

Distributed Computing Paradigms

Distributed Software Systems
CS 707

Distributed Software Systems

1

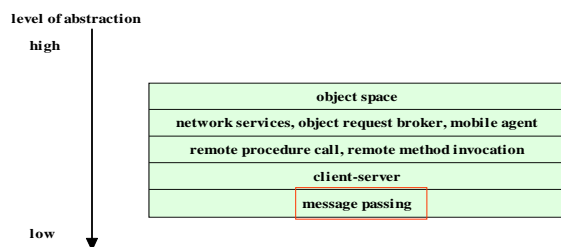
Paradigms for Distributed Applications

- ⌘ It is useful to identify the basic patterns or models of distributed applications, and classify the detail according to these models.
- ⌘ Characteristics that distinguish distributed applications from conventional applications running on a single machine are:
 - ⑩ *Interprocess communication*: A distributed application require the participation of two or more independent entities (processes). To do so, the processes must have the ability to exchange data among themselves.
 - ⑩ *Event synchronization*: In a distributed application, the sending and receiving of data among the participants of a distributed application must be synchronized.

Distributed Software Systems

2

Distributed Application Paradigms



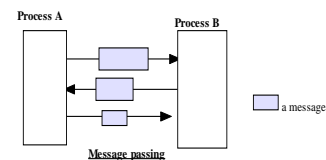
Distributed Software Systems

3

The Message Passing Paradigm

Message passing is the most fundamental paradigm for distributed applications.

- ⌘ A process sends a message representing a request.
- ⌘ The message is delivered to a receiver, which processes the request, and sends a message in response.
- ⌘ In turn, the reply may trigger a further request, which leads to a subsequent reply, and so forth.



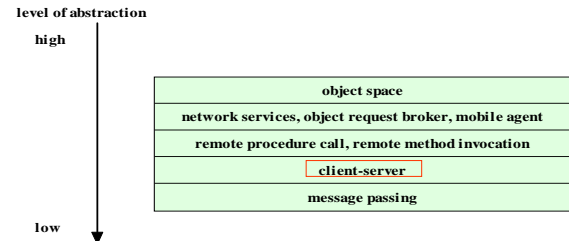
Distributed Software Systems

4

The Message Passing Paradigm - 2

- ⌘ The basic operations required to support the basic message passing paradigm are *send*, and *receive*.
 - ⌘ For connection-oriented communication, the operations *connect* and *disconnect* are also required.
 - ⌘ With the abstraction provided by this model, the interconnected processes perform input and output to each other, in a manner similar to file I/O. The I/O operations encapsulate the details of network communication at the operating-system level.
 - ⌘ The socket application programming interface is based on this paradigm.
- <http://java.sun.com/products/jdk/1.2/docs/api/index.html>
<http://www.sockets.com/>

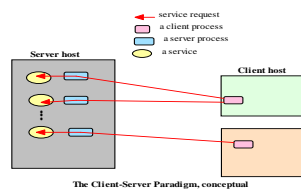
Distributed Application Paradigms



The Client-Server Paradigm

Perhaps the best known paradigm for network applications, the client-server model assigns asymmetric roles to two collaborating processes.

One process, the server, plays the role of a service provider which waits passively for the arrival of requests. The other, the client, issues specific requests to the server and awaits its response.



The Client-Server Paradigm - 2

- ⌘ Simple in concept, the client-server model provides an efficient abstraction for the delivery of services.
- ⌘ Operations required include those for a server process to listen for and to accept requests, and for a client process to issue requests and accept responses.
- ⌘ By assigning asymmetric roles to the two sides, event synchronization is simplified: the server process waits for requests, and the client in turn waits for responses.
- ⌘ Many Internet services are client-server applications. These services are often known by the protocol that the application implements. Well known Internet services include HTTP, FTP, DNS, finger, gopher, etc.

The Peer-to-Peer System Architecture

<http://www.peer-to-peerwg.org/whatis/index.html>

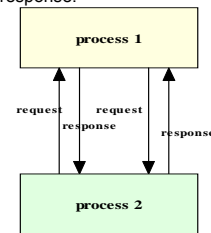
- ⌘ In system architecture and networks, peer-to-peer is an architecture where computer resources and services are directly exchanged between computer systems.
- ⌘ These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files...
- ⌘ In such an architecture, computers that have traditionally been used solely as clients communicate directly among themselves and can act as both clients and servers, assuming whatever role is most efficient for the network.

Distributed Software Systems

9

The Peer-to-Peer Distributed Computing Paradigm

In the peer-to-peer paradigm, the participating processes play equal roles, with equivalent capabilities and responsibilities (hence the term "peer"). Each participant may issue a request to another participant and receive a response.



Distributed Software Systems

10

Peer-to-Peer distributed computing

Whereas the client-server paradigm is an ideal model for a centralized network service, the peer-to-peer paradigm is more appropriate for applications such as instant messaging, peer-to-peer file transfers, video conferencing, and collaborative work. It is also possible for an application to be based on both the client-server model and the peer-to-peer model.

A well-known example of a peer-to-peer file transfer service is **Napster.com** or similar sites which allow files (primarily audio files) to be transmitted among computers on the Internet. It makes use of a server for directory in addition to the peer-to-peer computing.

Distributed Software Systems

11

The future of peer-to-peer

The file-swapping technology popularized by Napster, known as peer-to-peer networking, is about to change how people and corporations use the Internet. Instead of relying on central servers to process and relay information, new applications being developed will allow users to turn any computing device into a server.

A virtual meeting room

Users login to the Internet using a program that looks like an online chat room.

A file is placed into a "shared space" within the virtual meeting room, which allows users to work on data files at the same time.

Users work in real time and can instant message each other. In the future, this might be done through devices such as handhelds and cell phones.

What is on the screen

Names of online users

File that is being worked on

Instant messaging



SOURCES: Groove Networks; compiled from AP wire reports

distributed software systems

12

Peer-to-Peer distributed computing

The peer-to-peer paradigm can be implemented with facilities using any tool that provide message-passing, or with a higher-level tool such as one that supports the point-to-point model of the Message System paradigm.

For web applications, the *web agent* is a protocol promoted by the XNSORG (the XNS Public Trust Organization) for peer-to-peer interprocess communication

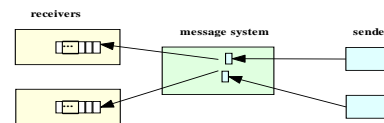
"Project JXTA is a set of open, generalized peer-to-peer protocols that allow any connected device (cell phone, to PDA, PC to server) on the network to communicate and collaborate. JXTA is short for Juxtapose, as in side by side. It is a recognition that peer to peer is juxtapose to client server or Web based computing -- what is considered today's traditional computing model. "

Distributed Software Systems

13

The Message System Paradigm

- ✳ The Message System or Message-Oriented Middleware (MOM) paradigm is an elaboration of the basic message-passing paradigm.
- ✳ In this paradigm, a message system serves as an intermediary among separate, independent processes.
- ✳ The message system acts as a switch for messages, through which processes exchange messages asynchronously, in a decoupled manner.
- ✳ A sender deposits a message with the message system, which forwards it to a message queue associated with each receiver. Once a message is sent, the sender is free to move on to other tasks.



Distributed Software Systems

14

Two subtypes of message system models

The Point-To-Point Message Model

- ✳ In this model, a message system forwards a message from the sender to the receiver's message queue. Unlike the basic message passing model, the middleware provides a message depository, and allows the sending and the receiving to be decoupled. Via the middleware, a sender deposits a message in the message queue of the receiving process. A receiving process extracts the messages from its message queue, and handles each one accordingly.
- ✳ Compared to the basic message-passing model, this paradigm provides the additional abstraction for asynchronous operations. To achieve the same effect with basic message-passing, a developer will have to make use of threads or child processes.

Distributed Software Systems

15

The Publish/Subscribe Message Model

- ✳ In this model, each message is associated with a specific topic or event. Applications interested in the occurrence of a specific event may subscribe to messages for that event. When the awaited event occurs, the process publishes a message announcing the event or topic. The middleware message system distributes the message to all its subscribers.
- ✳ The publish/subscribe message model offers a powerful abstraction for multicasting or group communication. The *publish* operation allows a process to multicast to a group of processes, and the *subscribe* operation allows a process to listen for such multicast.

Distributed Software Systems

16

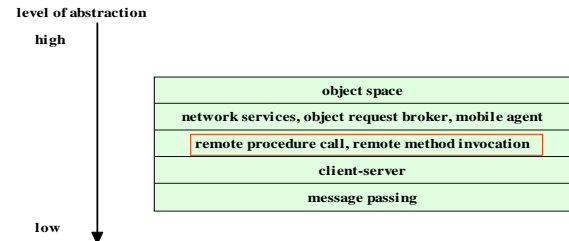
Toolkits based on the Message-System Paradigm

- ⌘ The MOM paradigm has had a long history in distributed applications.
- ⌘ Message Queue Services (MQS) have been in use since the 1980's.
- ⌘ The IBM MQ*Series is an example of such a facility.
<http://www-4.ibm.com/software/ts/mqseries/>
- ⌘ Other existing support for this paradigm are
 - ☐ Microsoft's Message Queue (MSQ),
http://msdn.microsoft.com/library/psdk/msmq/msmq_overview_41lh.htm
 - ☐ Java's Message Service
<http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/>

Distributed Software Systems

17

Distributed Application Paradigms



Distributed Software Systems

18

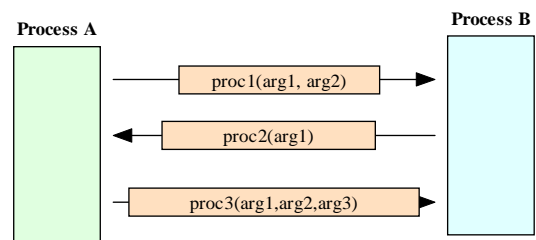
Remote Procedure Call

- ⌘ As applications grew increasingly complex, it became desirable to have a paradigm which allows distributed software to be programmed in a manner similar to conventional applications which run on a single processor.
- ⌘ The Remote Procedure Call (RPC) model provides such an abstraction. Using this model, interprocess communications proceed as procedure, or function, calls, which are familiar to application programmers.
- ⌘ A remote procedure call involves two independent processes, which may reside on separate machines. A process, *A*, wishing to make a request to another process, *B*, issues a procedure call to *B*, passing with the call a list of argument values. As in the case of local procedure calls, a remote procedure call triggers a predefined action in a procedure provided by process *B*. At the completion of the procedure, process *B* returns a value to process *A*.

Distributed Software Systems

19

Remote Procedure Call - 2



Distributed Software Systems

20

Remote Procedure Call - 3

- ⌘ RPC allows programmers to build network applications using a programming construct similar to the local procedure call, providing a convenient abstraction for both interprocess communication and event synchronization.
- ⌘ Since its introduction in the early 1980s, the Remote Procedure Call model has been widely in use in network applications.
- ⌘ There are two prevalent APIs for Remote Procedure Calls.
 - ☐ The *Open Network Computing Remote Procedure Call*, evolved from the RPC API originated from Sun Microsystems in the early 1980s.
 - ☐ The *Open Group Distributed Computing Environment (DCE) RPC*.
- ⌘ Both APIs provide a tool, *rpcgen*, for transforming remote procedure calls to local procedure calls to the stub.

Distributed Software Systems

21

The Distributed Objects Paradigms

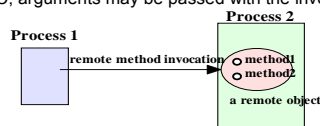
- ⌘ The idea of applying object orientation to distributed applications is a natural extension of object-oriented software development.
- ⌘ Applications access objects distributed over a network.
- ⌘ Objects provide methods, through the invocation of which an application obtains access to services.
- ⌘ Object-oriented paradigms include:
 - ☐ Remote method invocation (RMI)
 - ☐ Network services
 - ☐ Object request broker
 - ☐ Object spaces

Distributed Software Systems

22

Remote Method Invocation (RMI)

- ⌘ Remote method invocation is the object-oriented equivalent of remote method calls.
- ⌘ In this model, a process invokes the methods in an object, which may reside in a remote host.
- ⌘ As with RPC, arguments may be passed with the invocation.

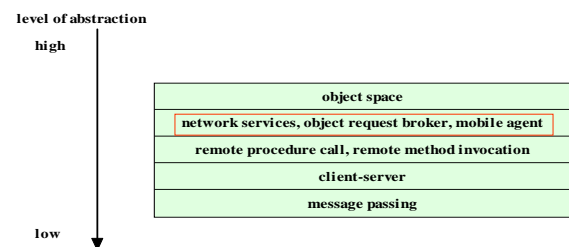


The Remote Method Call Paradigm

Distributed Software Systems

23

Distributed Application Paradigms

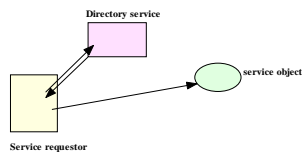


Distributed Software Systems

24

The Network Services Paradigm

- ⌘ In this paradigm, service providers register themselves with directory servers on a network. A process desiring a particular service contacts the directory server at run time, and, if the service is available, will be provided a reference to the service. Using the reference, the process interacts with the service.
- ⌘ This paradigm is essentially an extension of the remote method call paradigm. The difference is that service objects are registered with a global directory service, allowing them to be looked up and accessed by service requestors on a federated network.
- ⌘ Java's Jini technology is based on this paradigm.

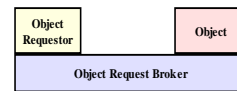


Distributed Software Systems

25

The Object Request broker Paradigm

- ⌘ In the object broker paradigm, an application issues requests to an *object request broker* (ORB), which directs the request to an appropriate object that provides the desired service.
- ⌘ The paradigm closely resembles the remote method invocation model in its support for remote object access. The difference is that the object request broker in this paradigm functions as a middleware which allows an application, as an object requestor, to potentially access multiple remote (or local) objects.
- ⌘ The request broker may also function as a mediator for heterogeneous objects, allowing interactions among objects implemented using different APIs and/or running on different platforms.



Distributed Software Systems

26

The Object Request broker Paradigm - 2

- ⌘ This paradigm is the basis of the Object Management Group's CORBA (Common Object Request Broker Architecture) architecture.
<http://www.corba.org/>
- ⌘ Tool kits based on the architecture include:
 - ☒ Inprise's Visibroker <http://www.inprise.com/visibroker/>
 - ☒ Java's Interface Development Language (Java IDL)
<http://java.sun.com/products/jdk/idl/>
 - ☒ Orbix's IONA, and TAO from the Object Computing, Inc.
<http://www.corba.org/vendors/pages/iona.html>

Distributed Software Systems

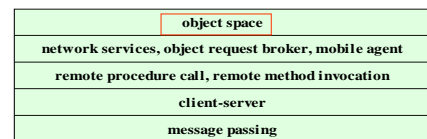
27

Distributed Application Paradigms

Level of abstraction

high

low

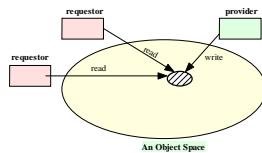


Distributed Software Systems

28

The Object Space Paradigm

- ⌘ The most abstract of the object-oriented paradigms, the object space paradigm assumes the existence of logical entities known as *object spaces*.
- ⌘ The participants of an application converge in a common object space.
- ⌘ A provider places objects as entries into an object space, and requesters who subscribe to the space access the entries.



Distributed Software Systems

29

The Object Space Paradigm - 2

- ⌘ In addition to the abstractions provided by other paradigms, the object space paradigm provides a virtual space or meeting room among providers and requesters of network resources or objects. This abstraction hides the detail involved in resource or object lookup needed in paradigms such as remote method invocation, object request broker, or network services.
- ⌘ Current facilities based on this paradigm include JavaSpaces

<http://java.sun.com/products/javaspaces/>.

Distributed Software Systems

30

Component-based Technologies

- ⌘ Component-based technologies such as Microsoft's COM, Microsoft DCOM, Java Bean, and Enterprise Java Bean are also based on distributed-object paradigms, as components are essentially specialized, packaged objects designed to interact with each other through standardized interfaces.
- ⌘ In addition, *application servers*, popular for enterprise applications, are middleware facilities which provide access to objects or components.

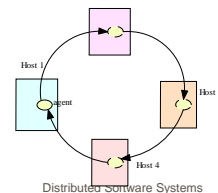
IBM's WebSphere,
<http://www.as400.ibm.com/products/websphere/docs/as400v35/docs/admover.html>

Distributed Software Systems

31

The Mobile Agent Paradigm

- ⌘ A mobile agent is a transportable program or object.
- ⌘ In this model, an agent is launched from an originating host.
- ⌘ The agent travels from host to host according to an itinerary that it carries.
- ⌘ At each stop, the agent accesses the necessary resources or services, and performs the necessary tasks to accomplish its mission.



Distributed Software Systems

32

The Mobile Agent Paradigm - 2

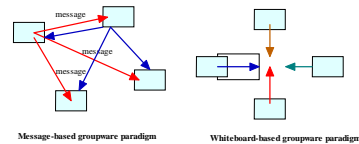
- ⌘ The paradigm offers the abstraction for a transportable program or object.
- ⌘ In lieu of message exchanges, data is carried by the program/object as the program is transported among the participants.
- ⌘ Commercial packages which support the mobile agent paradigm include:
 - ☒ Mitsubishi Electric ITA's Concordia system
<http://www.meitca.com/HSL/Projects/Concordia/Welcome.html>
 - IBM's Aglet system.
<http://www.tri.ibm.co.jp/aglets/>

Distributed Software Systems

33

The Collaborative Application (Groupware) Paradigm

- ⌘ In this model, processes participate in a collaborative session as a group. Each participating process may contribute input to part or all of the group.
- ⌘ Processes may do so using:
 - ☒ multicasting to send data to all or part of the group, or they may use a
 - ☒ virtual *sketchpads* or *whiteboards* which allows each participant to read and write data to a shared display.



Distributed Software Systems

34

Summary - 1

- ⌘ We have looked at a wide range of paradigms for distributed applications.
- ⌘ The paradigms presented were:
 - ☒ Message passing
 - ⌚ Client-server
 - ☒ Message system: Point-to-point; Publish/Subscribe
 - ☒ Distributed objects:
 - ☒ Remote method invocation
 - ☒ Object request broker
 - ☒ Object space
 - ⌚ Mobile agents
 - ⌚ Network services
 - ⌚ Collaborative applications

Distributed Software Systems

35

Summary - 2

- ⌘ To varying degrees, these paradigm provide abstractions that insulate the developers from the detail of interprocess communication and event synchronization, allowing the programmer to concentrate on the bigger picture of the application itself.
- ⌘ In choosing a paradigm or a tool for an application, there are **tradeoffs** that should be considered, including **overheads**, **scalability**, **cross-platform support**, and **software engineering** issues.

Distributed Software Systems

36