Paradigm Term Paper

Distributed Computing Paradigms

Name: Shiyun Liu

Course: Advanced Software Paradigms

Professor: Dr. A. Bellaachia

Date: Oct. 3, 2012

ABSTRACT

In modern society, widespread Internet gradually becomes one of the most powerful way for us to issue and absorb information. World Wide Web which is supported by the Internet is closely bound up with our daily lives.

With the development of science and technology, the Internet has become a univers platform for distributed computing. And the Web also offers a simple, consistent, and uniform model of distributed documents referenced by URL name.

This article aims to introduce the distributed computing paradigms. A distributed system is a collection of independent computers, interconnect via a network, capable of collaborating on a task. And paradigm means “a pattern, example, or model”.

In this paper, six paradigms for distributed applications will be included. They are Message Passing Paradigm, the Client-Server Paradigm, the Peer-to-Peer Paradigm, the Message System Paradigm, Remote Procedure Call Model, the Distributed Objects Paradigms.

In the future, more distributed computing paradigms will occur. Of course, these paradigms are important because in the study of any subject of great complexity, it is useful to identify the basic patterns or models, and classify the detail according to these models.

Table of Content

1. Introduction

A distributed system is a collection of independent computers that appears to its users as a single coherent system. (Andrew S. Tanenbaum & Maarten van Steen, 2007) The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way. (Andrew S. Tanenbaum & Maarten van Steen, 2007)

This paper analyzes several paradigms of distributed system, like Message Passing Paradigm, the Client-Server Paradigm, the Peer-to-Peer Paradigm, the Message System Paradigm, Remote Procedure Call Model, the Distributed Objects Paradigm.

Paradigm means a pattern, example, or Model. It’s the similarity of various distributed systems. In the study of any subject of great complexity, it is useful to identify the basic patterns or models, and classify the detail according to these models. (Jing Li, 2006) With the knowledge of the paradigms, we can easy to build or analyze the future complexity distributed systems.

2. Related Work

Although software or website development is well known of the world, a lot of new technics still occur year by year. Distributed system and its paradigms is an important technique for building a more efficient Internet, but only a little meaningful related work of its paradigms come out.

Paradigm can be seemed as abstraction at early time, so from David J. Barnes1 ”We often use abstraction when it is not necessary to know the exact details of how something works or is represented, because we can still make use of it in its simplified form.” So the distributed system paradigm will simplified the development procedure without having to build the things which have been realized.

In the book named *Distributed Systems: Principles and Paradigms,* Andrew S. Tanenbaum & Maarten van Steen provide four paradigms. They are Object-Based Paradigm (like CORBA), Distributed File Paradigm (like NFS), Document-Based Paradigm (like WWW) and Cooperation-Based Paradigm (like TIB/Rendezvous).

In addition, Liu M.L., , writes the *Distributed Computing: Principles And Applications* and talks about six paradigms. They are Message Passing Paradigm, the Client-Server Paradigm, the Peer-to-Peer Paradigm, the Message System Paradigm, Remote Procedure Call Model, the Distributed Objects Paradigms.

And this paper will focus on Liu M.L.'s classification of paradigms.

3. Paradigms of distributed systems.

This paper will introduce six paradigms of distributed systems. They are Message Passing Paradigm, the Client-Server Paradigm, the Peer-to-Peer Paradigm, the Message System Paradigm, Remote Procedure Call Model, the Distributed Objects Paradigms.

3.1 Message Passing Paradigm

Message passing is the most fundamental paradigm for distributed applications. Fig. 2.1 shows the message passing Paradigm.

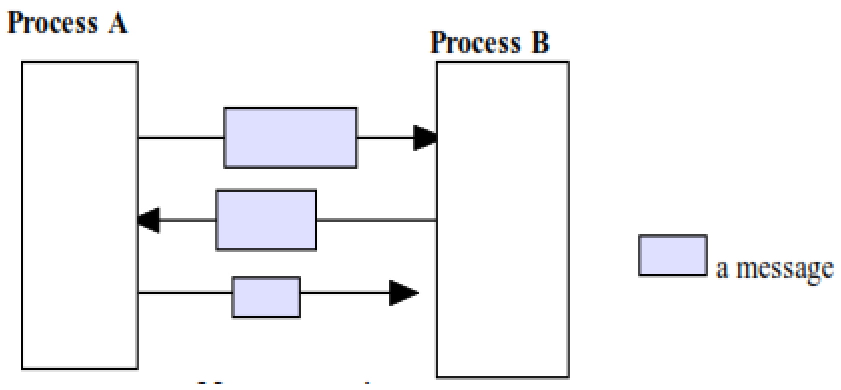

Fig.2.1 Message Passing Paradigm

There are many messages passing between Process A and Process B.

In this paradigm, A process sends a message representing a request. And then the message is delivered to a receiver, which will process the request and sends a message in response. In turn, the reply message may trigger a further request, which leads to a subsequent reply, and so forth.

To support the basic message passing paradigm, several basic operations are required: (1) Send and receive (2) connect and disconnect (for connection-oriented communication)

With the abstraction provided by this model, the interconnected processe perform input and output to each other, in a manner similar to file I/O. The I/O operations encapsulate the detail of network communication at the operating-system level. (Jing Li, 2006)

3.2 The Client-Server Paradigm

The client-server paradigm assigns asymmetric roles to two collaborating processes. One is as client and the other is as Server.

The client issues specific requests to the server and waits for its response, while the server plays the role of a service provider which waits for the arrival of requests passively. And Fig. 2.2 shows the Client-Server paradigm

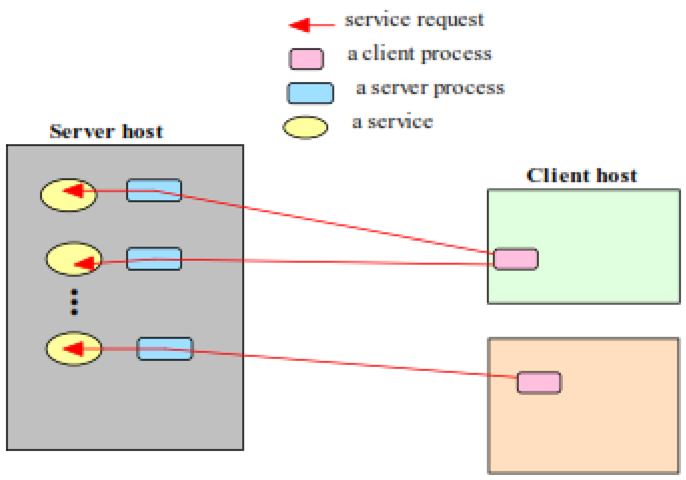

Fig.2.2 Client-Server Paradigm

In this paradigm, there are two basic parts: Client and Server. They communicate with each other by sending requests and receiving response.

The client-server model provides an efficient abstraction for the delivery of network services. Each of the two collaborating processes has its own required operations. For the client, it can issue requests and accept responses and for the server, listening and accepting requests are needed.

By assigning asymmetric roles to the two sides, event synchronization is simplified to that the server process waits for requests, and the client in turn waits for responses. So many Internet services are client-server applications. These services are often known by the protocol that the application implements.

3.3 The Peer-to-Peer Paradigm

In system architecture and networks, peer-to-peer is an architecture where computer resources and services are direct exchanged between computer systems. (Liu M.L., 2004) These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files.

By using this paradigm, all computers in the network that have traditionally been used solely as clients can communicate directly among themselves. It means that computers can act as both clients and servers, assuming whatever role is most efficient for the network. Fig.2.3 shows the peer-to-peer paradigm

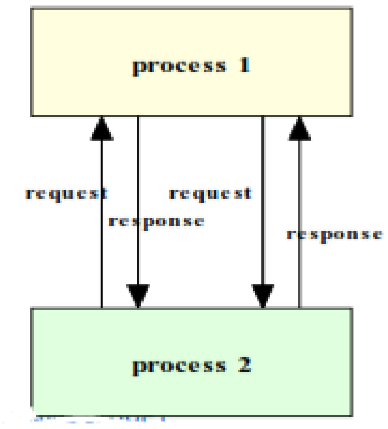

Fig. 2.3 Peer-to-Peer Paradigm

In this peer-to-peer paradigm, the participating processes play equal roles with equivalent capabilities and responsibilities. There are not important or unimportant effects or functions of all processes. Each participant may issue a request to another participant and receive a response from others.

Different from client-server paradigm which 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. It makes use of a server for directory in addition to the peer-to-peer computing.

3.4 The Message System Paradigm

The Message System or Message-Oriented Middleware (MOM) paradigm is an elaboration of the basic message passing paradigm. A message system serves as an intermediary among separate, independent processes. Fig. 2.4 shows the message system paradigm.

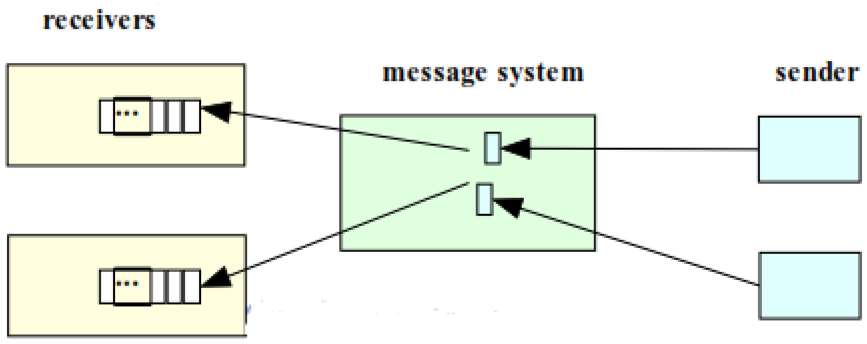

Fig. 2.4 Message System Paradigm

In this paradigm, there are three parts: Message system, receivers and senders. The message system acts as a switch for messages, through which processes exchange messages asynchronously, in a decoupled manner. It almost likes the basic massage passing paradigm but one thing that in this paradigm, the two functions, sending messages and receiving messages, are apart from each other. A sender saves a message with the message system and then forwards it to a message queue associated with each receiver. And once a message is sent, the sender is free to move onto other tasks.

Based on the basic message-passing model, this paradigm also provides an additional abstraction for asynchronous operations.

In the message system paradigm, there are two subtypes: One is the point-to-point message model and the other is the publish/subscribe message model.

In point-to-point model, senders send messages to receivers though the message system. In other words, message system forwards a message from the sender to the receiver's message queue. The middleware has a message depository and allows the sending and the receiving operations to be decoupled.

In publish/subscribe message model, each message is associated with a specific topic or event. Applications which prefer the occurrence of a specific event will subscribe to messages for that event.

3.5 Remote Procedure Call Paradigm

Using the Remote Procedure Call (RPC) model, inter-process communications proceed as procedure, or function, calls, which are familiar to the application programmers. Fig. 2.5 shows the remote procedure call paradigm.

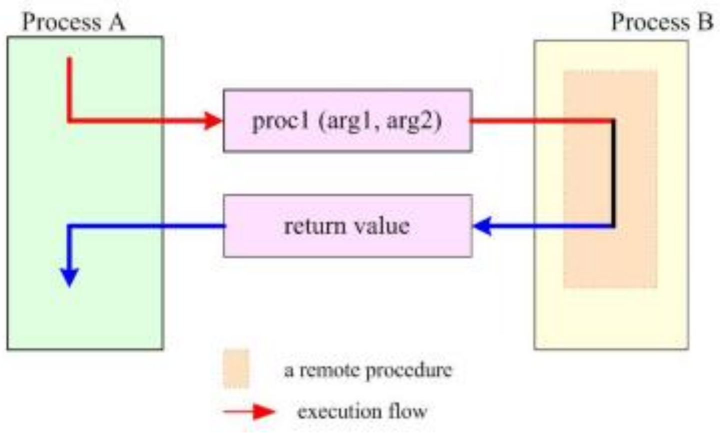

Fig. 2.5 Remote Procedure Call Paradigm

In this paradigm, there are two independent processes. Assume that process A wishes to make a request to another process B. Process A issues a procedure call to B, passing with the call a list of argument values. The remote procedure call triggers a predefined action in a procedure provided by process B. When the procedure is completed, process B returns a value to process A.

Compared to the message-passing model which is data-oriented with the actions triggered by the message exchanging, RPC model is action-oriented with the data passed as arguments. Developers use the RPC paradigm to build network applications, providing a convenient abstraction for both inter-process communication and event synchronization.

3.6 The Distributed Objects Paradigm

Applying object orientation model to distributed applications is an extension of object-oriented software development. Applications can access objects distributed over a network. And the objects offer methods by invoking which an application obtains access to services.

Distributed Object Paradigms contains four sub-paradigms. They are:

(1) Remote method invocation (RMI)

(2) Network services

(3) Object request broker
(4) Object spaces

3.6.1 Remote Method Invocation Sub-Paradigm

Remote method invocation (RMI) is the object-oriented equivalent of remote method calls. Fig. 2.6 shows the remote method invocation paradigm.

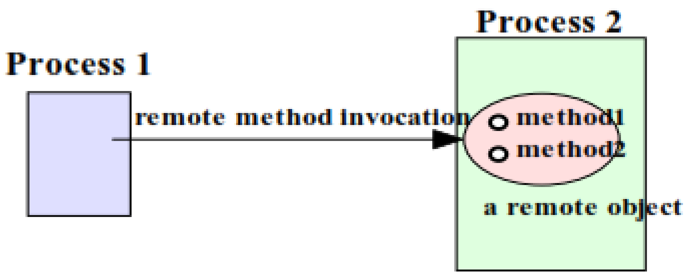

Fig. 2.6 Remote Method Invocation Sub-Paradigm

In this model, there are two processes and one of them contains methods in a remote object. Process 1 calls the methods with arguments in an object, which may reside in a remote host.

3.6.2 Network Services / Web Services Sub-Paradigm

This paradigm is essentially an extension of the remote method Invocation paradigm. The difference between them is that service objects are registered with a global directory service, which allows them to be looked up and accessed by service requestors on a federated network. Fig. 2.7 shows the network services / web services sub-paradigm.

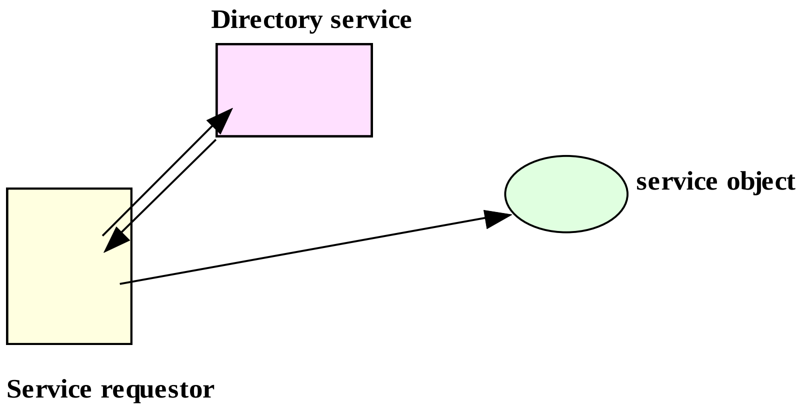

Fig.2.7 Network Services / Web Services Sub-Paradigm

In this paradigm, there are three parts: service requestor, directory service and service object. Directory server is used for registration. A process wants a particular service which will contact the directory server at run time. If the service is available, the server will be provided a reference to the service, so that the process interacts with the service.

3.6.3 Object Request Broker Sub-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. Fig. 2.8 shows the object request broker sub-paradigm

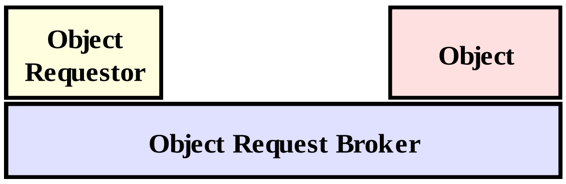

Fig.2.8 Object Request Broker Sub-Paradigm

The paradigm closely resembles the RMI model in its support for remote object access. On one hand, The difference between them is that the object request broker in this paradigm is looked as a middleware. The middleware allows an application to potentially access multiple remote (or local) objects. On the other hand, the request broker may also function as a mediator for heterogeneous objects. The mediator allows interactions among objects performed using different APIs or running on different platform

3.6.4 Object Spaces Sub-Paradigm

The object spaces paradigm assumes that the existence of logical entities known as object spaces.

Different from other paradigms, the object space paradigm provides a virtual space among provides and requesters of network resources or objects. This abstraction or virtual space hides the detail involved in resource and objects. Fig. 2.9 shows the object spaces sub-paradigm

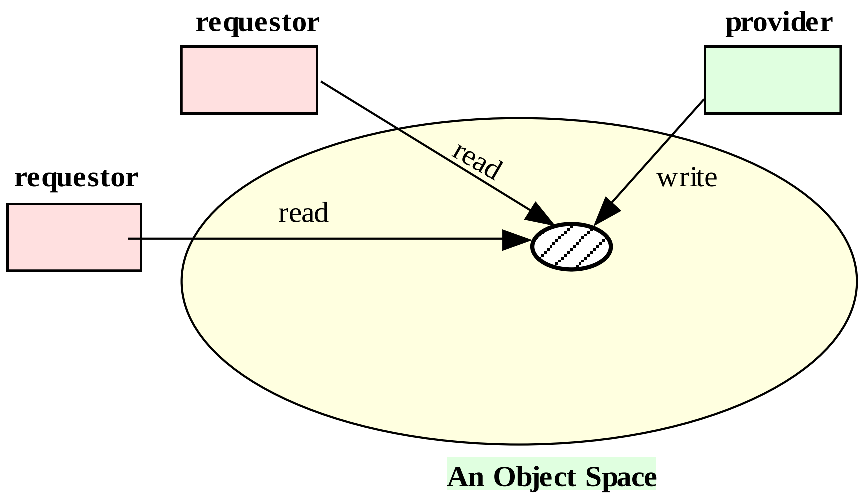

Fig.2.9 Object Spaces *Sub-Paradigm*

In this paradigm, there are three parts: an object space, requestors and providers. 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.

4 Applications of the distributed system paradigms

Paradigms are good for designing complex systems. A common definition of a programming paradigm: is that of a ``model or approach in solving a problem", or the system architecture encompassing definition of ``way of thinking about computer systems" (Stephen H. Kaisler, 2005). It focuses on algorithms reuse and many applications make uses of these six distributed paradigms.

4.1 The Applications using the six paradigms respectively

4.1.1 Message Passing Paradigm

The socket application programming interface is based on the message passing paradigm. Based on the logic structure of Socket, two processes can exchange data between each other. The sender writes or plugs the message into the socket and the receiver read or pick up the message from the socket.

4.1.2 The Client-Server Paradigm

Many Internet services support 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.

In TCP/IP, the inter-action between processes use the Client-Server model. Fig. 3.1 shows the TCP/IP application

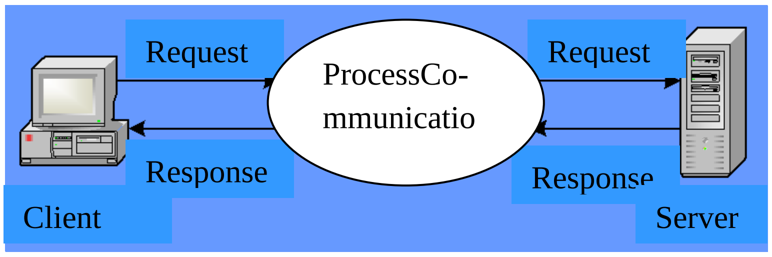

Fig. 3.1 TCP/IP Application

Client and Server denote the two communicated processes. Client send request message to Server, and Server make a response to the request and provide the required network service.

4.1.3 The Peer-to-Peer Paradigm

P2P paradigm is better fit for instant message, P2P file transmission, video session, cooperative work and so on. Napster.com is a famous instance of P2P file transmission service. The essence of Napster lies on guiding the Internet to decentralized management model from centralized management model and guiding the content to the edge of the network from the single center node. This will help make the most of processing ability and potential resources of a lot of end nodes. Fig. 3.2 shows the Napster application

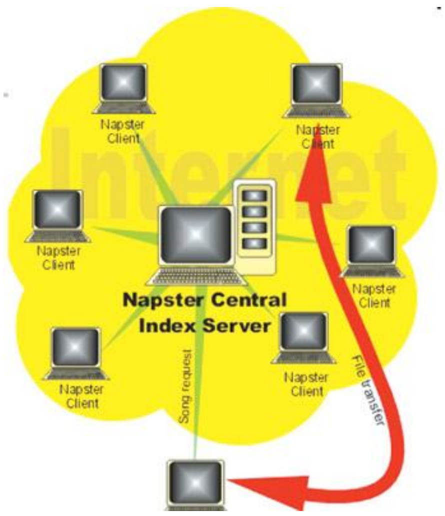

Fig. 3.2 Napster Application

In Napster, each client joins a server, where he registers its local files to the central index. A client make queries to the central server which returns references to the clients that actually hold the resources. The client connects to other peer clients and retrieves the resource. The selection is performed by the user but it could be done automatically based on bandwidth, load or other criteria.

4.1.4 The Message System Paradigm

There are many applications support the message system paradigm, like MQ*Series of IBM, MSMQ of Microsoft and message service of JAVA.

Take the WebSphere MQ as an example. WebSphere MQ is a complete universal messaging backbone, enabling rapid, reliable, and secure transport of messages and data between applications, systems, and services. WebSphere MQ accelerates the flow of messages across organizations without complexity, cutting costs and reducing maintenance, improving business responsiveness and connecting to new solutions and technologies from the mainframe to the mobile enterprise.

4.1.5 Remote Procedure Call Paradigm

There are two prevalent APIs for Remote Procedure Calls. One is the Open Network Computing Remote Procedure Call, evolved from the RPC API originated from Sun Microsystems in the early 1980s. and the other is 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 (Liu M.L., 2004).

4.1.6 The Distributed Objects Paradigm

JavaSpaces is a current facility based on the distributed paradigm. Compared with technologies such as message queues and databases, Javaspaces helps perform some of the tasks supported by these other technologies, they should be treated very differently.

A JavaSpaces service can store persistent data which will be searched later. However, JavaSpaces service is not a relational or object database. It’s designed to help solve problems in distributed computing, rather than to be as a data repository

JavaSpaces provide a simple API for the manipulation of Entry objects:
(1) Write - places a new Entry in the space.

(2) Read - returns a copy of an Entry matching a particular template.
(3) Take - removes an Entry matching a particular template from the space and returns it to the caller.

In the normal Java fashion, entry objects are not serialized. Only the public fields can be written to the JavaSpaces and each field is serialized separately. If two fields reference the same object, each of them will hold a separate copy of the original object when the Entry is recovered from the JavaSpace,

For example, Fig.3.3 shows the JavaSpaces application which can be searched on the Internet by the website address:
http://river.apache.org/doc/specs/html/js-spec.html

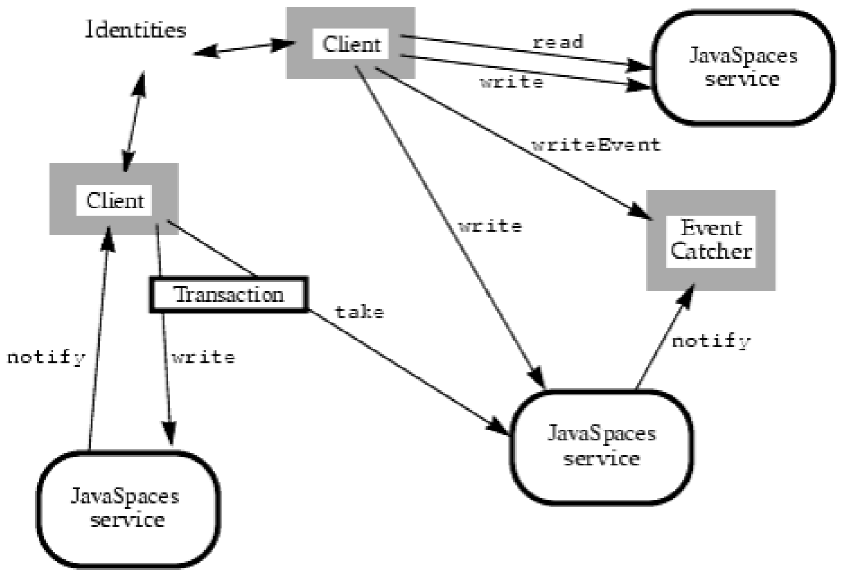

Fig.3.3 JavaSpaces Application

In JavaSpaces application, clients perform operations that map entries or templates onto JavaSpaces services which can be singleton operations or contained in transactions so that all or none of the operations will take place. In addition, a single client can interact with spaces as many as it needs to. Identities can be accessed from the security subsystem and passed parameters to method calling and notifications will be sent to event catchers, which may be clients themselves or proxies for a client.

4.2 An Application using all of the six paradigms.

An auction system can be looked as a common system which will have different effects after using different paradigms or abstracts. In the auction system, every session will deal with only one auction object which means one session will provide only one auction object for bidders to bid. And at the end of the auction session, auctioneer announces the auction result.

4.2.1 Using Message paradigm with Client-Server paradigm.

Every participant and auctioneer process play roles of Client or Server.

For session control, Server works as a participant which waits to hear an announcement from the auctioneer. And Client works as the auctioneer which sends a request that announces the above three types of events.

For accepting bids, Client works as a participant which sends a new bids to a server. And Server works as an auctioneer which accepts new bids and updates the current highest bid.

4.2.2 Using Peer-to-Peer Paradigm

If there is some available tools which support the peer-to-peer paradigm, the realization of auction system will be simplified greatly.

In the auction system, a participant can contact the auctioneer directly to register for the auction and the auctioneer contacts each participant to initiate the auction session. Each of the participants will obtain the latest status and submit a bid if they want. At the end of the auction, the winning bidder can be notified by the auctioneer. The other participants may know about the result by contacting the auctioneer.

4.2.3 Using Peer-to-Peer Paradigm

We know that there are two sub-paradigms of peer-to-peer paradigm: point-to-point model and publish/subscribe model. Both of them can be applied for the auction system.

(1) Point-to-Point Paradigm:
This paradigm provides the additional abstraction for asynchronous operations. And to achieve the same effect with basic message-passing, a developer will have to make use of threads or child processes.

If using the point-to-point paradigm, the realization of the auction system almost the same with using the basic message paradigm. The only difference is the message are transmitted by the middleware, so that sending messages and receive messages are separated with each other.

(2) Publish/Subscribe Paradigm:
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, while the subscribe operation allows a process to listen for such multicast
If using publish/subscribe paradigm, the auction system can be realized like following (Jing Li, 2006).

- Each participant subscribes to a being-auction event message.
- The auctioneer signifies the beginning of the auctioning session to an end-auction event message.
- Upon receiving the begin-auction event, a participant subscribes to an end-auction event message.
- The auctioneer subscribes to message for new-bid events.
- A participant wishing to place a new bid issues a new-bid event, which will be forwarded to the auctioneer.
- At the end of the session, the auctioneer issues an end auction event message to inform all participants of the outcome.

4.2.4 Using Remote Procedure Call Paradigm

If we applied the remote procedure call paradigm to the auction system, it can be realized as follows:

The auctioning program provides two remote procedures:
- A remote procedure for each participant to register itself
- A remote procedure for a participant to make a bid

Each participant program provides the following remote procedures:

- Allow the auctioneer to call a participant to announce the onset of the session
- Allow the auctioneer to inform a participant of a new highest bid
- Allow the auctioneer to announce the end of the session(Liu M.L. 2004).

4.2.5 Using distributed objects paradigm

The distributed objects paradigm applies the object orientation idea into the distributed applications.

There are four sub-paradigms of this distribute objects paradigm.

- Remote method invocation (RMI)
- Network services
- Object request broker
- Object spaces

(1) Remote Method Invocation

If using the remote method invocation model, the realization of auction system is almost the same with RPC (Remote Procedure Call) paradigm. The only difference is applying object method instead of procedure method.

(2) Object Request Broker Paradigm

If using the object request broker model, the realization of auction system is almost the same with RMI (Remote Method Invocation) paradigm. The only difference is that every object (auctioneers and bidders) of the object request broker model must have been registered on the ORB (Object Request Broker) and accept the ORB requests.

The bidders send the registration requests to auctioneer object to participate in bidding. Though the ORB, auctioneer can invoke every bidder object to make an announcement that the auction is started, update the newest price status and declare that the auction is ended.

(3) Object Spaces Paradigm

For the auction system, every participant and service provider are all need to subscribe for a public object spaces.

Every participant must put an object into its object space to register for the sessions and get the notification when the auction starts.

At the begging of the auction, the auctioneer puts an object into its corresponding object space. It contains the information of the waiting auction object and the auction history.

Bidders retrieve this object from the object spaces. If the bidder wants to bid higher than the current price, he or she needs to put a new price into this object before return this object to its object spaces.

When session is end, auctioneer will retrieve the object from the object spaces and contract with the bidder who has given the highest price.

5 Conclusion

This paper talks about six paradigms of distributed paradigms, which are Message Passing Paradigm, the Client-Server Paradigm, the Peer-to-Peer Paradigm, the Message System Paradigm, Remote Procedure Call Model, the Distributed Objects Paradigms. It also introduces several applications which make a use of these paradigms respectively and an auction system which can be applied by these six paradigms to understand the similarities and differences among these paradigms.

This article is meaningful, because it help both students and teachers to know the paradigms which can be used in distributed system and it is useful to identify the basic patterns or models, so that the developers can easy to build or analyze the future complexity distributed systems.

There is no doubt that something needs to be improved and more paradigms of the distributed systems will come out in the future.

Reference

Stephen H. Kaisler, 2005, Software Paradigms, John Wiley & Sons, Inc., Hoboken, New Jersey, Canada

Robert W. Sebesta, 2012, The Tenth Edition, Concepts of Programming Languages, Pearson Education, Inc.

Andrew S. Tanenbaum & Maarten van Steen, 2007, The second Edition, Distributed Systems: Principles and Paradigms, Pearson Education. Inc.

Liu M.L., 2004, Distributed Computing: Principles And Applications, Pearson Education, Inc.

Jing Li, 2006, Distributed Computing Paradigms, School of Software Engineering of BUPT

http://www.dancres.org/cottage/javaspaces.html
http://river.apache.org/doc/specs/html/js-spec.html

Wikipedia & Baidupedia