System Models

527950-1
Fall 2019
9/26/2019
Kyoung Shin Park
Applied Computer Engineering
Dankook University

Chapter 2. System Models

From Coulouris, Dollimore, Kindberg and Blair
Distributed Systems: Concepts and Design
Edition 5, © Addison-Wesley 2012

Overview

- **Physical Models**
- **Architectural Models**
  1. **Architectural Elements**
     - A. Communicating entities : Process, Object, Component, Web Service
     - B. Communication paradigms : IPC, Remote Invocation, Indirect Communication
     - C. Roles and responsibilities : Client-Server, Peer-to-peer
     - D. Placement : Multiple servers, Caching, Mobile Code, Mobile Agents
  2. **Architectural Patterns**
     - A. Layering
     - B. Tiered-Architecture : Two-tier, Three-tier Architectures
     - C. Thin Clients
     - D. Other patterns : proxy, brokerage
  3. **Middleware Platforms**
- **Fundamental Models**
  - Interaction Model
  - Failure Model
  - Security Model

Physical Models

- Definition
  - The **hardware composition** of a system in terms of the computers (and other devices) and their interconnecting networks
  - A **representation of the underlying hardware elements** of a distributed system that abstracts away from specific details of the computer and networking technologies employed
- Three generations of distributed systems
  - **Early distributed systems**; in response to LAN technologies
  - **Internet-scale distributed systems**; in response to Internet
  - **Contemporary distributed systems**
    - The emergence of mobile computing
    - The emergence of ubiquitous computing
    - The emergence of cloud computing

Architectural Models

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Figure 2-1. Generations of Distributed Systems

Architectural Models

- Definition
  - The architecture of a system is its structure in terms of separately specified components and their interrelationships
- Goal
  - Ensure that the structure will meet present and likely future demands on it
- Major Concerns
  - Make the system reliable, manageable, adaptable and cost-effective

Architectural Models

- Three-stage approach :
  1. Looking at the core underlying **architectural elements**
  2. Examining composite **architectural patterns**
  3. Considering middleware platforms that are available to support the various styles of programming that emerge from the above architectural styles

1. Architectural Elements

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Figure 2-2. Communicating Entities and Communication Paradigms

1. Architectural Elements

- Consider **four key questions** to understand the fundamental building blocks of a distributed system
  - A. **Communicating entities** - What are the entities that are communicating in the distributed system?
  - B. **Communication paradigms** - How do they communicate, or, more specifically, what communication paradigm is used?
  - C. **Roles and responsibilities** - What (potentially changing) roles and responsibilities do they have in the overall architecture?
  - D. **Placement** - How are they mapped on to the physical distributed infrastructure (what is their placement)?

1.A. Communicating Entities

1. **Processes**
   - The prevailing view of a distributed system
   - Coupled with appropriate inter-process communication paradigms
   - Caveats
     - In some primitive environments, such as sensor networks, the underlying operating systems may not support process abstractions, and hence the entities in such systems are nodes
     - In most distributed system environments, processes are supplemented by threads
2. **Objects**
   - Consists of a number of interacting objects representing natural units of decomposition for the given problem domain
   - Accessed via interfaces
   - An associated interface definition language (or **IDL**) providing a specification of the methods defined on an object

1.A. Communicating Entities

3. **Components**
   - Resemble objects in that they offer problem-oriented abstractions and are also accessed through interfaces
   - Components specify not only their interfaces but also the assumptions they make in terms of other components/interfaces that must be present for a component to fulfil its function
   - Make all dependencies explicit and provide a more complete contract for system construction
4. **Web Services**
   - Approach based on encapsulation of behavior and access through interfaces
   - Web services are intrinsically integrated into the World Wide Web, using web standards to represent and discover services

Distributed Objects and Components

- **Programming abstractions**

- **Middleware solutions**
  - Provide a higher-level programming abstractions for the distributed systems and, through layering, to abstract over heterogeneity in the underlying infrastructure to promote interoperability and portability

Distributed Objects and Components

- **Distributed object middleware**
  - Adopt an object-oriented programming model
  - Communicating entities are represented by objects
  - Objects communicate mainly using remote method invocation(RMI)
  - Encapsulation and data abstraction
  - **Java RMI** and **CORBA**

Distributed Objects and Components

```
struct Rectangle{                                   1
    long width;
    long height;
    long x;
    long y;
} ;

struct GraphicalObject {                            2
    string type;
    Rectangle enclosing;
    boolean isFilled;
};

interface Shape {                                   3
    long getVersion() ;
    GraphicalObject getAllState() ;        // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;                  4
interface ShapeList {                               5
    exception FullException{ };                     6
    Shape newShape(in GraphicalObject g) raises (FullException);   7
    All allShapes(); // returns sequence of remote object references   8
    long getVersion() ;
};
```

Figure 8-2. IDL Interfaces: Shape and ShapeList

Distributed Objects and Components

```
module Whiteboard {
    struct Rectangle{
    ...} ;
    struct GraphicalObject {
    ...};
    interface Shape {
    ...};
    typedef sequence <Shape, 100> All;
    interface ShapeList {
    ...};
};
```

Figure 8-3. IDL Model : Whiteboard

Distributed Objects and Components

Type	*Examples*	*Use*
sequence	*typedef sequence <Shape, 100> All;* *typedef sequence <Shape> All* bounded and unbounded sequences of *Shapes*	Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified.
string	*String name;* *typedef string<8> SmallString;* unbounded and bounded sequences of characters	Defines a sequences of characters, terminated by the null character. An upper bound on the length may be specified.
array	*typedef octet uniqueId[12];* *typedef GraphicalObject GO[10][8]*	Defines a type for a multi-dimensional fixed-length sequence of elements of a specified IDL type.

this figure continues on the next slide

Figure 8-4. IDL Constructed Types

Distributed Objects and Components

Type	*Examples*	*Use*
record	*struct GraphicalObject { string type; Rectangle enclosing; boolean isFilled; };*	Defines a type for a record containing a group of related entities. *Structs* are passed by value in arguments and results.
enumerated	*enum Rand (Exp, Number, Name);*	The enumerated type in IDL maps a type name onto a small set of integer values.
union	*union Exp switch (Rand) { case Exp: string vote; case Number: long n; case Name: string s; };*	The IDL discriminated union allows one of a given set of types to be passed as an argument. The header is parameterized by an *enum*, which specifies which member is in use.

Figure 8-4. IDL Constructed Types

Distributed Objects and Components

- **Component-based middleware**
  - To overcome limitations from distributed **object** middleware
  - **(Implicit dependency)** : Object interfaces do not describe what the implementation of an object depends on, making object-based systems difficult to develop (especially for third-party developers) and subsequently manage
  - **(Programming complexity)** : Programming distributed object middleware leads to a need to master a many low-level details associated with middleware implementation
  - **(Lack of separation of distribution concerns)** : Application developers are obliged to consider details of concerns such as security, failure handling and concurrency, which are largely similar from one application to another
  - **(No support for deployment)** : Object-based middleware provides little or no support for the deployment of configurations of objects
  - **Enterprise JavaBeans** and **Fractal**

Web Services

- Provides a **service interface** enabling clients to interact with servers in a more general way than web browsers do.
- Clients access the operations in the interface of a web service by means of requests and replies formatted in XML and usually transmitted over HTTP.
- Identified by a URI(Uniform Resource Identifier)
  - A string of characters designed for unambiguous identification of resources
  - The most common form of URI is the Uniform Resource Locator (URL), frequently referred to informally as a web address.
- Easily used in Internet-wide applications

Web Services

- **SOAP (Simple Object Access Protocol)**
  - A messaging protocol specification for exchanging structured information in the implementation of web services in computer networks
  - Its purpose is to induce extensibility, neutrality and independence.
  - Uses **XML** Information Set for its message format, and
  - Relies on application layer protocols, most often **Hypertext Transfer Protocol (HTTP)** or **Simple Mail Transfer Protocol (SMTP)**, for message negotiation and transmission.

Web Services

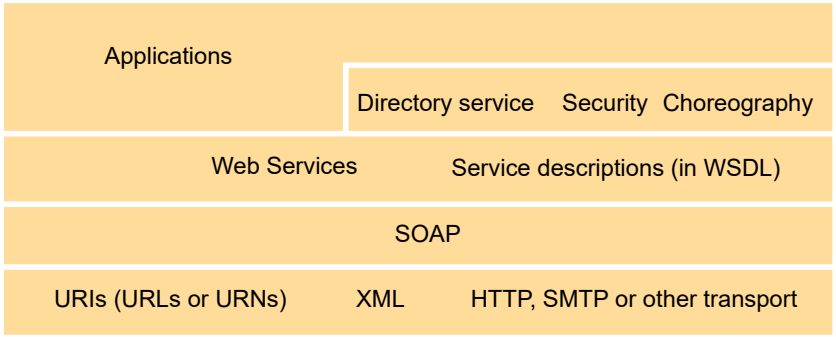


Figure 9-1. Web services infrastructure and components

1.B. Communication Paradigms

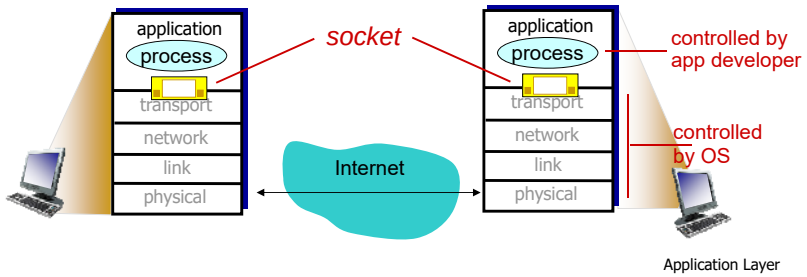
1. **Inter-Process Communication (IPC)**
2. **Remote Invocation**
3. **Indirect Communication**

Inter-Process Communication

- Relatively low-level support for communication between processes, including message-passing primitives, direct access to the API offered by Internet protocols (socket programming)
- **Process**: program running within a host
  - Within same host, two processes communicate using inter-process communication (IPC) (defined by OS)
  - Processes in different hosts communicate by exchanging messages
- Client process
  - Process that initiates communication
- Server process
  - Process that waits to be contacted

Socket Programing

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Remote Invocation

The most common communication paradigm

1. **Request-reply protocols**
   - A pattern imposed on an underlying message-passing service to support client-server computing
   - Rather primitive and only really used in embedded systems, also used in the HTTP protocol
2. **Remote procedure calls (RPC)**
   - Calls as if they are procedures in the local address space
   - Hides important aspects of distribution
   - Offers (at a minimum) access and location transparency
3. **Remote method invocation (RMI)**
   - Resembles RPCs but in a world of distributed objects

Remote Invocation

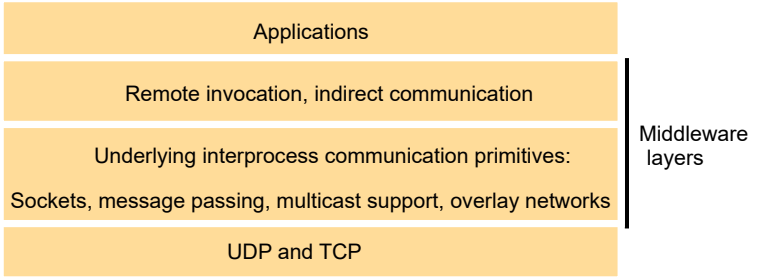


Figure 5-1. Middleware Layers

Remote Invocation

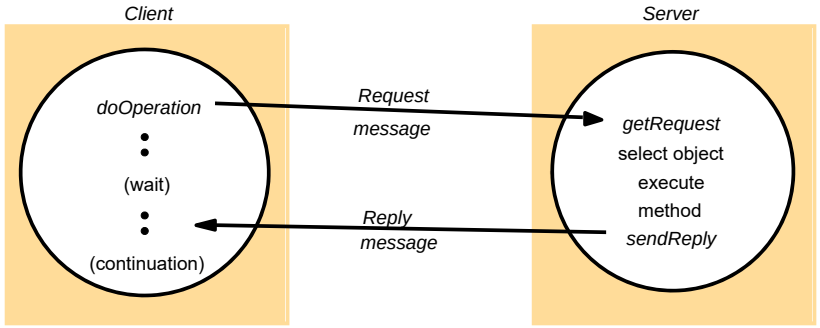


Figure 5-2. Request-Reply Communication

Indirect Communication

- **Through a third entity**, allowing a strong degree of decoupling between senders and receivers
  - **Space uncoupling** - senders do not need to know who they are sending to
  - **Time uncoupling** - senders and receivers do not need to exist at the same time

Indirect Communication

	Time-coupled	*Time-uncoupled*
Space coupling	*Properties*: Communication directed towards a given receiver or receivers; receiver(s) must exist at that moment in time *Examples*: Message passing, remote invocation (see Chapters 4 and 5)	*Properties*: Communication directed towards a given receiver or receivers; sender(s) and receiver(s) can have independent lifetimes *Examples*: See Exercise 15.3
Space uncoupling	*Properties*: Sender does not need to know the identity of the receiver(s); receiver(s) must exist at that moment in time *Examples*: IP multicast (see Chapter 4)	*Properties*: Sender does not need to know the identity of the receiver(s); sender(s) and receiver(s) can have independent lifetimes *Examples*: Most indirect communication paradigms covered in this chapter

Figure 6-1. Space and Time Coupling in Distributed Systems

Indirect Communication

1. **Group Communication**
   - Multiparty communication paradigm supporting one-to-many communication
   - Senders send messages to the group via the **group identifier**, and hence do not need to know the recipients of the message
2. **Publish-Subscribe Systems**
   - **Information-dissemination systems**
   - A large number of producers (or publishers) distribute information items of **interest** (**events**, **topic** in Kafka) to a similarly large number of consumers (or subscribers)
   - Offer a **one-to-many** style of communication

Indirect Communication

3. **Message Queues**
   - Offer a point-to-point service
   - Queues offer an indirection between the **producer and consumer processes**
4. **Tuple Spaces**
   - Processes can place arbitrary items of structured data, called tuples, in a persistent tuple space
   - Other processes can either read or remove such tuples from the tuple space by specifying patterns of interest
5. **Distributed Shared Memory(DSM)**
   - Provide an abstraction for sharing data between processes that do not share physical memory.
   - The underlying infrastructure must ensure a copy is provided in a timely manner and also deal with issues relating to synchronization and consistency of data

Publish-Subscribe Paradigm

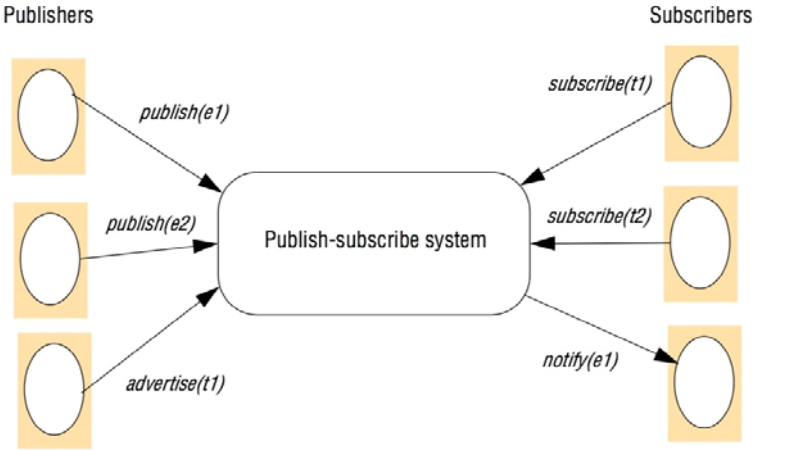


Figure 6-8. The Publish-Subscribe Paradigm

Publish-Subscribe Architecture

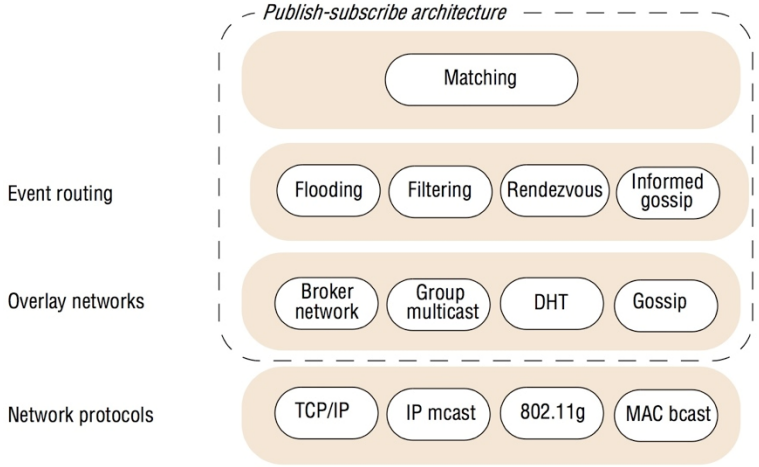


Figure 6-10. The Architecture of Publish-Subscribe Systems

Broker Network

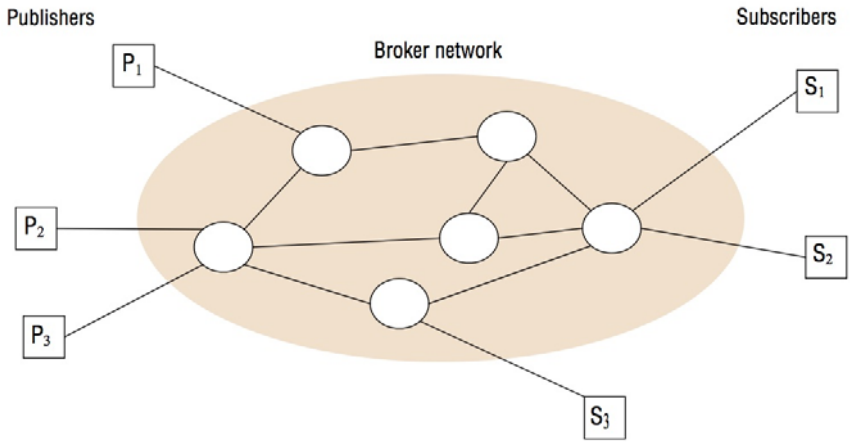


Figure 6-9. A Network of Brokers

Message Queue Paradigm

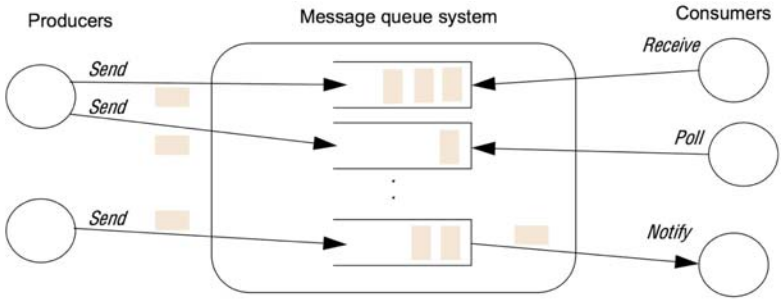


Figure 6-14. The Message Queue Paradigm

Distributed Shared Memory

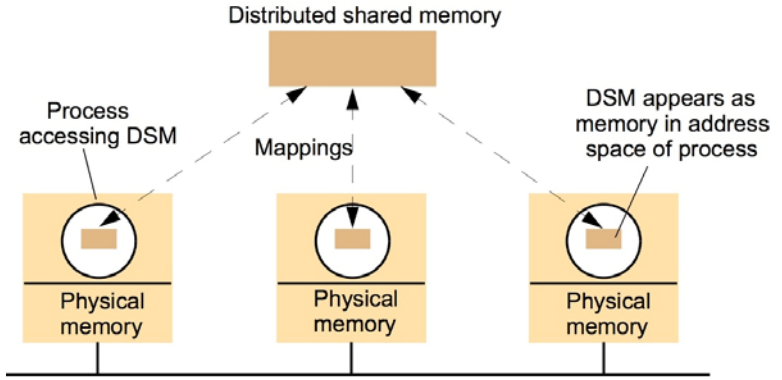


Figure 6-19. The Distributed Shared Memory (DSM) Abstraction

1.C. Roles and Responsibilities

- Two architectural styles stemming from the role of individual processes

1. **Client-Server architectural style**
2. **Peer-to-Peer architectural style**

Client-Server Architecture

- **Client-Server Architectural Style**
  - Client processes interact with individual server processes in potentially separate host computers in order to access the shared resources that they manage
  - Servers may in turn be clients of other servers
  - Scales poorly

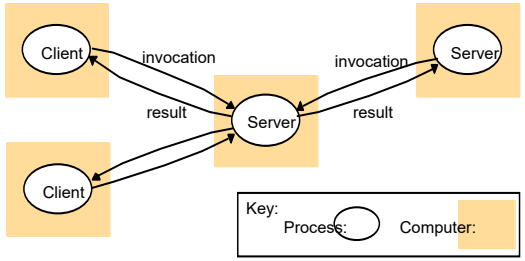


Figure 2.3. Clients Invoke Individual Servers

Peer-to-Peer Architecture

- **Peer-to-Peer Architectural Style**
  - All of the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes
  - All participating processes run the same program and offer the same set of interfaces to each other
  - Recent and widely used instance is the BitTorrent file-sharing system

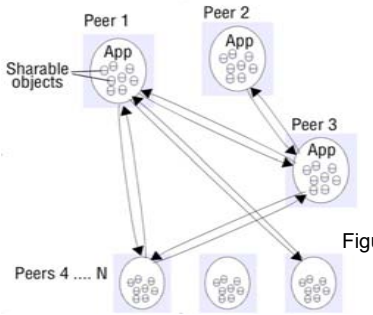


Figure 2.4a. Peer-to-Peer Architecture

1.D. Placement

- How entities such as objects or services map on to the underlying physical distributed infrastructure
- A matter of careful design issue
- Placement **needs to take into account**
  - The patterns of communication between entities
  - The reliability of given machines
  - Their current loading
  - The quality of communication between different machines
- Placement must be determined with strong application knowledge
  - There are few universal guidelines to obtaining an optimal solution

1.D. Placement

1. **Multiple servers**
2. **Caching**
3. **Mobile code**
4. **Mobile agents**

Multiple Servers

- Services may be implemented as several server processes in separate host computers
- The servers may partition the set of objects on which the service is based and distribute those objects between themselves, or
- They may maintain replicated copies of them on several hosts

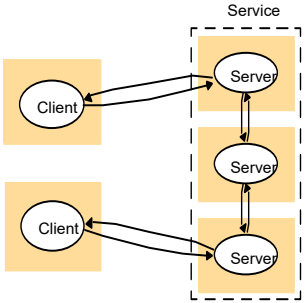


Figure 2.4b. A Service Provided by Multiple Servers

Caching

- Store recently used data objects that is closer to one client or a particular set of clients than the objects themselves
- May be co-located with each client
  - Web browsers maintain a cache of recently visited web pages and other web resources in the client's local file system
- May be located in a proxy server that can be shared by several clients
  - The purpose of proxy servers is to increase the availability and performance of the service by reducing the load on the wide area network and web servers

Caching

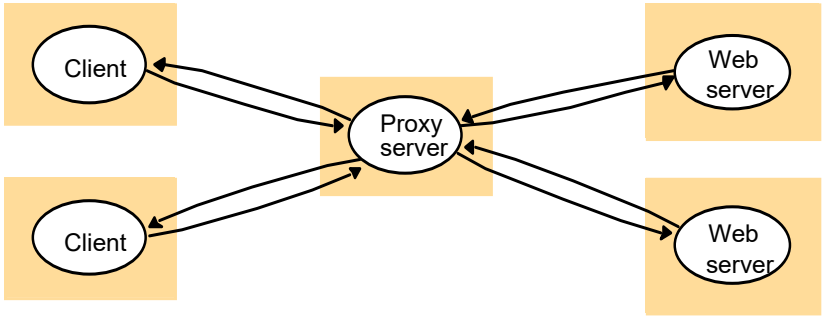


Figure 2.5. Web Proxy Server

Mobile Code

- Applets are a well-known and widely used example of mobile code
  - The user running a browser selects a link to an applet whose code is stored on a web server
  - The code is downloaded to the browser and runs there
- Give good interactive response since it does not suffer from the delays or variability of bandwidth
- Potential security threat to the local resources

Mobile Code

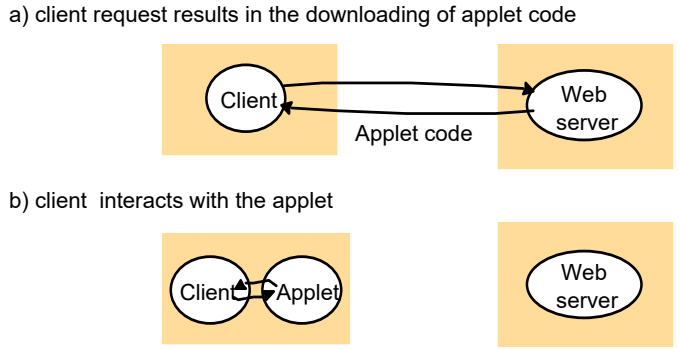


Figure 2.6. Web Applets

Mobile Agents

- A running program (including both code and data) that
- Travels from one computer to another in a network
- Carrying out a task on someone's behalf, such as collecting information, and
- Eventually returning with the results
- May make many invocations to local resources at each site it visits – for example, accessing individual database entries
- Potential security threat to the resources in computers that they visit

2. Architectural Patterns

- Build on the more primitive architectural elements
- Used in isolation or, more commonly, in combination, in developing more sophisticated distributed systems solutions
- *Not* themselves necessarily *complete solutions* but rather offer *partial insights* that, when combined with other patterns, lead the designer to a solution for a given problem domain
  - A. **Layering**
  - B. **Tiered Architecture**
  - C. **Thin Clients**
  - D. **Other commonly occurring patterns: proxy, brokerage, Reflection**

2.A. Layering

- Deals with the vertical organization of services into layers of abstraction
- A complex system is partitioned into a number of layers, with a given layer making use of the services offered by the layer below.
- A given layer therefore offers a **software abstraction**, with higher layers being unaware of implementation details, or indeed of any other layers beneath them

2.A. Layering

- **Platform**
  - Consists of the lowest-level hardware and software layers
- **Middleware**
  - A layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application

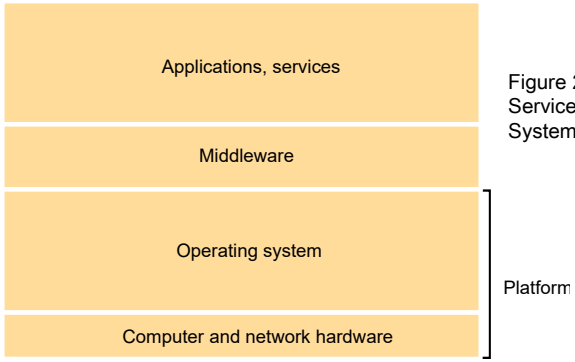
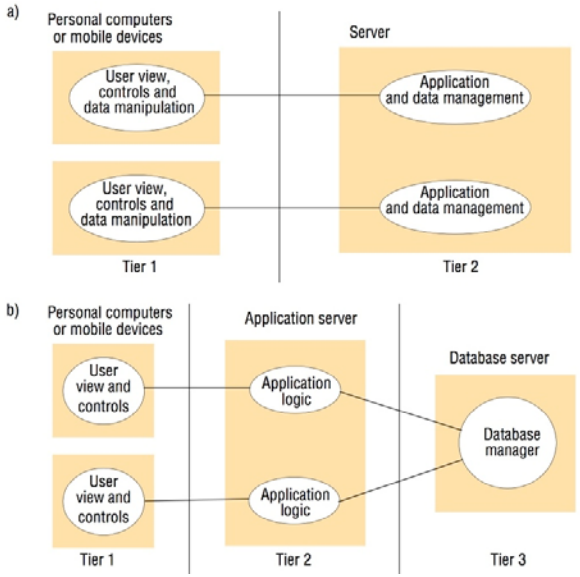


Figure 2.7 Software and Hardware Service Layers in Distributed Systems

2.B. Tiered Architecture

- **Organize functionality** of a given layer and **place this functionality into appropriate servers** and, as a secondary consideration, **on to physical nodes**
- **Functional decomposition of a given application**
  - **Presentation logic (presentation tier)**
    - Concerned with handling user interaction and updating the view of the application as presented to the user
  - **Application logic (application tier** or **business tier)**
    - Concerned with the detailed application-specific processing associated with the application
    - Also referred to as the **business logic**, although the concept is not limited only to business applications
  - **Data logic (data tier)**
    - Concerned with the persistent storage of the application, typically in a database management system

Two-Tier and Three-Tier Architectures



Two-Tier Architecture

- **Two-tier solution**
  - Three aspects must be partitioned into two processes, the client and the server
  - Splitting the application logic, with some residing in the client and the remainder in the server
  - Low latency in terms of interaction

Three-Tier Architecture

- **Three-tier solution**
  - Presentation logic tier can also be a simple user interface allowing intrinsic support for thin clients
  - Application logic tier is held in one place
    - Enhance maintainability of the software
  - Data logic tier is simply a database offering a (potentially standardized) relational service interface
  - Added complexity of managing three servers and also the added network traffic and latency
- Generalizes to **n-tiered (or multi-tier) solutions**

AJAX

- The role of AJAX (Asynchronous JavaScript And XML)
  - An extension to the standard client-server style of interaction used in the WWW
  - Meets the need for fine-grained communication between a Javascript front-end program running in a web browser and a server-based back-end program holding data describing the state of the application
  - Constitutes an effective technique for the construction of responsive web applications in the context of the indeterminate latency of the Internet

AJAX

- The role of AJAX (Asynchronous JavaScript And XML)
  - In the **standard** web style of interaction
    - A browser sends an HTTP request to a server for a page, image or other resource with a given URL.
    - The server replies by sending an entire page that is either read from a file on the server or generated by a program
    - When the resultant content is received at the client, the browser presents it according to the relevant display method for its MIME type

AJAX

- The role of AJAX (Asynchronous JavaScript And XML)
  - This standard style of interaction constrains the development of web applications
    - Once the browser has issued an HTTP request for a new web page, the user is unable to interact with the page until the new HTML content is received and presented by the browser.
    - In order to update even a small part of the current page with additional data from the server, an entire new page must be requested and displayed. This results in a delayed response to the user, additional processing at both the client and the server and redundant network traffic.
    - The contents of a page displayed at a client cannot be updated in response to changes in the application data held at the server.

AJAX

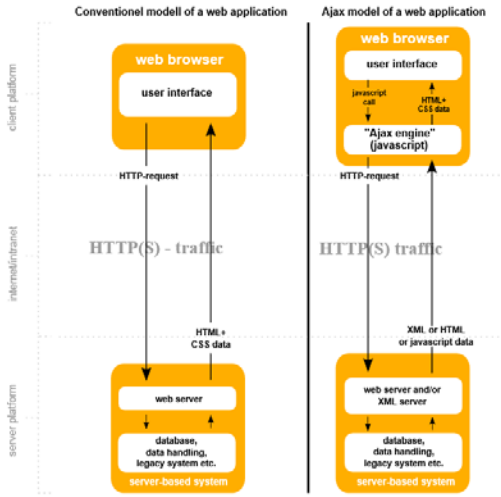
- The role of AJAX (Asynchronous JavaScript And XML)
  - AJAX is the 'glue' that supports the construction of such applications
    - It provides a communication mechanism enabling front-end components running in a browser to issue requests and receive results from back-end components running on a server.

```
new Ajax.Request('scores.php?
                  game=Arsenal:Liverpool',
                  {onSuccess: updateScore});
function updateScore(request) {
.....
      ( request contains the state of the Ajax request including the returned result.
            The result is parsed to obtain some text giving the score, which is used
            to update the relevant portion of the current page.)
.....
}
```

Figure 2.9 AJAX Example: Soccer Score Updates

What is AJAX?

- Classic model is inefficient
  - All page content disappeared then reappeared
  - Each time a page was reloaded due to a partial change, all of the content had to be re-sent, even though only some of the information had changed
  - Additional load on the server and excessive bandwidth



2.C. Thin Clients

- The trend is towards moving complexity away from the end-user device towards services in the network
  - Apparent in cloud computing and in tiered architectures
- A software layer that supports a window-based user interface that is local to the user while executing application programs or, more generally, accessing services on a remote computer

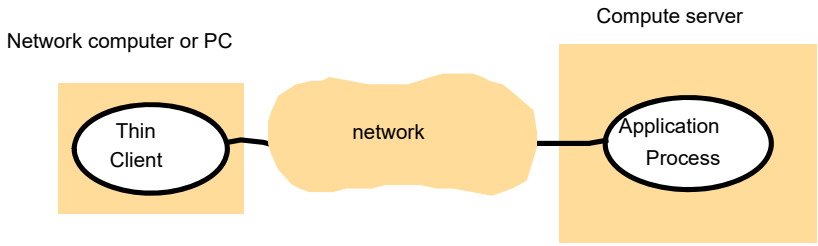


Figure 2.10 Thin Clients and Compute Servers

2.C. Thin Clients

- Advantage and Drawback
  - Potentially resource-constrained local devices can be significantly enhanced with sophisticated networked services and capabilities
  - In highly interactive graphical activities, where the delays experienced by users are increased to unacceptable levels
    - By the need to transfer image and vector information between the thin client and the application process
    - Due to both network and operating system **latencies**

2.D. Other Commonly Occurring Patterns

1. **Proxy**
2. **Broker**

Proxy

- **Proxy**
  - Designed particularly to support **location transparency** in RPC or RMI
  - A proxy is created in the **local address space** to **represent the remote object**
  - The proxy offers exactly the same interface as the remote object
  - The programmer makes calls on this proxy object and hence does not need to be aware of the distributed nature of the interaction.

Broker

- **Broker**
  - The use of brokerage in web services can usefully be viewed as an architectural pattern supporting interoperability in potentially complex distributed infrastructures
  - Consists of the trio of **service provider**, **service requester** and **service broker** (a service that matches services provided to those requested)
  - Replicated in many areas of distributed systems, for example with the registry in Java RMI and the naming service in CORBA

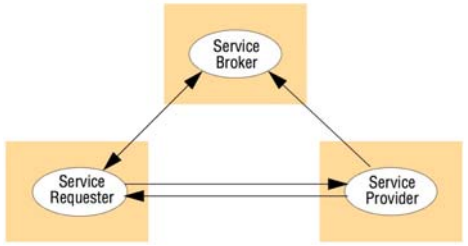
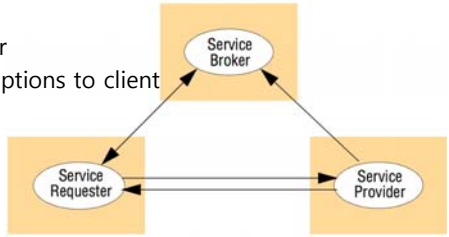


Figure 2.11
The Web Service Architectural Pattern

Broker

- **Broker**
  - Design broker component to decouple clients from servers
  - Servers:
    - Register with broker
    - Present method interfaces to clients
  - Clients
    - Access server's methods via broker
    - Uses same form to call server's methods
  - Brokers
    - Locating appropriate server
    - Forwarding requests to server
    - Transmitting results and exceptions to client



3. Associated Middleware Platforms

Middleware provides a higher-level programming abstraction and, through layering, to abstract over heterogeneity in the underlying infrastructure to promote interoperability and portability

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Figure 2.12
Categories of Middleware

Fundamental Models

- Definition
  - Models based on the fundamental properties
- Models of systems so far
  - Share some fundamental properties
    - All of them are composed of processes
  - Share the design requirements of
    - Achieving the performance and reliability characteristics of processes and networks
    - Ensuring the security of the resources in the system
- Purpose of models based on the fundamental properties
  - To make explicit all the relevant assumptions about the systems we are modelling
  - To make generalizations concerning what is possible or impossible, given those assumptions.

Interaction Model

- Many processes are interacting in complex ways
  - Multiple server processes may cooperate with one another to provide a service
  - A set of peer processes may cooperate with one another to achieve a common goal
- **Algorithm**
  - A sequence of steps to be taken in order to perform a desired computation
  - Simple programs are controlled by algorithms in which the steps are strictly sequential
  - The behavior of the program and the state of the program's variables is determined by the algorithms
  - Such a program is executed as a single process

Interaction model

- **Distributed Algorithm**
  - Distributed systems composed of multiple processes
  - Their behaviour and state can be described by a distributed algorithm
  - A definition of the steps to be taken by each of the processes, including the transmission of messages between them
    - Messages are transmitted between processes to transfer information between them and to coordinate their activity
  - The rate at which each process proceeds and the timing of the transmission of messages between them cannot in general be predicted.
- **Two significant factors** affecting interacting processes :
  - Communication performance is often a limiting characteristic
  - It is impossible to maintain a single global notion of time

Performance Characteristics of Communication Channels

- **Latency**
  - The delay between the start of a message's transmission from one process and the beginning of its receipt by another process
    - The time taken for the first of a string of bits transmitted through a network to reach its destination
    - The delay in accessing the network
      - Increases significantly when the network is heavily loaded
    - The time taken by the operating system communication services at both the sending and the receiving processes
- **bandwidth**
  - Total amount of information that can be transmitted over it in a given time
- **Jitter**
  - The variation in the time taken to deliver a series of messages
  - Relevant to multimedia data