CS 580 Client-Server Programming
Fall Semester, 2002
Doc 1 Introduction
Contents

References

Code Complete by Steve McConnell

Course Introduction
Course Outline

Introduction
XML-RPC
Source Version Control
Client Development Issues
Concurrency
GUI
Testing
Network Basics
Server Types & Structure
Client-Server Protocols
Logging
Databases
Security
Web based Applications
  CGI, Fast-CGI, Servlets
Advanced topics

This outline will be changed during the semester.

Main Points of Class

Common design issues & solutions for building a server

Issues in designing a client-server network protocol

Handling Concurrency

Accessing databases

Programming issues dealing with working on client-server code

Programming languages for the Course

Java

- We will be using parts of JDK 1.4
- NIO contains richer socket support
- Logging

Smalltalk – VisualWorks 7

- Improved code browsers & support for network programming

What does it mean to "Know" Java or Smalltalk

Basic syntax of the language

Core API

No one knows the entire API of either language

You should have good grasp of the common or core API

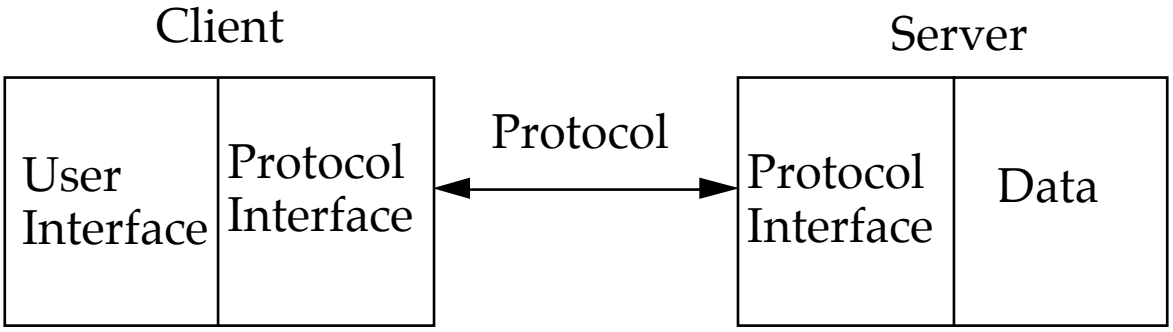
Collections, Files, Exceptions, Streams

Language culture - Ways of doing things in each language

- Java Doc
- Searching the API
- Compiling/running code
- Using Smalltalk browsers
- Naming conventions

Object-oriented programming

Introduction to Client-Server
What is Client-Server?



Client

Application that initiates peer-to-peer communication

Translate user requests into requests for data from server via protocol

GUI often used to interact with user

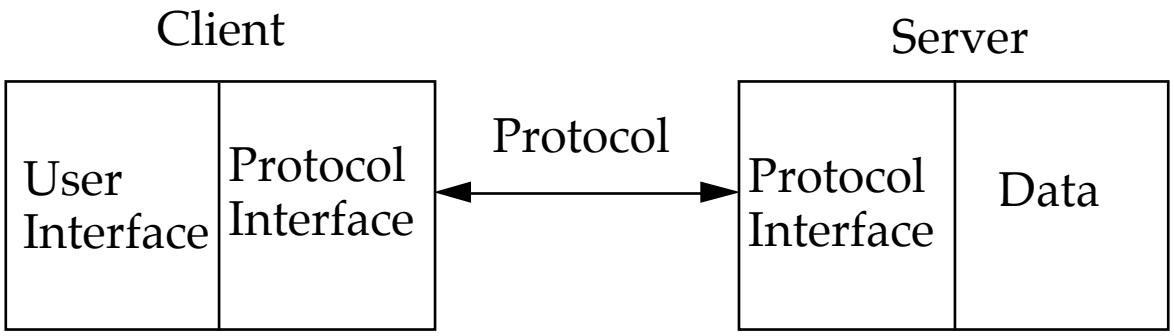
Server

Any program that waits for incoming communication requests from a client

Extracts requested information from data and return to client

Common Issues

- Authentication
- Authorization
- Data Security
- Privacy
- Protection
- Concurrency



Example: World Wide Web (WWW)

Data

Server normally provides data to clients

Often utilizes some data base

WWW data is HyperText Markup Language (html) files

```
<!DOCTYPE HTML SYSTEM "html.dtd">
<HTML>
<HEAD><TITLE>
Client Server Programming
</TITLE></HEAD>
<BODY>
<H2>Client Server Programming</H2>
<HR>
```

Protocol

How the client and server interact

Glue that makes client-server work

Involves using low level network protocols and application specific protocols

Designing application specific protocols is very important

WWW uses the HyperText Transfer Protocol

Request = SimpleRequest | FullRequest

SimpleRequest = GET <uri> CrLf

FullRequest = Method URI ProtocolVersion CrLf
[*<HTRQ Header>]
[<CrLf> <data>]

<Method> = <InitialAlpha>

ProtocolVersion = HTTP/1.0

uri = <as defined in URL spec>

<HTRQ Header> = <Fieldname> : <Value> <CrLf>

<data> = MIME-conforming-message

Protocol Choices

- Text Based

  Transmit ASCII or Unicode between machines

  HTTP is common transport layer

  XML becoming common

  SOAP new XML standard

- Binary

  Transmit objects between machines

  Faster development time

  RMI, Corba are examples

What this Course is not

An advanced (or beginning) Networking course

OSI Model

7	Application	Process Layer
6	Presentation	
5	Session	
4	Transport	
3	Network	
2	Data Link	
1	Physical	

How to use a client builder application/system

Powerbuilder

What this Course covers

Skills & knowledge required to build client-server applications

What Client-Server Requires of a Programmer

- Designing robust protocols
- Network programming
- Designing usable computer-human interfaces
- Good documentation skills
- Good debugging skills
- Understand the information flow of the company/customer
- Mastery of concurrency
- Multi-platform development
- Database programming
- Security

Programming Issues
Naming Convention for Classes, Variables & Methods

Smalltalk and Java use the same basic naming convention

- Use full words – avoid abrvtns
- “Camel” style case
  First letter of each word capitalized except for the first word

  Class names start with a capital letter

Names

"Finding good names is the hardest part of OO Programming"

"Names should fully and accurately describe the entity the variable represents"

What role does the variable play in the program?

Data Structure	**Role, function**
InputRec	EmployeeData
BitFlag	PrinterReady

Some Examples of Names, Good and Bad

TrainVelocity	Velt, V, X, Train
CurrentDate	CD, Current, C, X, Date
LinesPerPage	LPP, Lines, L, X

Names should be as short as possible and still convey meaning to the reader

Comments

"Comments are easier to write poorly than well, and comments can be more damaging than helpful"

What does this do?

```
for i := 1 to Num do
 MeetsCriteria[ i ] := True;
for  i := 1 to Num / 2  do begin
 j := i + i;
 while ( j <= Num ) do begin
  MeetsCriteria[ j ] := False;
  j := j + i;
 end;
for i := 1 to Mun do
 if MeetsCriteria[ i ] then
  writeln( i, ' meets criteria ' );
```

How many comments does this need?

```
for PrimeCandidate:= 1 to Num do
   IsPrime[ PrimeCandidate] := True;

for  Factor:= 1 to Num / 2  do begin
   FactorableNumber := Factor + Factor ;
   while ( FactorableNumber <= Num ) do begin
      IsPrime[ FactorableNumber ] := False;
      FactorableNumber := FactorableNumber + Factor ;
   end;
end;

for PrimeCandidate:= 1 to Num do
   if IsPrime[ PrimeCandidate] then
      writeln( PrimeCandidate, ' is Prime ' );
```

Good Programming Style is the Foundation of Well Commented Program

Commenting Paragraphs of Code

Write comments at the level of the code's intent

Comment the why rather than the how

Make every comment count

Document surprises

Avoid abbreviations

How verses Why

How

```
/* if allocation flag is zero */
if ( AllocFlag == 0 ) ...
```

Why

```
/* if allocating a new member */
if ( AllocFlag == 0 ) ...
```

Even Better

```
/* if allocating a new member */
if ( AllocFlag == NEW_MEMBER ) ...
```

Summary comment on How

```
{ check each character in "InputStr" until a
  dollar sign is found or all characters have
  been checked }

Done   := false;
MaxPos := Length( InputStr );
i      := 1;
while ( (not Done) and (i <= MaxLen) ) begin
   if ( InputStr[ i ] = '$' ) then
      Done := True
   else
      i := i + 1
end;
```

Summary comment on Intent

```
{ find the command-word terminator }

Done   := false;
MaxPos := Length( InputStr );
i      := 1;

while ( (not Done) and (i <= MaxPos ) ) begin
   if ( InputStr[ i ] = '$' ) then
      Done := True
   else
      i := i + 1
end;
```

Summary comment on Intent with Better Style

```
{ find the command-word terminator }

FoundTheEnd      := false;
MaxCommandLength := Length( InputStr );
Index            := 1;

while ((not FoundTheEnd) and
       (Index <= MaxCommandLength)) begin

   if ( InputStr[ Index ] = '$' ) then
      FoundTheEnd := True;
   else
      Index := Index + 1;
end;
```

Commenting Data Declarations

Comment the units of numeric data

Comment the range of allowable numeric values

Comment coded meanings

```
var
  CursorX:   1..MaxCols;      { horizontal screen position of cursor }
  CursorY:   1..MaxRows;      { vertical position of cursor on screen }

  AntennaLength:   Real;      { length of antenna in meters: >= 2 }
  SignalStrength: Integer;    { strength of signal in kilowatts: >= 1 }

  CharCode:  0..255;      { ASCII character code }
  CharAttib:  Integer;    { 0=Plain; 1=Italic; 2=Bold  }
  CharSize:   4..127;     { size of character in points }
```

Comment limitations on input data

Document flags to the bit level