SENG 401
Software Architecture

Lecture 7
Topic – Documenting Software Architecture
Sub-Topic – Component and Connector Styles

Gias Uddin
Assistant Professor, University of Calgary

http://giasuddin.ca/

SCHULICH
School of Engineering

Books Used/Cited in this Course

ID	Title
B1	Software Architecture: Foundations, Theory, and Practice. Taylor, Medvidovic, Dashofy. Wiley, 2009. *Available at UofC Library*
B2	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume I. http://www.aosabook.org/en/index.html
B3	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume II. http://www.aosabook.org/en/index.html
B4	**Documenting Software Architecture: Views and Beyond, 2nd Edition. Addison-Wesley, 2010. Clements et al.** ***Available at UofC Library***
B5	Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Nick Rozanski. Addison-Wesley Professions, 2011. *Available at UofC Library*
B6	Software Architecture in Practice. Bass, Clements, Kazman. Addison-Wesley, 2012 *Available at UofC Library*

SCHULICH
School of Engineering

Documenting Software Architecture
Topic: Collection of Software Architectural Styles
Sub-Topic: Component and Connector Styles

References: B4 – C4

C&C Styles

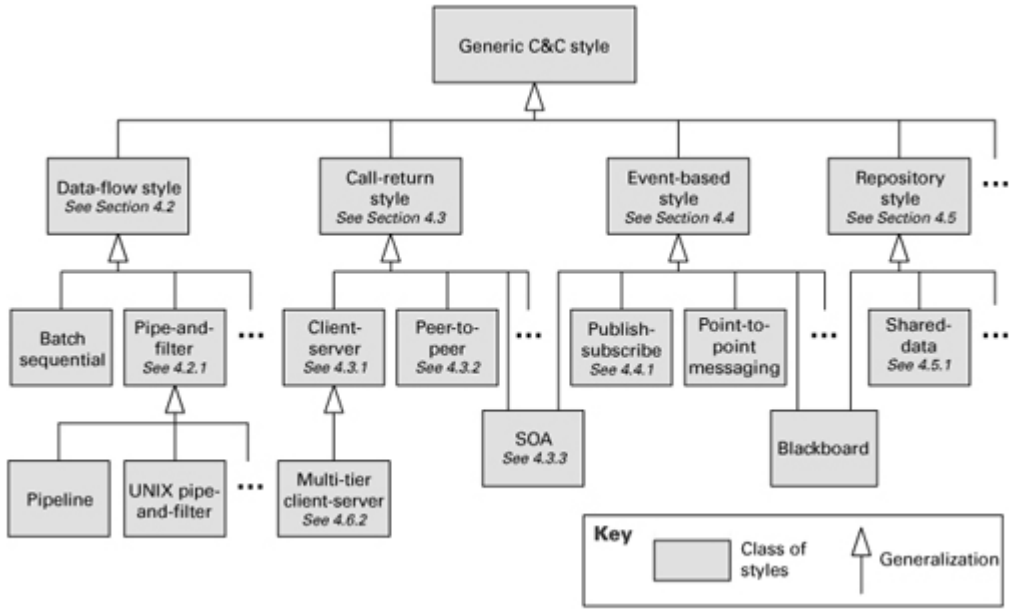
4 Major Types

A component-and-connector (C&C) style introduces a specific set of component-and-connector types and specifies rules about how elements of those types can be combined. Additionally, given that C&C views capture runtime aspects of a system, a C&C style is typically also associated with a computational model that prescribes how data and control flow through systems designed in that style.

Style Name	What is it?
1. Data flow styles	Styles in which computation is driven by the flow of data through the system.
2. Call-return styles	Styles in which components interact through synchronous invocation of capabilities provided by other components.
3. Event-based styles	Styles in which components interact through asynchronous events or messages.
4. Repository styles	Styles in which components interact through large collections of persistent, shared data.

C&C Styles

A partial representation of the space of C&C styles



Naturally this is only a partial representation of the space of C&C styles: there are other general categories, and there are many styles that are specializations of these categories. Additionally, in most real systems several styles may be used together, often from across categories. For example, enterprise IT applications are frequently a combination of client-server and shared-data styles.

1. Data Flow Styles

Overview

Data flow styles embody a computational model in which components act as data transformers and connectors transmit data from the outputs of one component to the inputs of another. Each component type in a data flow style has some number of input ports and output ports. Its job is to consume data on its input ports and write transformed data to its output ports.	
Batch Sequential Data Flow Style	In the early days of computing, one common data flow style was “batch sequential,” a style in which each component transforms all of its data before the next component can consume its outputs.
Pipe-and-Filter Data Flow Style	Later a form of data flow style was invented in which components run concurrently and data is incrementally processed: the pipe-and-filter style. Today data flow styles are common in domains where stream processing occurs, and where the overall computation can be broken down into a set of transformational steps.

Pipe and Filter Style

The pattern of interaction in the pipe-and-filter style is characterized by successive transformations of streams of data. Data arrives at a filter's input ports, is transformed, and then is passed via its output ports through a pipe to the next filter. A single filter can consume from, or produce data to, multiple ports. Modern examples of such systems are signal-processing systems, systems built using UNIX pipes, the request-processing architecture of the Apache Web server, the map-reduce paradigm for search engines, Yahoo! Pipes for processing RSS feeds, and many scientific computation systems that have to process and analyze large streams of experimental data.

Elements	• **Filter**, which is a component that transforms data read on its input ports to data written on its output ports. Filers typically execute concurrently and incrementally. Properties may specify processing rates, input/output data formats, and the transformations executed by the filter. A filter transforms data that it receives through one or more pipes and transmits the result through one or more pipes. Filters typically execute concurrently and incrementally. • **Pipe**, which is a connector that conveys data from a filter's output ports to another filter's input ports. A pipe has a single data-in and a single data-out role, preserves the sequence of data items, and does not alter the data passing through. Properties may specify buffer size, protocol of interaction, and data format that passes through a pipe. A pipe is a connector that conveys streams of data from the output port of one filter to the input port of another filter. Pipes act as unidirectional conduits, providing an order-preserving, buffered communication channel to transmit data generated by filters. In the pure pipe-and-filter style, filters interact only through pipes.

Pipe and Filter Style

Relations	The attachment relation associates filter output ports with data-in roles of a pipe, and filter input ports with data-out roles of pipes
Computational Model	Data is transformed from a system's external inputs to its external outputs through a series of transformations performed by its filters
Constraints	• Pipes connect filter output ports to filter input ports • Connected filters must agree on the type of data being passed along the connecting pipe • Specializations of the style may restrict the association of components to an acyclic group or a linear sequence – sometimes called a pipeline • Other specializations may prescribe that components have certain named ports, such as stdin, stdout, and stderr ports of UNIX filters
What it's for	• Improving reuse due to the independence filters • Improving throughput with parallelization of data processing • Simplifying reasoning about overall behavior
Relation to Other Styles and Models	A pipe-and-filter view of a system is not the same as a data flow model. In the pipe-and-filter style, lines between components represent connectors, which have a specific computational meaning: They transmit streams of data from one filter to another. In data flow models, the lines represent relations, indicating the communication of data between components. Flows in a data flow model have little computational meaning: They simply indicate that data flows from one element to the next. This flow might be realized by a connector, such as a procedure call, the routing of an event between a publisher and a subscriber, or data transmitted via a pipe. The reason that these views might be confused is that the data flow model of a pipe-and-filter style looks almost identical to the original pipe-and-filter view. Data flow styles are often combined with other styles by using them to characterize a particular subsystem. A good example of this is the filter processing chains of the Apache Web server.

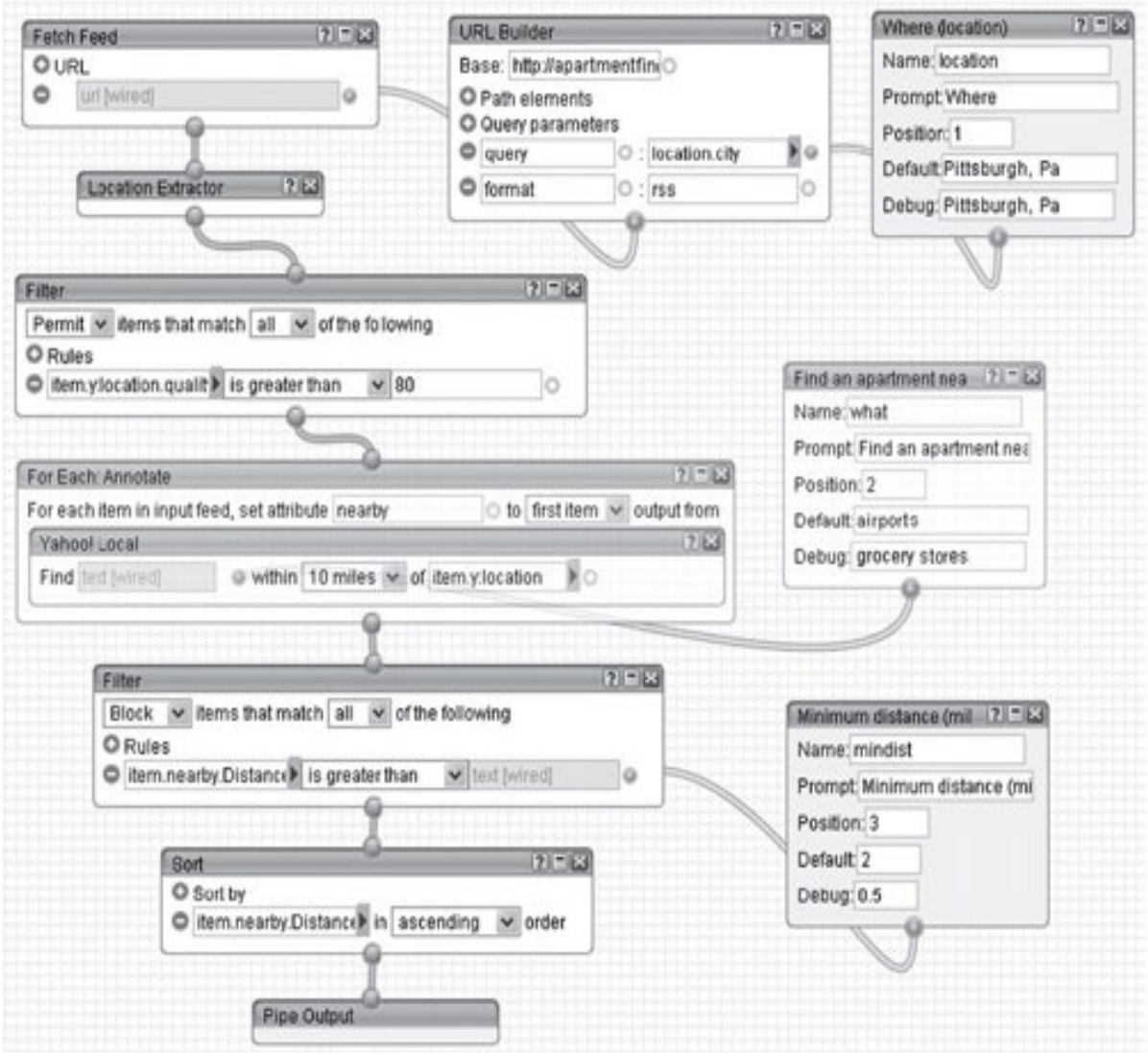
Pipe and Filter Style
Example Yahoo! Pipes

- “Rewire the Web” is the motto of Yahoo! Pipes, a composition tool that lets Web users combine simple functions quickly and easily into pipe-and-filter applications that aggregate and manipulate content from around the Web.
- The basis of Yahoo! Pipes is the many RSS feeds available from sites on the Internet. These data streams form the input to the applications that users build, applications that combine and manipulate the data in the streams to form useful results. Many of the building blocks to perform general-purpose filtering and manipulation of the data streams are made available in the composition environment itself, rather like library functions.
- For example, you can take an RSS stream from a financial news site and filter it so that only news items related to stocks that you own are shown. Or you can take an RSS stream from a sports site and filter it so that you see news about your favorite teams or athletes.
- Yahoo! Pipes uses terminology not quite the same as that in this book. It calls a complete application a pipe; the building blocks are called modules. A filter is a special kind of module that removes values from a stream based on given comparison criteria.

Pipe and Filter Style

Example Yahoo! Pipes

A Yahoo! Pipes application for finding apartments for rent near a given location (shown using the notation of the Yahoo! Pipes editor). The pipe-and-filter flow runs from top to bottom through the seven "modules" down the left-hand side (each representing what our pipe-and-filter style calls a filter); this is indicated by the thick solid lines (the pipes) connecting the output port of one to the input port of the next. The other "modules" supply inputs to the mainline components; this is indicated by the thinner, hollow lines. The Fetch Feed component uses the RSS output from an apartment-finder search; it is fed the search site URL and the search parameters by the helper modules to its right. The Location Extractor and the Filter component extract high-quality (well-formed) addresses from the apartment-finder search. That stream feeds Yahoo! Local, which finds businesses of a given type (supplied by its helper module) near a given location. (The For Each component applies the function shown in its interior to every item in the input stream.) The second Filter removes listings that aren't a minimum distance from our search term. The Sort component orders the stream in ascending order of distance for viewing via the Pipe Output component.



2. Call-Return Styles

- Call-return styles embody a computational model in which components provide a set of services that may be invoked by other components.1 A component invoking a service pauses (or is blocked) until that service has completed. Hence, call-return is the architectural analog of a procedure call in programming languages. The connectors are responsible for conveying the service request from the requester to the provider and for returning any results.
- Call-return styles differ among each other in a variety of ways. Some variants differ in terms of the behavior of their connectors. For example, connectors in some call-return styles may support error handling (such as when the service provider is not available). Other differences relate to constraints on topology. Some call-return architectures are organized in tiers. Others partition the set of components into disjoint sets of components that can make requests and those that can service them.
- Examples of call-return styles include client-server, peer-to-peer, and representational state transfer (REST) styles.

2. Call-Return Styles
Client-Server Style

As with all call-return styles, client-server style components interact by requesting services of other components. Requesters are termed clients, and service providers are termed servers, which provide a set of services through one or more of their ports. Some components may act as both clients and servers. There may be one central server or multiple distributed ones. Typical examples of systems in the client-server style include the following:• Information systems running on local networks, where the clients are GUI applications (such as Visual Basic) and the server is a database management system (such as Oracle)• Web-based applications where the clients run on Web browsers and the servers are components running on a Web server (such as Tomcat)

Elements	• Client, which is a component that invokes services of a server component • Server, which is a component that provides services to client components. Properties will vary according to concerns of the architect but typically include information about the nature of the server ports (such as how many clients can connect) and performance characteristics (such as maximum rates of service invocation). • Request/reply connector, which is used by a client to invoke services on a server. Request/reply connectors have two roles: a request role and a reply role. Connector properties may include whether the calls are local or remote and whether data is encrypted.
Relations	The attachment relation associates client service-request ports with the request role of the connector and server service-reply ports with the reply-role of the connectors

2. Call-Return Styles

Client-Server Style

Computational Model	Clients instantiate interactions, invoking services as needed from servers and waiting for the results of those requests.

Constraints	• Clients are connected to servers through request/reply connectors.
	• Server components can be clients to other servers
	• Specializations may impose restrictions:
	  • Numbers of attachments to a given port
	  • Allowed relations among servers
	• Components may be arranged in tiers

What it's for	• Promoting maintainability and reuse by factoring out common services • Improving scalability and availability in case server replication is in place • Analyzing dependability, security, and throughput
Relation to other styles	• Like many C&C styles, the client-server style decouples producers of services and data from consumers of those services and data. Other styles, such as peer-to-peer, involve a round-trip form of communication. However, these styles do not have the asymmetric relationship between clients and servers found in the client-server style. • Clients and servers are often grouped and deployed on different machines in a distributed environment to form a multitier hierarchy.

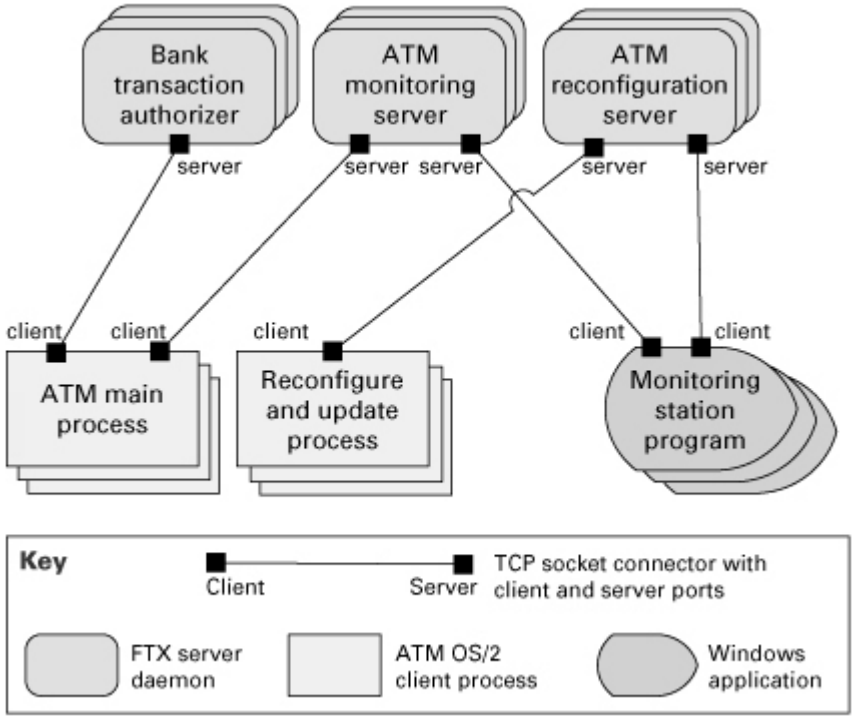
2. Call-Return Styles

Real world applications using Client-Server Style

Application	Example
World Wide Web (WWW)	It is a hypertext-based system that allows clients (Web browsers) to access information from servers distributed across the Internet. Clients access the information, written in Hypertext Markup Language (HTML), provided by a Web server using Hypertext Transfer Protocol (HTTP). HTTP is a form of request/reply invocation. HTTP is a stateless protocol; the connection between the client and the server is terminated after each response from the server.
An ATM Banking system	There are three components: • The FTX server daemons are processes running in the background on the fault-tolerant UNIX (FTX) server. Each daemon creates one or more socket ports using predefined TCP ports, through which calls from client components arrive. • ATM OS/2 client processes are concurrent processes that run on the ATMs, which were powered with the IBM OS/2 operating system. Although it can't be inferred from the diagram, each ATM runs one instance of the ATM main process and one instance of the Reconfigure and update process. • The Windows application component was also a client component. It was a Windows 3.x GUI program developed using the Borland OWL API. Each instance was used by an operator to monitor a group of ATMs from his or her workstation. When this system was developed, the TCP socket connector was very often used for communication in client-server and distributed applications. Today HTTP is far more common. In the protocol used in this TCP socket connector, the client opens a connection to a server identified by an IP address and port number. The client then sends a nonblocking request, after which it may display in the UI a "Please wait..." message to the user. Then the client calls an operation to receive the response from the server. For both client and server, sending and receiving messages are separate steps. Therefore, the connector implementation has to handle the correlation of request and response messages, as well as time-outs and communication errors.

2. Call-Return Styles

Real world applications using Client-Server Style



Client-server architecture of an ATM banking system. The ATM main process sends requests to Bank transaction authorizer corresponding to user operations (such as deposit, withdrawal). It also sends messages to ATM monitoring server informing the overall status of the ATM (devices, sensors, and supplies). The Reconfigure and update process component sends requests to ATM reconfiguration server to find out if a reconfiguration command was issued for that particular ATM. Reconfiguration of an ATM (for example, enabling or disabling a menu option) and data updates are issued by bank personnel using the Monitoring station program. Monitoring station program also sends periodic requests to ATM monitoring server to retrieve the status of the range of ATMs monitored by that station.

2. Call-Return Styles
Peer-to-Peer Styles

In the peer-to-peer style, components directly interact as peers by exchanging services. Peer-to-peer communication is a kind of request/reply interaction without the asymmetry found in the client-server style. That is, any component can, in principle, interact with any other component by requesting its services. Each peer component provides and consumes similar services, and sometimes all peers are instances of the same component type. Connectors in peer-to-peer systems may involve complex bidirectional protocols of interaction, reflecting the two-way communication that may exist between two or more peer-to-peer components. Examples of peer-to-peer systems include file-sharing networks, such as BitTorrent and eDonkey; instant messaging and VoIP applications, such as Skype; and desktop grid computing systems.

Elements	• Peer component, which are typically independent programs running on network nodes. Each peer component acts as both client and server. Peers have interfaces that describe the services they request from other peers and the services they provide. The computational flow of peer-to-peer systems is symmetric: Peers first connect to the peer-to-peer network and then initiate actions to achieve their computation by cooperating with their peers by requesting services from one another. • Call-return connector, which is used to connect to the peer network, search for other peers, and invoke services from other peers. Often a peer's search for another peer is propagated from one peer to its connected peers for a limited number of hops. A peer-to-peer architecture may have special peer nodes (called ultrapeers, ultranodes, or supernodes) that have indexing or routing capability and allow a regular peer's search to reach a larger number of peers. Constraints on the use of the peer-to-peer style might limit the number of peers that can be connected to a given peer or impose a restriction about which peers know about which other peers.

2. Call-Return Styles

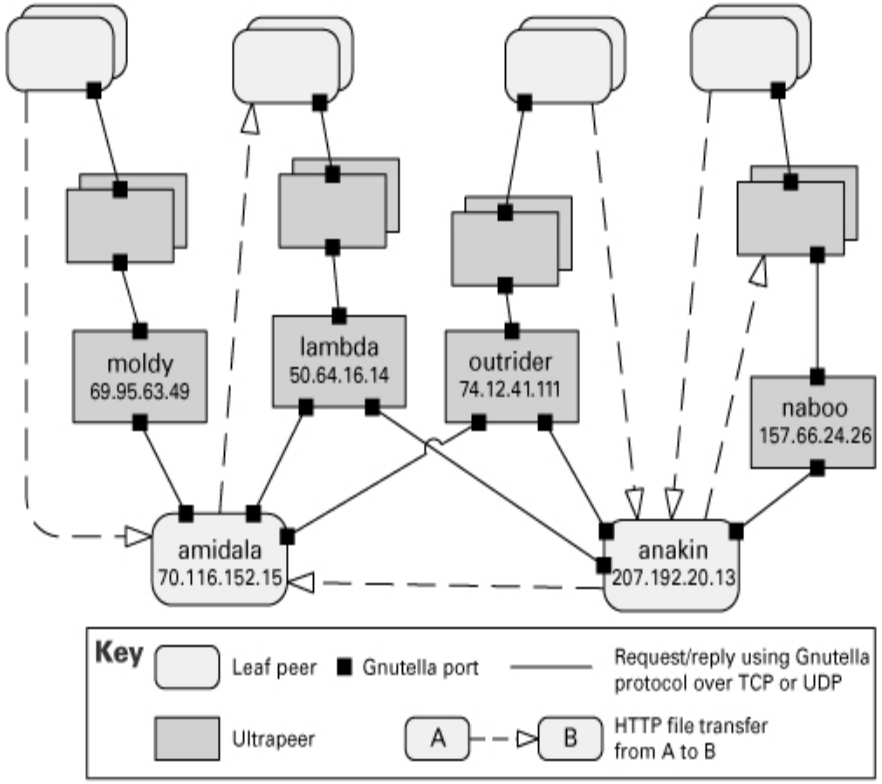
Peer-to-Peer Styles

Relations	The attachment relation associates peers with call-return connectors
Computational Model	Computation is achieved by cooperating peers that request services of one another.
Properties	Same as other C&C views, with an emphasis on protocols of interaction and performance-oriented properties. Attachments may change at runtime.
Constraints	• Restrictions may be placed on the number of allowable attachments to any given port or role. • Special peer components can provide routing, indexing, and peer search capability. • Specialization may impose visibility restrictions on which components can know about other components.
What it's for	• Providing enhanced availability • Providing enhanced scalability • Enabling highly distributed systems, such as file sharing, instant messaging, and desktop grid computing.
Relation to other styles	The absence of hierarchy means that peer-to-peer systems have a more general topology than client-server systems.

2. Call-Return Styles

Examples of Peer-to-Peer Styles

- Gnutella is a peer-to-peer network that supports bidirectional file transfers. The topology of the system changes at runtime as peer components connect and disconnect to the network. A peer component is a running copy of a Gnutella client program connected to the Internet. Upon startup, this program establishes a connection with a few other peers. The Web addresses of these peers are kept in a local cache.
- The Gnutella protocol supports request/reply messages for peers to connect to other peers and search for files. Peers are identified by their IP address, and the Gnutella protocol messages are carried over dedicated UDP and TCP ports. To perform a search, a Gnutella peer requests information from all of its connected peers, which respond with any information of interest. The connected peers also pass the request to their peers successively, up to a predefined number of "hops." All the peers that have positive results for the search request reply directly to the requester, whose IP address and port number go along with the request. The requester then establishes a connection directly with the peers that have the desired file and initiates the data transfer using HTTP (outside the Gnutella network).
- Later versions of Gnutella differentiate between leaf peers and ultrapeers. An ultrapeer runs on a computer with a fast Internet connection. A leaf peer is usually connected to a small number (say, three) of ultrapeers, and an ultrapeer is connected to a large number of other ultrapeers and leaf peers. The ultrapeers are responsible for routing search requests and responses for all leaf peers connected to them.



Acknowledgment

Lecture contents and pictures are taken from B4 Chapter 4