# Rational Networking Training

Document Control Number: 1033
Rev. 1.0, April 1987



Rational
1501 Salado Drive
Mountain View, California 94043

# Rational Networking Training

## Slides

## Contents


Rational Networking—TCP/IP
- Course Introduction 1
- Product Introduction 4
- Product Components 7
- Networking Applications 14
- Terminology 17

Telnet
- Introduction 21
- How to Form and Terminate a Telnet Session 25
- How to Resume a Telnet Session 30
- How to Manage Multiple Sessions 33
- Use of Telnet Defaults 36
- Use of Telnet 41

FTP
- Introduction 45
- How to Form and Terminate an FTP Session 49
- How to Transfer Single Files 52
- Transfer Types 57
- Mapping of Names 65
- Naming Contexts 69
- Use of FTP Defaults 77
- How to Transfer Multiple Files 80

Additional FTP Operations 89

RPC

Introduction 95
Terminology 101
Use of RPC 106

# Seminar Outline

Rational Networking—TCP/IP

- Course Introduction
  Product Introduction
  Product Components
  Networking Applications
  Terminology

Telnet

FTP

RPC

# Course Objectives

- Introduce the features and benefits of the Rational Networking—TCP/IP product
- Introduce the fundamental concepts and mechanisms required for understanding how to use basic product features
- Provide hands-on experience in using these concepts and mechanisms

# Documentation

- *Rational Networking Training*
- *Rational Networking—TCP/IP*
- *Rational Environment Reference Summary*
- *Rational Environment Basic Operations*

# Seminar Outline

Rational Networking—TCP/IP

- Course Introduction
- Product Introduction
- Product Components
- Networking Applications
- Terminology

Telnet

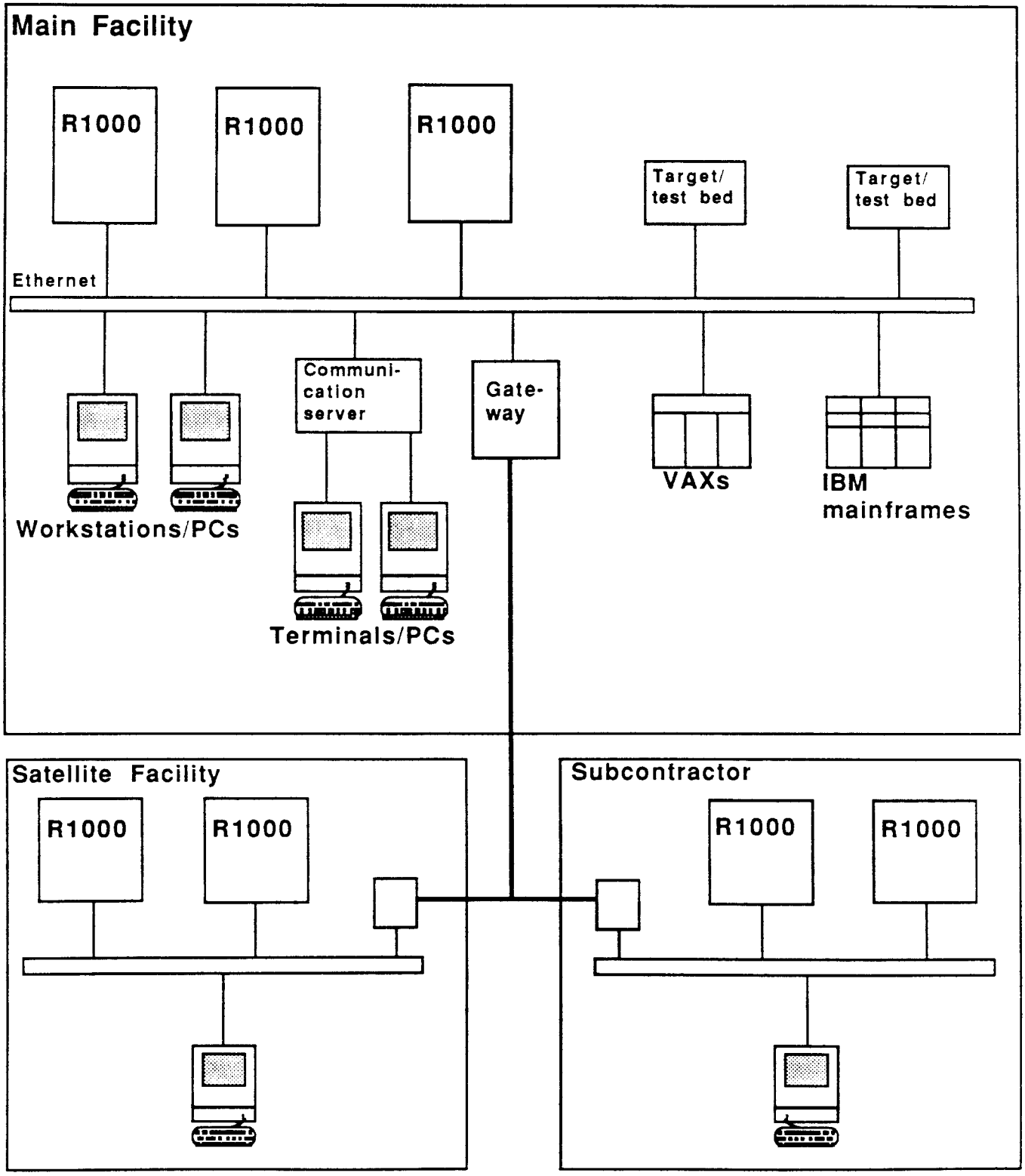
FTP

RPC

# Rational Networking—TCP/IP

- Provides the means for integrating the Rational Environment into an existing computing base
- Enables distributed machine development in a local area and across long distances
- Is based on industry-standard protocols enabling interface to off-the-shelf solutions available for many computers
  - — Ethernet
  - — TCP/IP
  - — FTP
  - — Telnet
- Forms foundation for host/target development

# Rational Network Architecture



# Seminar Outline

Rational Networking—TCP/IP
- Course Introduction
- Product Introduction
- • Product Components
- Networking Applications
- Terminology

Telnet

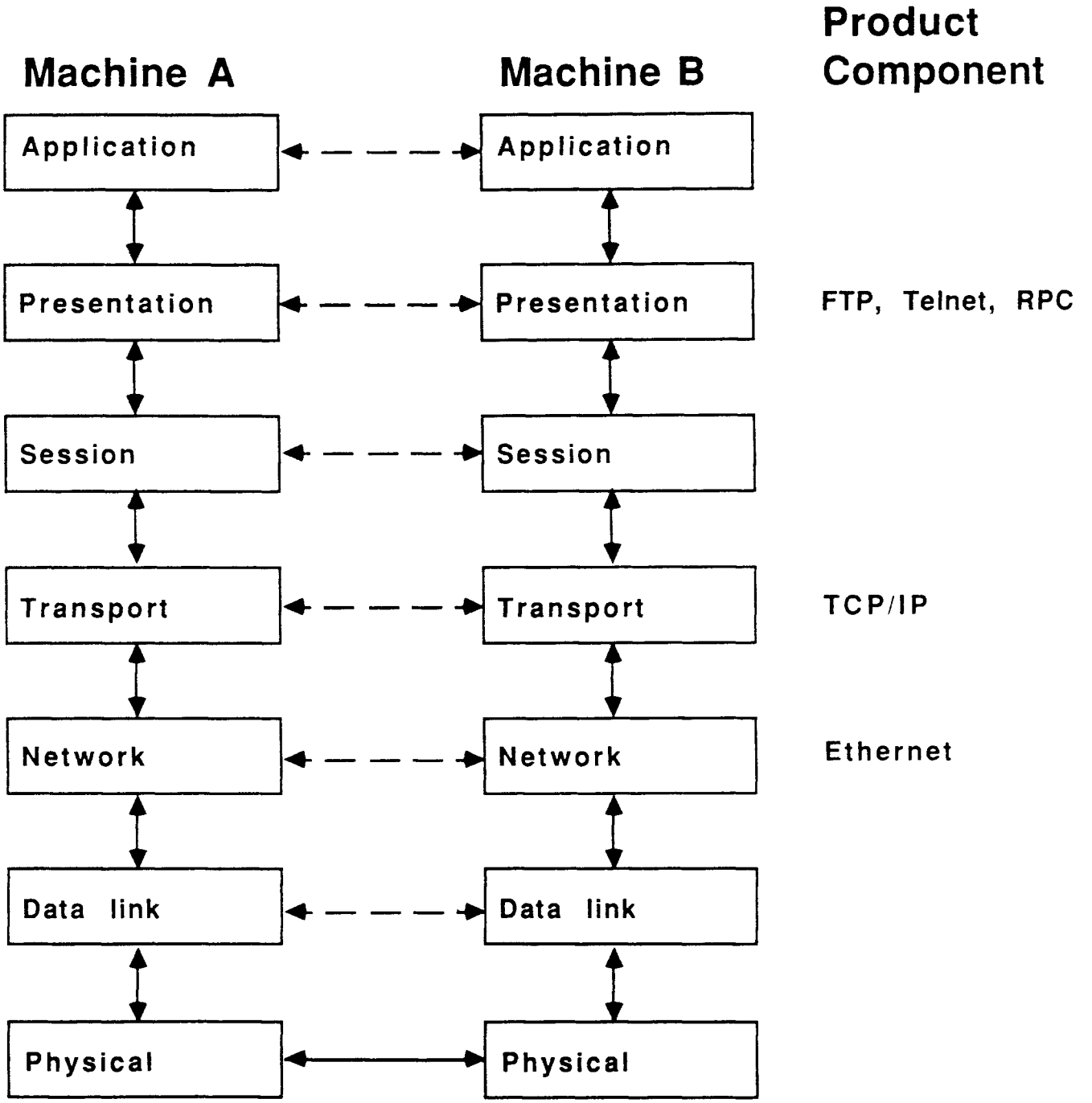
FTP

RPC

# Overview of Product Components

- Ethernet
- TCP/IP: Transport layer protocol
- FTP: File transfer protocol facility
- Telnet: Remote login facility
- RPC: Remote procedure call facility

# ISO Model


Machine A
Machine B
Product Component
Application
Application
Presentation
Presentation
FTP, Telnet, RPC
Session
Session
Transport
Transport
TCP/IP
Network
Network
Ethernet
Data link
Data link
Physical
Physical

# Ethernet—TCP/IP

- Allows interface to any computer system using Ethernet and running TCP/IP as the transport protocol
- Offers 1 to 2 Mbits per second effective throughput
- Offers reliable transmission of bytes
- Provides compatibility with ARPANET and MILNET

# Telnet Remote Login

- Allows RS232 devices connected to a terminal server to establish logical connections to an R1000
- Allows a user at a terminal on one machine to log into another machine on the network
- Enables a user connected to an R1000 to log into other systems on the network with no need for point-to-point terminal interconnection
- Enables a user of any terminal supported by the Rational Environment to log in and use the R1000 facilities while directly connected to a different machine
- Is a widely used standard

# FTP—File Transfer Protocol

- Allows transporting text and arbitrary binary data files among R1000s and other computers
- Is fully compatible with the U.S. Department of Defense (DoD) standard FTP file transfer protocol
- Has a command and programmatic interface

# RPC—Remote Procedure Call

- Allows a program on one system to call facilities exported by programs on other systems
- Allows values of all Ada types (except access and task types) to be passed as parameters or results
- Enables exceptions to be propagated back to the caller
- Extends the semantics of subprogram calls across multiple machines

# Seminar Outline

Rational Networking—TCP/IP
- Course Introduction
- Product Introduction
- Product Components
- Networking Applications
- Terminology

Telnet

FTP

RPC

# Overview

- Multiple R1000 projects
  - — Development of large projects distributed across multiple R1000 hosts
- Multihost development
  - — Development of projects on the R1000 with access to tools on non-Rational systems
- Host/target development
  - — Development of software on the R1000 for execution on target computers

# Use of Product Components

- FTP: Transfer source files or data objects between computer systems for processing or storage
- Telnet: Be simultaneously logged into several computers from a single terminal
- RPC: From one machine invoke project tools, databases, and other software resident on a different machine
- RPC: Build a test scaffold on an R1000 (for functional integration) that accesses target-dependent libraries or device drivers on other systems

# Seminar Outline

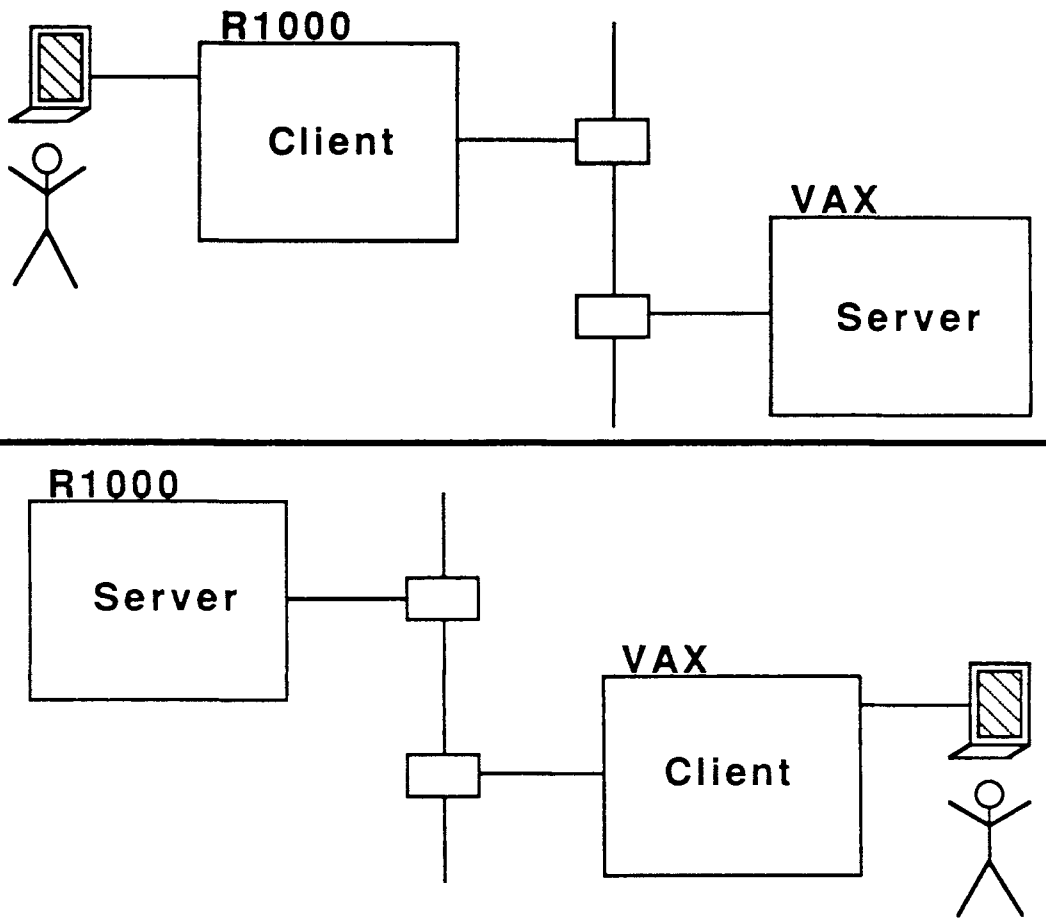
Rational Networking—TCP/IP
- Course Introduction
- Product Introduction
- Product Components
- Networking Applications
- Terminology

Telnet

FTP

RPC

# Clients and Servers




- A *client* is the machine attached to the terminal that initiates communication
- A *server* must be present on the receiving machine for communication to take place

# Connections and Sessions

- To begin communication, a client asks a server to establish a *connection.* If the server responds positively, a connection is established
- When a connection is established, a *session* is considered to be under way
- The client must log into this session before issuing further commands or requests
- To terminate communication, the client must log out and disconnect from the session

# Symbology

- Used in slides and exercise instructions
- `>>Host Name<<` means enter string that is name of host
- Typical examples
  - `>>Host Name<<`
    `>>Remote Host Name<<`
  - `>>Username<<`
    `>>Remote Username<<`
  - `>>Password<<`
    `>>Remote Password<<`
  - `>>Filename<<`
  - `>>Pathname<<`
    `>>Remote Pathname<<`

# Seminar Outline

Rational Networking—TCP/IP

Telnet

- Introduction
  How to Form and Terminate a Telnet Session
  How to Resume a Telnet Session
  How to Manage Multiple Sessions
  Use of Telnet Defaults
  Use of Telnet

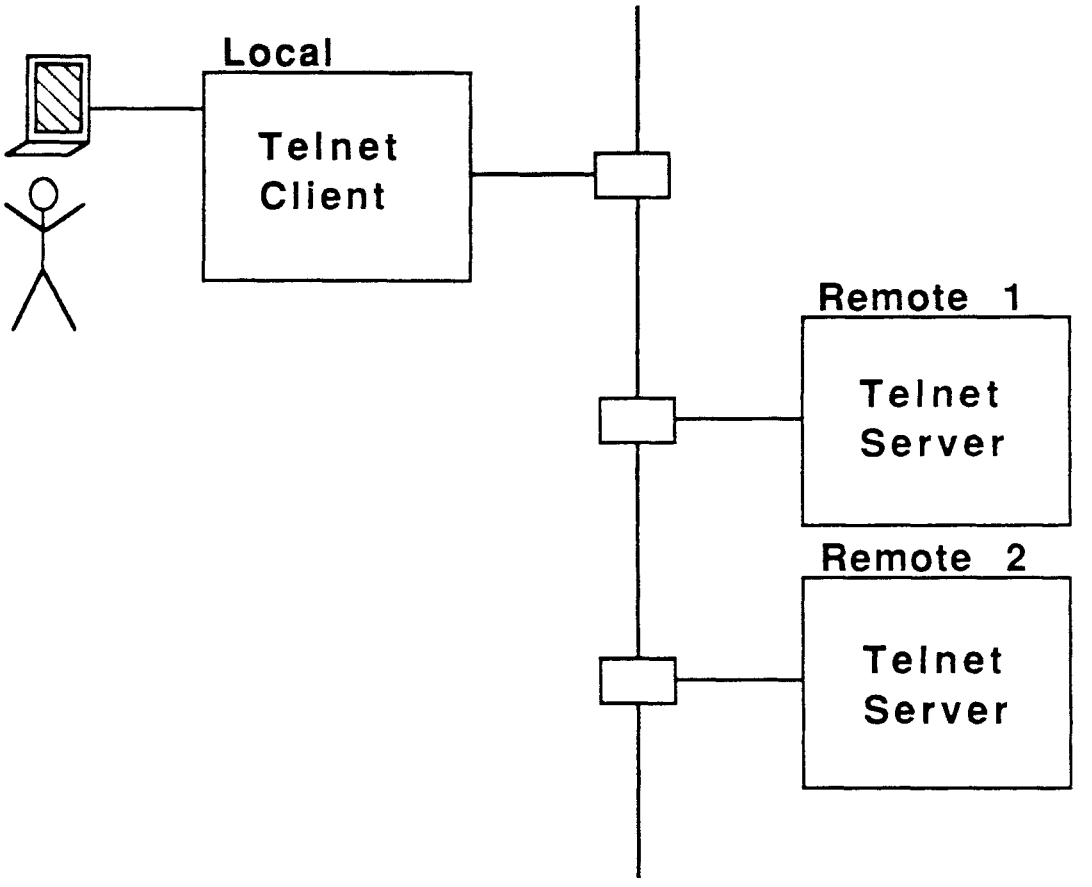
FTP

RPC

# Telnet

- Enables login to an R1000 from other systems on the same network
- Enables login from an R1000 to other systems on the same network
- Can be used with any terminal (or emulator) supported by the Rational Environment
- Supports multiple, simultaneous sessions
- Enables RS232 devices connected to a terminal server to establish logical connections to an R1000

# Telnet Sessions




- A client can form multiple Telnet sessions with a single server and/or multiple servers
- The client logs into each session separately

# Telnet Sessions, cont.

- A user can be simultaneously logged into multiple contexts from one terminal
- A user can switch between sessions rapidly

# Seminar Outline

## Rational Networking—TCP/IP

Telnet

- Introduction
- How to Form and Terminate a Telnet Session
- How to Resume a Telnet Session
- How to Manage Multiple Sessions
- Use of Telnet Defaults
- Use of Telnet

FTP

RPC

# Basic Model

- Form a Telnet session by connecting to a remote machine
- Log into the Telnet session
  - — Gain access to some context on the remote machine
- Terminate the session when done working in this context
  - — Return automatically to the initial local context

# How to Form a Telnet Session

- Form a Telnet session by connecting to a remote machine and specifying the machine name
  - —
```
Telnet.Connect (
    Remote_Machine => ">>Host Name<<");
```
  - — The terminal is now in ANSI mode
  - — The terminal appears to be directly connected to the remote machine
  - — Telnet remembers the initial local context
- Log into the Telnet session
  - — Use the normal login procedure for gaining access to the desired context on the particular remote

# How to Terminate a Telnet Session

- The procedure varies with different Telnet servers
- With most Telnet servers
  - — You log out from the remote
  - — The machines are disconnected
  - — You are returned to the initial local context
  - — The Telnet session is terminated
- If the session is not disconnected automatically
  - — Escape from the Telnet session: Break
  - — You are returned to the initial local context
  - — Disconnect the Telnet session:

```
Telnet.Disconnect
```

# How to Terminate a Telnet Session, cont.

— The machines are disconnected

— The Telnet session is terminated

# Seminar Outline

## Rational Networking—TCP/IP

Telnet

- Introduction
- How to Form and Terminate a Telnet Session
- How to Resume a Telnet Session
- How to Manage Multiple Sessions
- Use of Telnet Defaults
- Use of Telnet

FTP

RPC

# Basic Model

- Form a Telnet session and log into it
- Return to the initial local context, leaving the Telnet session intact
  - — Escape from the current session: Break
  - — Telnet remembers the remote context so you can resume working in it without logging in again
  - — The machines remain connected
- Resume the Telnet session
  - — Return to the remote context and continue work without logging in again
- Terminate the Telnet session
  - — Log out from the remote context

# Basic Mechanisms

- Form a session and log in as before
- From the remote context, return to the initial local context, leaving the Telnet session intact
  - — Break (Control Meta Mark)
- Resume the Telnet session, using the same mechanism you used when you initially connected
  - — From the local context

```
Telnet.Connect (

   Remote_Machine => ">>Host Name<<");
```

  - — No need to log in this time; use the Editor.Screen.Redraw command (Control L) to redraw the screen
- Terminate the Telnet session, logging out as before

# Seminar Outline

Rational Networking—TCP/IP

Telnet
- Introduction
- How to Form and Terminate a Telnet Session
- How to Resume a Telnet Session
- How to Manage Multiple Sessions
- Use of Telnet Defaults
- Use of Telnet

FTP

RPC

# How to Form Multiple Sessions

- Form a first Telnet session and log in
- Return to the initial local context, leaving the session intact
- Form a second Telnet session and log in
  - — From the local context

```
Telnet.Connect (
    Remote_Machine => ">>Host Name<<",
    Session => 2);
```

  - — Specify a new session number
  - — Log into the desired context
- Return to the initial local context, leaving the second session intact

# How to Form Multiple Sessions, cont.

- Show all active Telnet sessions:

  ```
  Telnet.Show_Sessions;
  ```

- Resume either Telnet session

  — Specify the appropriate session number

- Terminate all sessions by logging out

# Seminar Outline

Rational Networking—TCP/IP

Telnet
- Introduction
- How to Form and Terminate a Telnet Session
- How to Resume a Telnet Session
- How to Manage Multiple Sessions
- Use of Telnet Defaults
- Use of Telnet

FTP

RPC

# Command Default Parameters

- Many commands have default parameter values that can be used to save time and typing
  - — The user can default some parameter values and enter others
  - — Default values can be used repeatedly, without entering them each time a command is invoked
- Many default Telnet parameter values are assigned indirectly using session switches
  - — The user doesn't get simply a system default value before command invocation
  - — The user can provide a default value before command invocation
  - — This provides even greater flexibility and customization than ordinary command default parameters

# Alteration of Telnet Defaults

- Indirectly assignable default Telnet parameters begin with the prefix `Telnet_Profile`
- A list of all initial Telnet default values can be viewed by looking at the user session switches for the Telnet function
- To alter a default value for a parameter with the `Telnet_Profile` prefix, set the corresponding session switch

# Examples of Default Parameters

- `Telnet_Profile.Remote_Machine`
  - — This parameter allows connecting repeatedly to the same machine without typing in its name each time
  - — The user assigns the `Telnet.Remote_Machine` switch to the desired machine name
- `Telnet_Profile.Escape`
  - — This parameter allows any key to be used for switching from remote to local context while leaving a Telnet session intact
  - — The user assigns the `Telnet.Escape` switch to the desired key

# Examples of Default Parameters, cont.

- `Telnet_Profile.Escape_On_Break`
  - — This parameter allows Break to be disabled or enabled for switching from remote to local context while leaving a Telnet session intact
  - — The user assigns the `Telnet.Escape_On_Break` switch to `True` or `False`

# Seminar Outline

Rational Networking—TCP/IP

Telnet
- Introduction
- How to Form and Terminate a Telnet Session
- How to Resume a Telnet Session
- How to Manage Multiple Sessions
- Use of Telnet Defaults
- Use of Telnet

FTP

RPC

# Exercise: Forming a Telnet Session

1. Log into the R1000.
2. Form a Telnet session, supplying only the `Remote_Machine` parameter. Use the network name of your machine.
3. Log into the remote machine as your remote user (`">>Remote Username<<"`).
4. Traverse to the `!Users` library.
5. Switch back to the R1000, leaving the Telnet session intact.
6. Resume the same Telnet session. Note that you are reconnected to the same context.
7. Switch back to the R1000 again without disconnecting the Telnet session.

# Exercise: Forming Multiple Telnet Sessions

1. Form a second Telnet session with the same machine by specifying the same machine name and `Session => 2`.
2. Log in as you did before but with a different session.
3. Switch back to the R1000 without disconnecting the Telnet session.
4. You can now quickly resume either Telnet session, enabling simultaneous work in multiple contexts. Try it.
5. Terminate both Telnet sessions.

# Exercise: Setting Telnet Defaults

1. Bring up your session switches for editing.
2. Set the appropriate switch to the name of your machine.
3. Select a new key to use for switching from remote to local context, and assign it to the corresponding switch.
4. Repeat the exercise called "Forming a Telnet Session," using minimal typing and using your new key for switching from remote to local context.
5. Terminate all the sessions you formed.

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

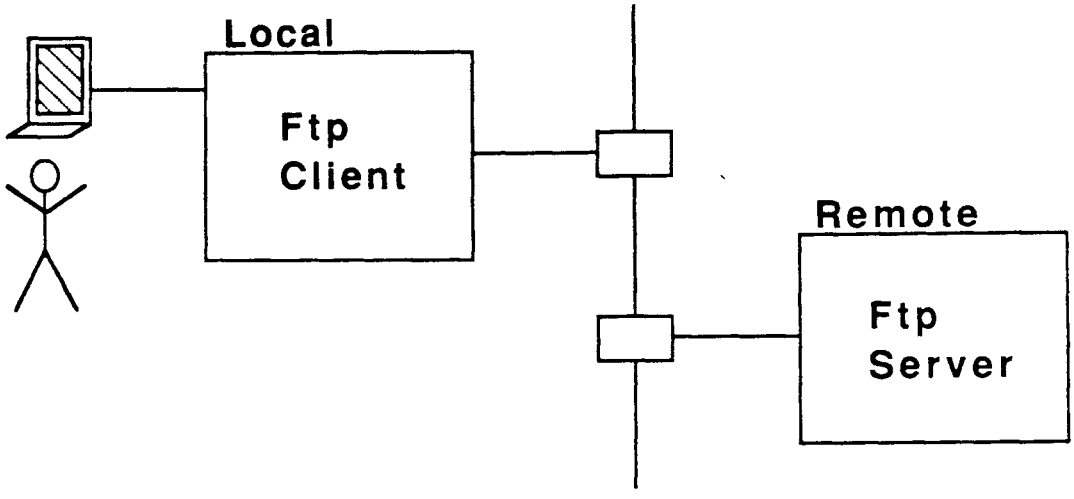
- Introduction
  How to Form and Terminate an FTP Session
  How to Transfer Single Files
  Transfer Types
  Mapping of Names
  Naming Contexts
  Use of FTP Defaults
  How to Transfer Multiple Files
  Additional FTP Operations

RPC

# FTP

- Enables transferring text files between R1000s and other computers
- Enables transferring binary data files between R1000s and other computers
- Offers operations for manipulating files on a remote machine
- Has a command and programmatic interface
- Is fully compatible with the DoD standard

# FTP Sessions




- A client must form an FTP session with a server and log in before transferring files

# FTP Sessions, cont.

- When an FTP session is formed, the client can
  - — Transfer files to the server
  - — Request receipt of files from the server
  - — Query the status of the FTP session
  - — Manipulate files on the remote machine via the server
- When communication is complete, the FTP session should be terminated

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Basic Model

- Form an FTP session by connecting to a remote machine and logging in
  - — Gain access to a remote context without leaving the local context
  - — Can now transfer files and issue other FTP commands
- Terminate the session when done using the FTP connection
  - — Form a new session to perform further FTP operations

# Basic Mechanisms

- Form an FTP session:

```
Ftp.Connect (
   To_Machine => ">>Host Name<<",
   Auto_Login => True,
   Username => ">>Remote Username<<",
   Password => ">>Remote Password<<");
```

  - — Specify the machine name
  - — Set `Auto_Login => True` and supply the remote `Username` and `Password` for login to occur as part of this command

- Terminate an FTP session: `Ftp.Disconnect;`

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Use of Basic Commands

- Form an FTP session and log in:

```
Ftp.Connect
```

- Transfer a file from local to remote:

```
Ftp.Store (
  From_Local_File => ">>Filename<<",
  To_Remote_File => ">>Filename<<",
  Append_To_File => False);
```

- Transfer a file from remote to local:

```
Ftp.Retrieve (
  From_Remote_File => ">>Filename<<",
  To_Local_File => ">>Filename<<",
  Append_To_File => False);
```

- Terminate an FTP session: `Ftp.Disconnect;`

# Macro FTP Commands

- Allow you to issue a single command that will perform the following operations:
  - — Form an FTP session
  - — Log in
  - — Transfer file(s)
  - — Log out
  - — Terminate an FTP session
- Can save time and typing
- Eliminate potential resource conflict for connections

# Macro File Transfers

- To transfer a file from local to remote:

```
Ftp.Put (
  From_Local_File => ">>Filename<<",
  To_Remote_File => ">>Filename<<",
  Remote_Machine => ">>Host Name<<",
  Username => ">>Remote Username<<",
  Password => ">>Remote Password<<");
```

# Macro File Transfers, cont.

- To transfer a file from remote to local:

```
Ftp.Get (
   From_Remote_File => ">>Filename<<",
   To_Local_File => ">>Filename<<",
   Remote_Machine => ">>Host Name<<",
   Username => ">>Remote Username<<",
   Password => ">>Remote Password<<");
```

- With these commands, you don't need to issue `Ftp.Connect` and `Ftp.Disconnect` commands separately

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Transfer Types

- FTP uses one of the following file representations whenever a file is transferred
  - — Ascii: text files
  - — Image: byte-aligned
  - — Binary: bit-aligned (R1000 to R1000 only)
  - — Others
- These are called *transfer types*
- Selecting an appropriate transfer type yields a more efficient transfer

# Ascii (Text) Transfer Type

- This type is used for transferring text files between machines
- It can also be used for transferring Ada units between R1000s and to other systems
- The file looks the same under the editor on either machine
- In making the transfer, FTP
  - — Understands the file format on both machines
  - — Converts the text to FTP transfer format
  - — Sends the text
  - — Reconverts the text to remote file format

# Image Transfer Type

- This type is used for transferring binary data between machines
- It is also a more efficient means of transferring text files from R1000 to R1000
- The type is called *byte-aligned* transfer mode
- It transfers a raw stream of bytes
- Examples
  - — Program output
  - — Program input

# Binary Transfer Type

- This type should be used only for transfer from R1000 to R1000
- It is used to transfer binary data with a specific bit length that must be retained
- The type is called *non-byte-aligned* transfer
- Bit length of data is the same at both ends
- In doing the transfer, FTP
  - — Pads the final byte before sending
  - — Strips the same amount of padding upon receipt
- Examples
  - — Polymorphic I/O files
  - — Keyboard macro files
  - — Source_Archive data files

# Binary Transfer Type, cont.

- This represents an addition to FTP beyond the standard

# Ways to Set the Transfer Type

- The default transfer type is Ascii
- Some ways to specify a different transfer type include

```
— Ftp.Connect (
        Transfer_Type => ... );

— Ftp.Put (
        Transfer_Type => ... );

— Ftp.Get (
        Transfer_Type => ... );

— Ftp.Use_Type (
        Value => ... );
```

# Transfer of R1000 Objects with Source_Archive

- Note that this is the best means for transfer from R1000 to R1000
- Use for transferring Ada units, whole libraries, subsystem views, and so on
- Use the command: `Source_Archive.Transfer`
- Note that the object transferred is identical on the other end
- Note that the command is built on top of TCP/IP and RPC
- Use only for small- to medium-sized collections of items; use tape for large collections of items to improve performance

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Basics of Name Mapping

- FTP can automatically generate legal filenames for files transferred to a remote machine
- Remote machine types whose naming conventions are currently supported include
  - — Rational
  - — UNIX
  - — AOS
  - — VMS
- Name differences among machines primarily pertain to punctuation
- Examples

```
My_Program'Ada

My_Program_Ada

My_Program.Ada
```

# Selection of Automatic Name Mapping

- Specify the desired remote type (Rational, UNIX, AOS, VMS):

```
— Ftp.Connect (
        Remote_Type => ... );

— Ftp.Put (
        Remote_Type => ... );

— Ftp.Get (
        Remote_Type => ... );

— Ftp.Use_Remote_Type (
        Value => ... );
```

- Note that the default remote type is `Rational`

# Selection of Automatic Name Mapping, cont.

- Set the destination file parameter to the FTP transfer command to be " "

```
— Ftp.Store (
        To_Remote_File => "",... );

— Ftp.Put (
        To_Remote_File => "",... );

— Ftp.Retrieve (
        To_Local_File => "",... );

— Ftp.Get (
        To_Local_File => "",...);
```

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

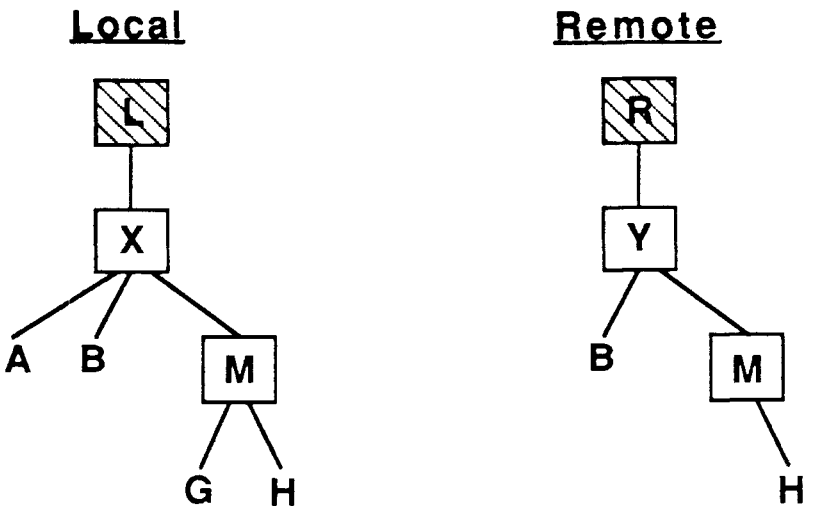
# Local and Remote Contexts

- Names of source and destination files are resolved relative to the current local and remote contexts
- The current local context is the context of your Command window when you issue an FTP command
- Some ways to establish the remote context include
  - The context you log into when forming an FTP session
  - `Ftp.Connect ( Remote_Directory => ... );`
  - `Ftp.Put ( Remote_Directory => ... );`
  - `Ftp.Get ( Remote_Directory => ... );`

# Local and Remote Contexts, cont.

```
— Ftp.Change_Working_Directory (
      Remote_Directory => ...);
```

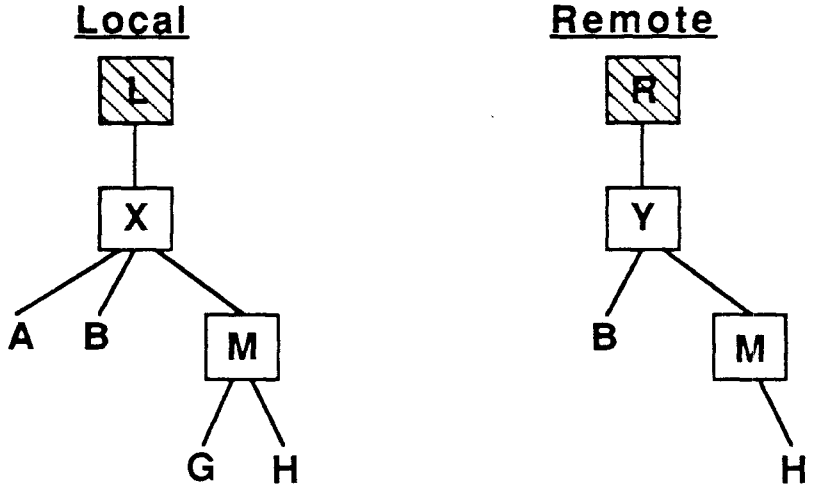
# Naming Example 1




- Local context: `L`
- Remote context: `R`
- Goal: Transfer file `A` into remote directory `Y`
- Solution:

```
Ftp.Store (
   From_Local_File => "X.A",
   To_Remote_File => "Y.A" );
```

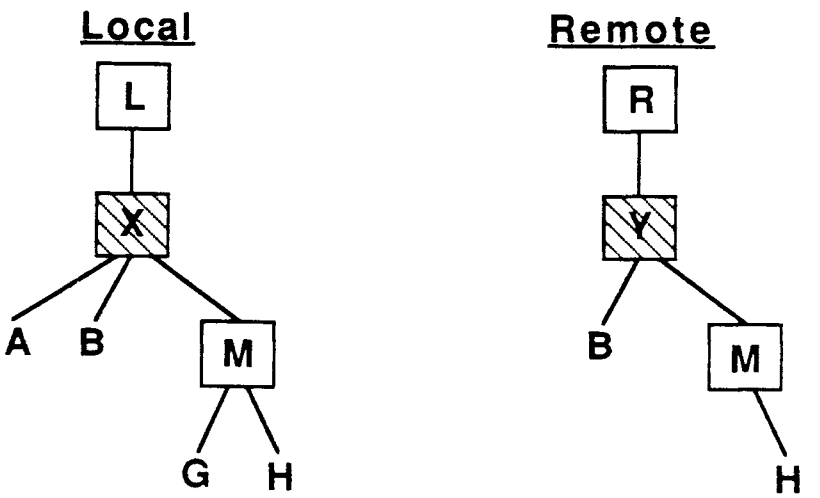
# Naming Example 2




- Local context: `L`
- Remote context: `R`
- Goal: Transfer file `G` into remote directory `M`
- Solution:

```
Ftp.Store (
  From_Local_File => "X.M.G",
  To_Remote_File => "" );
```

# Naming Example 3




- Local context: `X`
- Remote context: `Y`
- Goal: Transfer file `G` into remote directory `M`
- Solution:

```
Ftp.Store (
  From_Local_File => "M.G",
  To_Remote_File => "" );
```

# Exercise: Using Local to Remote FTP Transfer

1. Go to the R1000 `Test_Software` library.
2. Use `Ftp.Put` to transfer the text file `Sample_Input` to the remote. Set the `To_Remote_File` parameter to be `Sample_Input`. Note that this macro command connects, logs in, transfers, and then disconnects the FTP session.
3. Telnet to the remote. Note that the file has been transferred.
4. Return to the R1000, leaving your Telnet session intact.
5. Now form an FTP session between the R1000 and the remote, so that you can perform multiple FTP operations without connecting and disconnecting each time.

# Exercise: Using Local to Remote FTP Transfer, cont.

6. Using `Ftp.Store`, transfer `Display_Complex_Sums'Body` to the remote. This time default the `To_Remote_File` parameter to get automatic name generation.

7. Telnet to the remote to view the transferred unit. Note the name that has been assigned.

8. Return to the R1000, leaving your Telnet session intact.

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Basics of FTP Defaults

- Many FTP commands have default parameter values that can be assigned indirectly
- Indirectly assignable default FTP parameters begin with the prefix `Ftp_Profile`
- Initial default values are resolved as follows
  - — The switch file associated with the enclosing library for the current context is checked first
  - — The user session switch file is checked next
- To alter a default value with the `Ftp_Profile` prefix, set the corresponding session or library switch

# Examples of Default Parameters

- `Ftp_Profile.Auto_Login`
- `Ftp_Profile.Username`
- `Ftp_Profile.Password`
- `Ftp_Profile.Remote_Machine`
- `Ftp_Profile.Transfer_Type`
- `Ftp_Profile.Remote_Directory`
- `Ftp_Profile.Remote_Type`

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Terminology

- FTP commands are available for transferring multiple files at once
- Each command for transferring multiple files expects either a *set* or a *list* of filenames
- A *set* of files is specified by a name that resolves to many filenames
  - — The names are resolved on the machine from which the files are taken
- A *list* of files is specified using a text file stored on the local machine
  - — The list contains one full pathname per line
  - — The list contains no comments or extra white space
  - — The names are resolved on the machine from which the files are taken

# Multiple File Transfer Commands

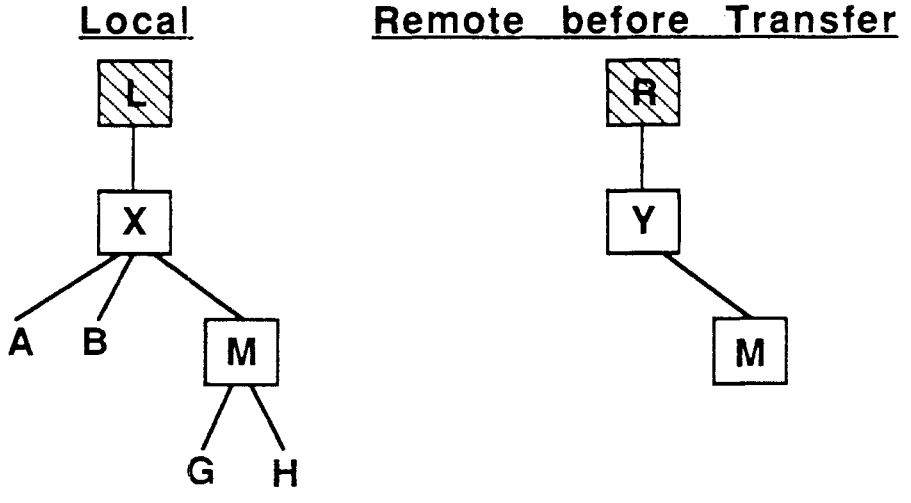
- Commands expecting a *set* of files
  - `Ftp.Store_Set`
  - `Ftp.Retrieve_Set`
- Macro commands expecting a *set* of files
  - `Ftp.Put_Set`
  - `Ftp.Get_Set`
- Commands expecting a *list* of files
  - `Ftp.Retrieve_List`
- Macro commands expecting a *list* of files
  - `Ftp.Get_List`
- Some FTP servers implement only a subset of these commands

# Local and Remote Roofs

- Multiple file transfer commands have parameters called `Local_Roof` and `Remote_Roof`
- A *roof* is an ancestor directory of a group of files being transferred
- Specifying a local and a remote roof allows the transfer of a set of nested subdirectories, preserving their hierarchical organization

# Isomorphic Transfer

- A transfer that preserves hierarchical structure is called an *isomorphic transfer*
- Example

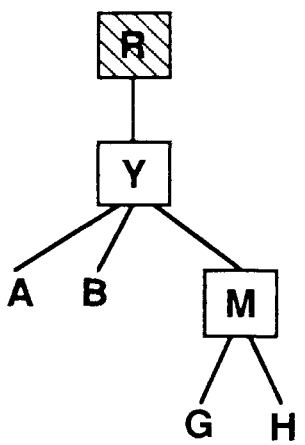


# Isomorphic Transfer, cont.

- Local context: `L`
- Remote context: `R`
- Goal: Transfer all files in `X` to a remote directory `Y`, preserving their hierarchical structure
- Solution:

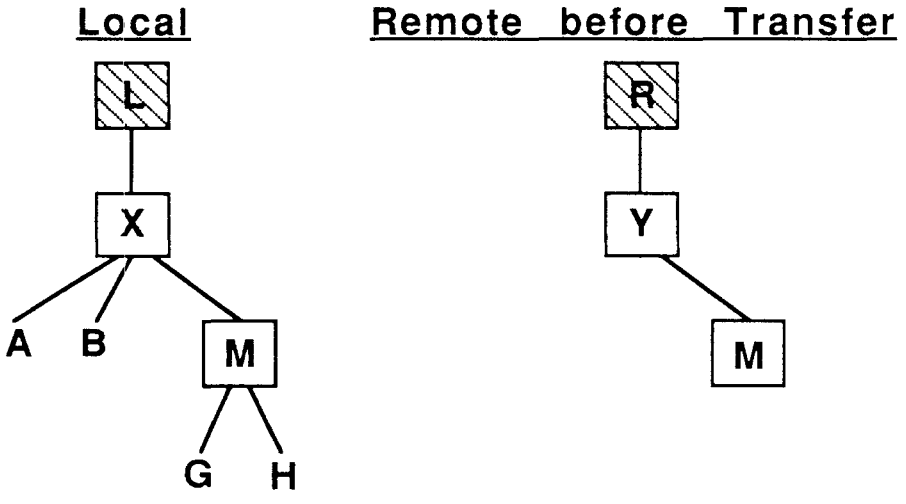
```
Ftp.Store_Set (
        From_Local_File_Set => "X.?",
        Local_Roof => "X",
        Remote_Roof => "Y");
```

Remote after Transfer



# Flat Transfers

- Transferring files scattered within a hierarchical structure into a single directory is called a *flat* transfer
- To accomplish a flat transfer, specify the source roof as "".
- Example

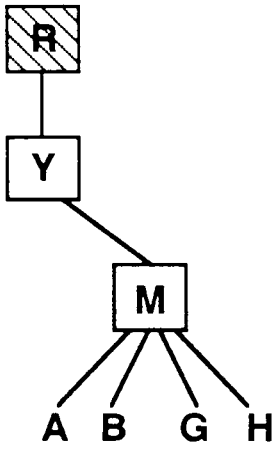


# Flat Transfers, cont.

- Local context: `L`
- Remote context: `R`
- Goal: Transfer all files in `X` to remote directory `M`
- Solution:

```
Ftp.Store_Set (
        From_Local_File_Set => "X.?",
        Local_Roof => "",
        Remote_Roof => "Y.M");
```

Remote after Transfer



# Rules for Multiple File Transfers

- A source file is transferred isomorphically if and only if the subdirectories containing it already exist as subdirectories of the destination roof
- If a source roof is not an ancestor directory of any file being transferred, those will be flattened out under the destination roof

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

- Introduction
- How to Form and Terminate an FTP Session
- How to Transfer Single Files
- Transfer Types
- Mapping of Names
- Naming Contexts
- Use of FTP Defaults
- How to Transfer Multiple Files
- Additional FTP Operations

RPC

# Viewing Remote Directory Listings

- A remote directory listing can be obtained and stored in a local file using FTP

```
Ftp.List (
   Remote_Pathname =>
     ">>Remote Pathname<<",
   Verbose => False);
```

# How to Obtain FTP Status Information

- Display status of all FTP sessions formed with local machine

  — `Ftp.Status_All`

- Display status of the current FTP session

  — `Ftp.Status`

- Display status from the remote machine

  — `Ftp.Remote_Status`

- Display current user session switch values for all FTP default parameters

  — `Ftp.Show_Profile`

# Exercise: Using Remote to Local FTP Transfer

1. Create a world within the R1000 `Test_Software` library called `Ftp_Example`.
2. Using FTP, from the R1000 change the working directory on the remote to be the `Remote_Software` library.
3. Using `Ftp.Store_Set`, transfer `List_Generic` and `List_Generic'Body` to the remote. Note that you can use the name `List_Generic@` to indicate both these units. Use automatic remote name generation.
4. Telnet to the remote and then back, noting that the files have been transferred to the remote `Remote_Software` library and seeing the names that were generated.
5. Go to the `Ftp_Example` world on the R1000.

# Exercise: Using Remote to Local FTP Transfer, cont.

6. From there, use `Ftp.Retrieve` to transfer back the `List_Generic` spec from the remote to the R1000. Transfer it to an R1000 file called `List_Generic_File`.
7. Now use `Ftp.Retrieve` to transfer the `List_Generic` body from the remote to the R1000, appending it to the same R1000 `List_Generic_File`.
8. Execute `Compilation.Parse`, specifying `List_Generic_File` as the `File_Name`. Execute this command, and note that the R1000 creates appropriate Ada units for those found in the `List_Generic_File`.
9. Now try performing the same movement on the `List_Generic` spec and body using the `Source_Archive.Transfer` command.
10. Terminate your FTP session.

# Exercise: Using Remote to Local FTP Transfer, cont.

11. Terminate your Telnet session.

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

RPC

- Introduction
  Terminology
  Use of RPC

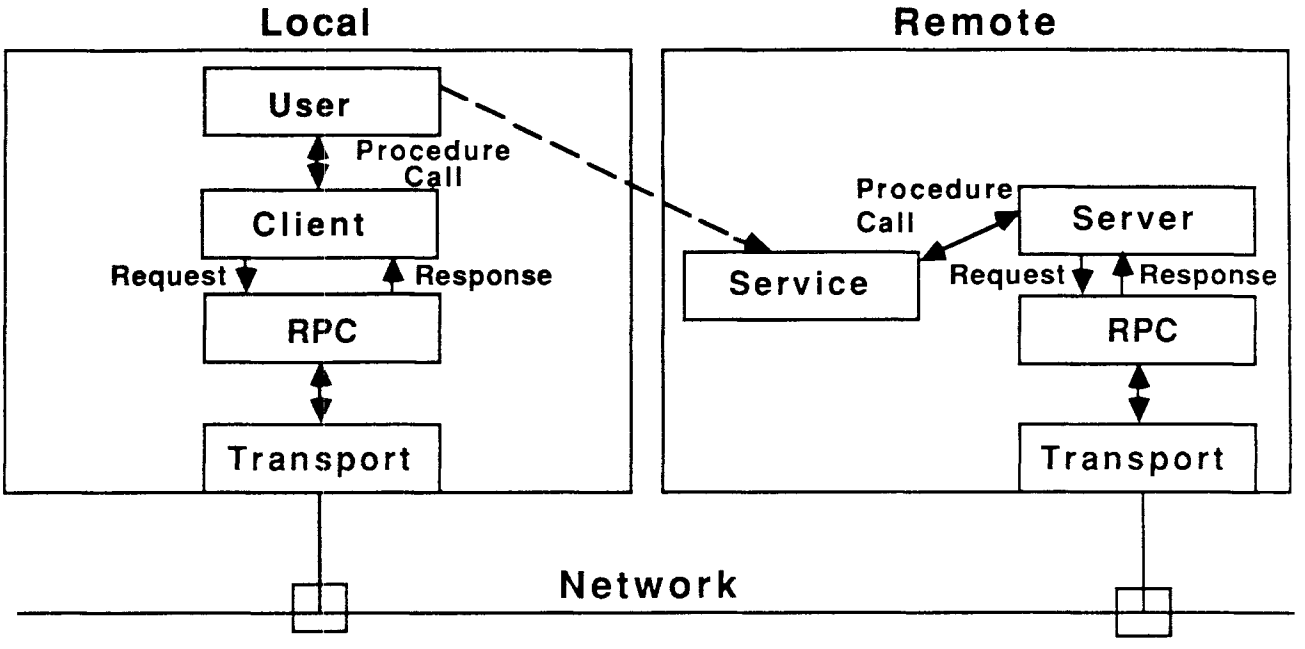
# Rational Remote Procedure Call

- Makes interfaces on one machine programmatically available on another
- Allows a program on one system to call the facilities exported by programs on another
- Extends the semantics of procedure and function calls across multiple machines
- Allows the values of all Ada types (except access and task types) to be passed as parameters or as function results
- Enables exceptions to be propagated back to the caller

# Some RPC Applications

- Building R1000 functional test scaffolds
  - — Interface to a database on a remote system
  - — Interface to target-dependent software modules that reside on a target
- Integrating tools running on other hosts with the Rational Environment
- Building a host/target debugger

# RPC Model

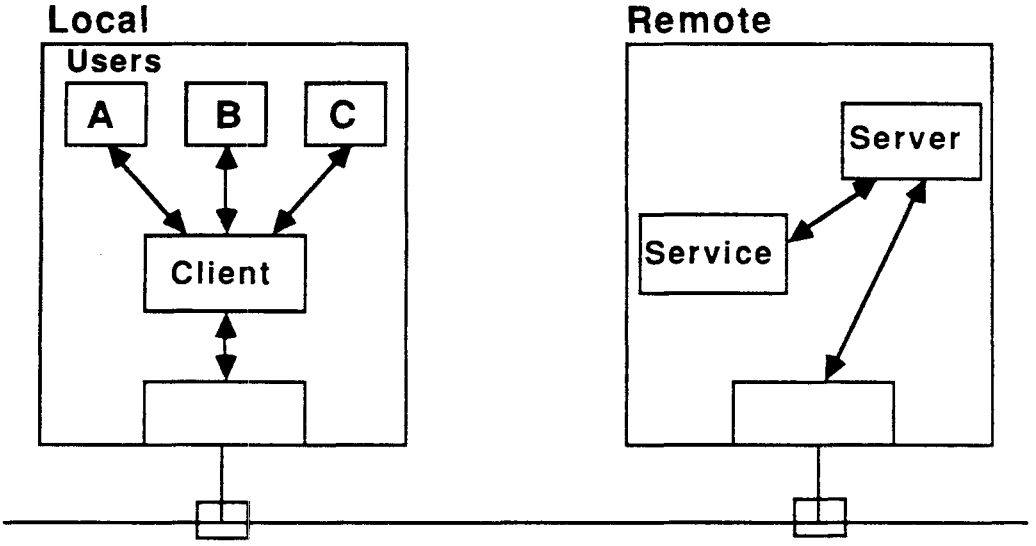

Local
Remote
User
Procedure Call
Client
Request
Response
RPC
Transport
Procedure Call
Server
Service
Request
Response
RPC
Transport
Network

# Basic Model

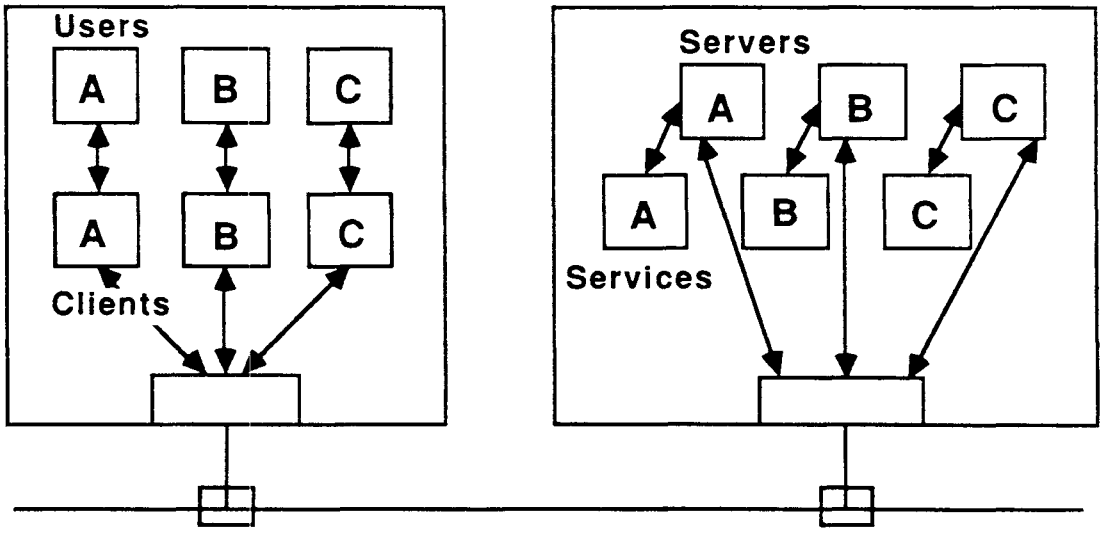
- When a remote procedure is called by a local program
  - — The client converts a call to an appropriate interchange format
  - — The request goes over the network
  - — The server and remote RPC receive the request and convert it from the interchange format into a format understood by the remote
  - — The server calls the required interfaces of the service
  - — The server passes the response back from the service to the client by reversing the above steps
  - — The client receives the response, converts it, and passes the results to the calling routine

# Service Options

- Shared service




- Separate service



# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

RPC

- Introduction
- Terminology
- Use of RPC

# Remote Facilities

- Service
  - — Is a collection of procedures resident on a remote machine that are to be called from programs on a local machine
  - — Has a specification that must be expressable in Ada
  - — Has an implementation that need not be implemented in Ada
- Server
  - — Is a program resident on the remote machine that is associated with a particular service
  - — Enables the local machine to access the facilities of the service
  - — Can be built from a Rational-supplied template

# Remote Facilities, cont.

— Could be coded in any language

# Local Facility

- Client
  - — Is an Ada interface resident on a local machine that enables RPC calls to a particular remote service
  - — Has Ada specifications for the service, allowing local units to compile against them
  - — Is built from a Rational-supplied template

# Common Facility

- <Service Name>_Defs
  - — Is a common package resident on both local and remote machines
  - — Is invoked by both the client and the server
  - — Provides a common format that allows client and server to exchange data
  - — Enumerates service subprograms for identification across machines
  - — Specifies an interchange format that provides a way of representing Ada types so that they can be transported in a machine-independent way

# Seminar Outline

Rational Networking—TCP/IP

Telnet

FTP

RPC
- Introduction
- Terminology
- Use of RPC

# Use of RPC

- Build a client and a server
  - — Identify remote interfaces that the local machine intends to access
  - — Collect these interfaces into a remote service
  - — Create: <Service Name>_Defs, server, and client
- Elaborate the server
- Execute main programs that use the client to invoke the remote service

# Server Elaboration

- This is done before any client invokes it
- It can be done as part of the machine boot
- Several server tasks are begun upon elaboration; others begin if needed by a client

# Client Elaboration

- This is done during elaboration of the closure of the main programs that call the client
- The client has a dynamic pool of tasks so that multiple threads of the main program can use the client simultaneously
- Several client tasks are started upon elaboration; others begin if needed
- The `Finalize` command can be inserted in any thread of the program; if encountered, this causes all client tasks to terminate so that the client and the main program can terminate

# Exercise: Using RPC

1. Go to the R1000 `Rpc_Client` library and from there to the `Client` subdirectory. This is where an RPC client spec and body reside. Bring up a window containing the client spec and expand it for a fuller view.

   This client provides a local Ada interface to the subprograms declared in a remote service. This spec is identical to the spec of the remote service you'll be calling.

   Note that there is an exception declared in the client also. This, too, is identical to the one defined in the remote service and is exported to the local machine by the RPC client and server.

# Exercise: Using RPC, cont.

2. Return to the R1000 `Rpc_Client` library and from there go to the `User` subdirectory. This is where a main program that makes a remote procedure call resides. Bring up a window containing the main program body and find the statement that makes the remote procedure call. Note that from the user perspective, it is transparent that a remote interface is being accessed.

3. Telnet to the remote. Go to the remote `Rpc_Server` library and list it. Note that the directory contains an RPC server as well as a program for elaborating the server.

4. Elaborate the RPC server by running the program that elaborates the server. Now the remote is set up to respond properly to a remote procedure call from the local.

# Exercise: Using RPC, cont.

5. Leaving the server running and the Telnet session intact, return to the R1000.
6. Run the user program with valid input. You have just executed a remote procedure.
7. Execute the program again, this time with invalid input (e.g., -1). Note the remote exception propagation.
8. Return to the remote and kill the server. Terminate the Telnet session.
9. (Optional) Try adding another subprogram to the client and server.

   For further information about clients and servers, read the chapter on "Rational Remote Procedure Call Facility" in the "Introduction to Networking" section of the *Rational Networking—TCP/IP* manual.