received.
public int **getPort**( )	Return the port number on the remote host from a socket of which the datagram was received.

UML Diagram for the Datagram Daytime server



service logic

UML Diagram for the Datagram Daytime Client



Daytime Protocol Implementation

Connection-oriented Daytime Server Client:

Sample 2 – using connection-oriented sockets:

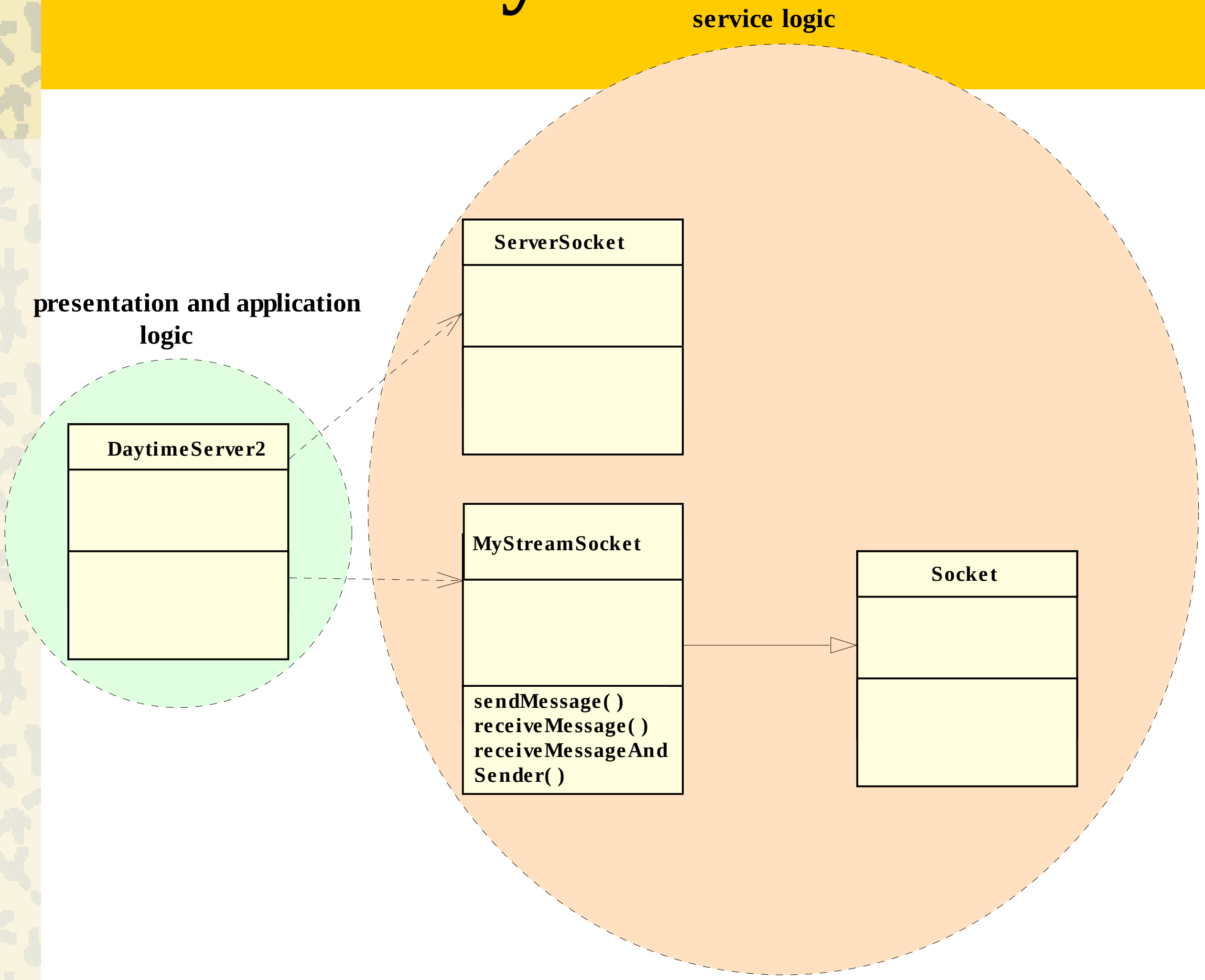
DaytimeServer2.java

DaytimeClient2.java

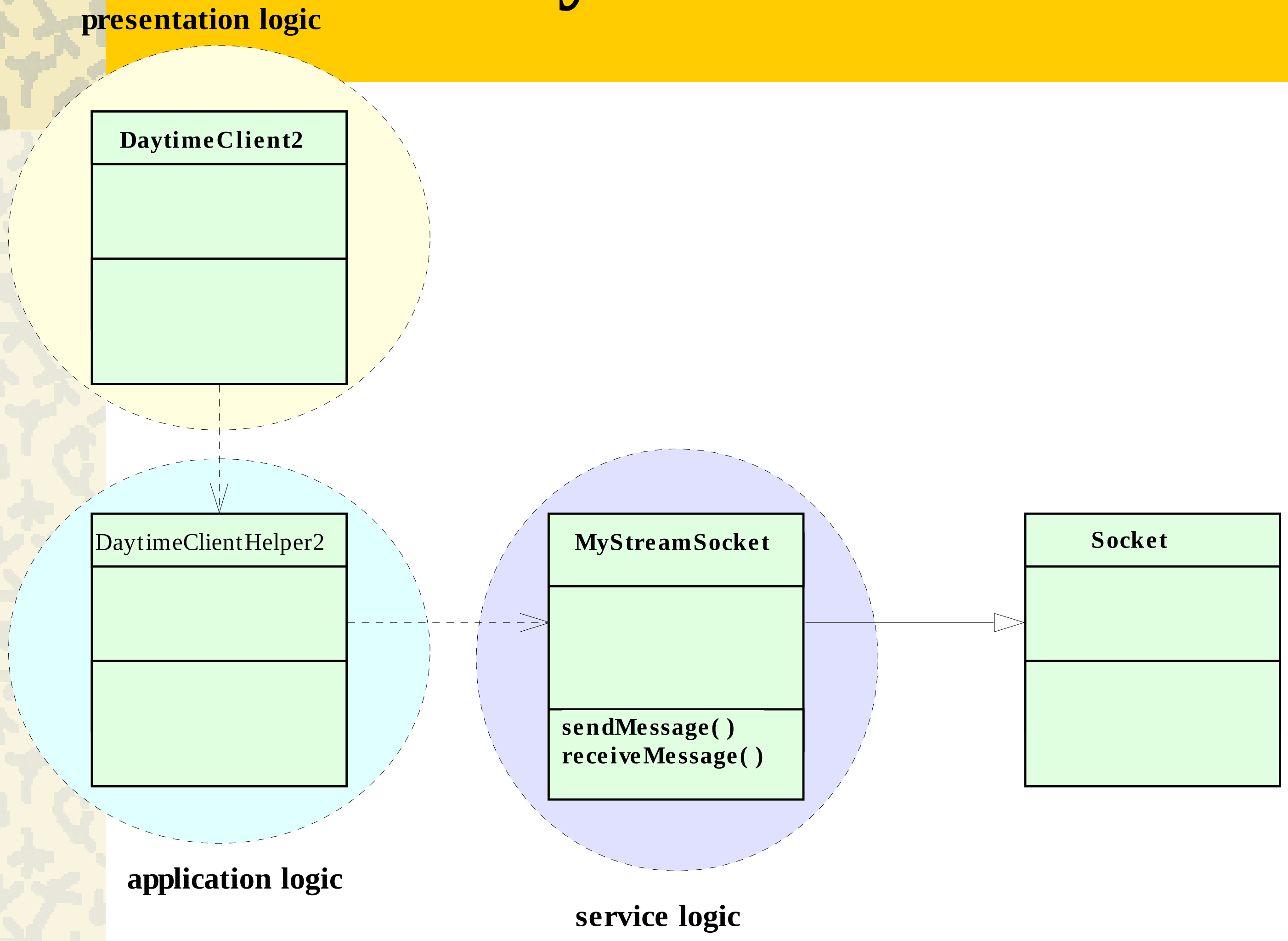
DaytimeClientHelper2.java

MyStreamSocket.java

UML Diagram for stream mode Daytime Server



UML Diagram for stream mode Daytime Client



Testing a Network Service

- Because of its inherent complexity, network software is notoriously difficult to test.
- Use the three-layered software architecture and modularize each layer on both the client and the server sides.
- Use an incremental or stepwise approach in developing each module. Starting with stubs for each method, compile and test a module each time after you put in additional details.
- Develop the client first. It is sometimes useful to employ an Echo server (to be introduced in the next section) which is known to be correct and which uses a compatible IPC mechanism to test the client independent of the server; doing so allows you to develop the client independent of the server.
- Use diagnostic messages throughout each program to report the progress of the program during runtime.
- Test the client-server suite on one machine before running the programs on separate machine.