Distributed Systems - Middlewares

P-O Östberg

2007-09-07
1 Middlewares

2 Remote Method Invocation
   Java RMI
   Corba

3 Remote Procedure Call
   Sun RPC
Middlewares

- Abstraction layer for inter-process communication (IPC)
- Offer programming models that hide underlying details
- Handles data and call marshalling
- Provides call semantics
Middlewares

Applications

RMI, RPC and events

Request-Response protocol

External data representation

Operating System

Middleware layers
Middleware Example: Globus

- Middleware and development toolkit for Grid computing
- Provides a web service-based programming model
- Abstracts details of underlying systems
  (user systems, batch queues, monitoring programs, system heterogeneity issues, load balancing systems etc)
- Web services provide language and platform neutral access to system functionality
- Security models mapped between systems
Remote Method Invocation (RMI)

- Extends the object oriented model to the \textit{distributed object model}
- Objects are referenced using remote object references
- Objects publish \textit{remote interfaces} which are used to invoke service methods
- Call semantics for distributed objects differ from those of local objects
- Distributed garbage collection techniques employed for memory management (e.g., reference counting)
RMI Anatomy

![RMI Anatomy Diagram]

- **Client**
  - Object A
  - Proxy for B
  - Remote reference module
  - Communication module

- **Server**
  - Skeleton & dispatcher for B’s class
  - Remote reference module
  - Remote object B
  - Servant

- **Communication**
  - Request
  - Reply
Java RMI

- Java Remote Method Invocation
  
  [http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiT0C.html](http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiT0C.html)

- Distributed object model
  - runs in separate processes (possibly) on separate hosts

- Facilitates object calls between JVMs
  - marshalling transparent to programmers

- Integrated part of Java (J2SE)

- Language dependent (100% pure Java)
Java RMI Overview

- Server publishes remote object references in *RMIRegistry*
  - naming service maps names to remote object references
- Clients acquire remote object references via
  - name resolutions (*RMIRegistry*)
  - return values of remote method calls
- Clients use remote objects as regular Java objects
  - parameter semantics differ
- Classes can be dynamically downloaded by RMI
  - receive object of unknown class (automatic download)
  - allows dynamic introduction of new classes
Remote Objects

- Java objects that implement a remote interface
  - interface extends `java.rmi.Remote`
  - declares remote methods
  - methods in the remote interface throw `RemoteException`

- Clients perform remote method invocations via the remote interface (using only the methods in the remote interface)

- Clients must handle `RemoteExceptions`

- Parameter passing
  - local and remote objects may be passed as parameters / return values
  - all objects must implement `java.io.Serializable`
  - primitive types / local objects passed by value
  - remote objects passed by reference
Using Java RMI

1. Define a remote interface
2. Implement the remote interface
3. Write a client program
4. Compile
   - generate client stubs (`rmic <servant class>`) 
   - server and client classes
5. Make classes available over the network
Implementing a Java RMI Client

- Acquire an initial remote object reference
  - `Naming.lookup(String name)`
  - gives a Remote reference
- Handle RemoteException
- Regular Java code
Implementing a Java RMI Server

- Inherit UnicastRemoteObject
- Implement Remote interface(s)
- Install a security manager
- Publish remote object(s) in RMIRegistry (bootstrapping)
  - Naming.rebind(String name, Remote object)
Security

- Security Manager
  - protects local system resources from downloaded code
  - determines access rights
- RMISecurityManager (default)
- Alternative: policy file specifying rights
- Network traffic can be encrypted (using, e.g., SSL)
Example: Policy Files

grant {
    permission java.security.AllPermission;
}

grant {
    permission java.net.SocketPermission "*:1024-65536", "connect,accept";
    permission java.net.SocketPermission "*:80", "connect";
}
Threading Issues

• From the RMI specification:
  A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread. The RMI runtime makes no guarantees with respect to mapping remote object invocations to threads. Since remote method invocation on the same remote object may execute concurrently, a remote object implementation needs to make sure its implementation is thread-safe.

• Always protect concurrent data access (regardless of middleware)
Example: Result Service

- Result service, offering sport results to interested users using a publish-subscribe pattern
  - sport results are reported to a service
  - users can subscribe to results for specific games
  - When a game result update is submitted, the result service sends updates to subscribers

- Callback-based updates
  - avoids resource drains due to polling issues
  - client provides server with callback references for updates
  - server also acts as a client (and vice versa)
Example Architecture

- Remote interfaces
  - ResultService: handles subscriptions and results
  - ResultSubscriber: callback, receives updates

- Server classes
  - ResultServiceImpl: implements ResultService
  - ResultServer: instantiates and registers the service
  - Result: contains game result information
  - ResultSubscriberImpl: implements ResultSubscriber

- Client classes
  - ReportClient: reports results to a ResultService
  - SubscribeClient: creates game result subscriptions
Example Component Interactions

1. Naming.rebind()

2. Naming.lookup()

3. registerSubscriber()

4. Naming.lookup()

5. reportResult()

6. reportResult()

ReportClient

ResultServer

RMIRegistry

SubscribeClient
Example: Remote Interfaces

```java
public interface ResultService extends Remote {
    public void reportResult (Result result)
        throws RemoteException;
    public void registerSubscriber (ResultSubscriber subscriber, String match)
        throws RemoteException;
    public void deregisterSubscriber (ResultSubscriber subscriber, String match)
        throws RemoteException;
}

public interface ResultSubscriber extends Remote {
    public void reportResult(Result result)
        throws RemoteException;
}
```
Example: Result

```java
dpublic class Result
    implements Serializable
{
    protected String match = null;
    protected String result = null;
    public Result(String match, String result)
    {
        this.match = match;
        this.result = result;
    }

    public String getMatch()
    {
        return this.match;
    }

    public String getResult()
    {
        return this.result;
    }
}
```
Example: ResultServiceImpl 1/3

public class ResultServiceImpl extends UnicastRemoteObject
    implements ResultService
{
    protected Hashtable<String, List<ResultSubscriber>> subscriberMap;

    public synchronized void registerSubscriber(ResultSubscriber subscriber,
                                                 String match)
    throws RemoteException
    {
        List<ResultSubscriber> matchSubscribers = subscriberMap.get(match);
        if (matchSubscribers == null)
        {
            matchSubscribers = new ArrayList<ResultSubscriber>();
            subscriberMap.put(match, matchSubscribers);
        }
        matchSubscribers.add(subscriber);
    }
Example: ResultServiceImpl 2/3

```java
public synchronized void deregisterSubscriber (ResultSubscriber subscriber, String match) throws RemoteException {
    List<ResultSubscriber> matchSubscribers = subscriberMap.get(match);
    if (matchSubscribers != null) {
        matchSubscribers.remove(subscriber);
    }
}

public void reportResult(Result result) throws RemoteException {
    notifySubscribers(result);
}
```
private synchronized void notifySubscribers(Result result) {
    List<ResultSubscriber> matchSubscribers = subscriberMap.get(result.getMatch());
    if (matchSubscribers == null) {
        return;
    }
    Iterator<ResultSubscriber> subscriberIter = matchSubscribers.iterator();
    while (subscriberIter.hasNext()) {
        ResultSubscriber subscriber = subscriberIter.next();
        try {
            subscriber.reportResult(result);
        } catch (RemoteException e) {
            subscriberIter.remove();
        }
    }
}
Example: ResultServer

```
ResultService resultService = new ResultServiceImpl();
Naming.rebind("//localhost/result", resultService);
```
Example: ResultSubscriberImpl

```java
public class ResultSubscriberImpl extends UnicastRemoteObject
        implements ResultSubscriber
{
    public void reportResult(Result result)
            throws RemoteException
    {
        System.out.println("Result update: " + result.getMatch() + "\t" +
                result.getResult());
    }
}
```
Example: SubscribeClient

```java
ResultSubscriber subscriber = new ResultSubscriberImpl();
ResultService service = (ResultService) Naming.lookup("//localhost/result");
service.registerSubscriber(subscriber,"SWE-FIN");
System.out.println("Waiting for score reports ..."的职业);
System.in.read();
service.deregisterSubscriber(subscriber,"SWE-FIN");
...
```
Example: ReportClient

```java
Result result = new Result("SWE-FIN","1-0");
ResultService service = (ResultService)Naming.lookup("//localhost/result");
...
```
Using Java RMI

- Generate stubs and compile
- Start RMIRegistry
  - rmiregistry <port>
- Register objects
  - Naming.rebind("//<host>:<port>/object");
- Specify code base (class loading paths)
  - Djava.rmi.codebase=<class URLs>
- Specify security policy
  - Djava.security.policy=<policy file>
Corba

- Standardized framework for RMI
- Language dependent
  - multiple CORBA-implementations exists (multiple languages and OS)
- CORBA objects have interfaces and remote object references but can be implemented in non-OO languages (e.g., C)
  - marshalling more complicated
  - IDLs can not define classes
  - cannot send object instances
  - can send primitive types, aggregated types and remote objects references
Corba
Corba

- Interface Definition Language (IDL) - interface definitions
- Common Data Representation (CDR) - external data representations
- Interoperable Object References (IOR) - remote object references
- General Inter-ORB Protocol (GIOP) - communication protocol
- **Object adapter**
  - acts as a data marshalling bridge
  - acts as a dispatcher (via skeletons) to servant instances
- **Interface repository**
  - used for dynamic (non-proxy) call ("interface reflection")
- **Implementation repository**
  - locates and activates registered servers
compiled using an IDL-compiler to generate client proxies, server skeletons, classes and types for the IDL types.

```
struct Result
{
    string match;
    string score;
};

interface ResultSubscriber
{
    void reportResult(in Result score);
};

interface ResultService
{
    void reportResult(in Result score);
    void registerSubscriber(in ResultSubscriber subscriber, in string match);
    void deregisterSubscriber(in ResultSubscriber subscriber, in string match);
};
```
Example: Corba Servants

```java
public class ResultServiceServant extends ResultServicePOA
{
    ... // see Java RMI ResultServiceImpl
}

public class ResultSubscriberServant extends ResultSubscriberPOA
{
    ... // see Java RMI ResultSubscriberImpl
}
```
Example: Corba ResultService

```java
// Initialize ORB
ORB orb = ORB.init(args, null);

// The servant object must be registered with a POA.
// Obtain a reference to the root POA.
POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

// Create our servant object, register it with the POA and activate it.
ResultServiceServant servant = new ResultServiceServant();
rootPOA.activate_object(servant);

// Print the IOR of our servant
System.out.println("Servant IOR: "+
    orb.object_to_string(servant._this_object()) + "");

// Activate the POA, enabling it to serve client requests.
rootPOA.the_POAManager().activate();

// Wait for client invocations
orb.run();
```
Example: Corba ResultClient

```java
String resultServiceIOR = args[0];
Result result = new Result();
result.match = args[1];
result.score = args[2];

// Initialize the ORB
ORB orb = ORB.init(args, null);

// Get remote object reference from IOR
org.omg.CORBA.Object objRef = orb.string_to_object(resultServiceIOR);

// Downcast CORBA object to its appropriate type
ResultService resultService = ResultServiceHelper.narrow(objRef);

// Remote method invocation
resultService.reportResult(result);
```
Remote Procedure Call (RPC)

- Inter-process communication over networks
- Allows processes to call procedures in other processes
- Fore-runner to RMI (introduced in RFC 707 anno 1976)
- Served as basis for the communication model in the *Distributed Component Object Model (DCOM)*
Remote Procedure Call (RPC)
Sun RPC

- Created for client-server communication abstraction in the Sun Network File System (NFS)
- Alternative name: Open Network Computing (ONC) RPC
- The most common library for RPC, see
  - Brown. *Unix Distributed Programming*. Prentice Hall
  - RFC 1831
  - man rpc
- Supported by most UNIX operating systems
- Multiple language ties
  - C, Perl, Java, etc
External Data Representation (XDR)

- **Sun XDR**
  - originally a standard that defined external data representations for primitive and aggregated types in NFS
  - uses implicit typing (protocol context determines type)
  - extended to an IDL

- **Sun RPC interface**
  - program number + version number used as interface id
  - procedure declarations with associated types
  - procedure signature + number used as procedure id
  - single input and return parameters
• `rpcgen` - interface compiler that generates
  - client stubs
  - server main, stubs and dispatcher
  - XDR marshalling routines
  - header files for (declared) types
Binding Service

- **Port mapper**
  - runs locally on all hosts
  - maps services to ports
- Services register with port mapper, specifying
  - program number
  - version
  - port
- Clients resolve server port using
  (program number + version) tuples
Example: fileserver.x

```c
const MAX_BLOCK_SIZE = 1000;

struct Block {
    int length;
    char buffer[MAX_BLOCK_SIZE];
};

struct readargs {
    string name<>;
    int block_offset;
    int block_length;
};

program FILESERVER {
    version VERSION {
        Block READBLOCK(readargs)=1;
    } = 1;
    } = 9999;
```
Example: readblock.c

```c
Block *readblock_1 (readargs *args)
{
    static Block block;
    int fd = open(args->name, O_RDONLY);
    lseek(fd, args->block_offset, SEEK_SET);
    block.length = read(fd, block.buffer, args->block_length);
    return &block;
}
```
Example: fileclient.c

```c
main (int argc, char ** argv)
{
    CLIENT *clientHandle;
    char *serverName;
    char *filePath;
    readargs readArgs;
    Block *data;
    serverName = argv[1];
    readArgs.name = argv[2];
    readArgs.block_offset = atoi(argv[3]);
    readArgs.block_length = atoi(argv[4]);
    /* creates socket and a client handle */
    clientHandle = clnt_create(serverName, FILESERVER, VERSION, "udp");
    /* call remote procedure */
    data = readblock_1(&readArgs, clientHandle);
    data->buffer[1+data->length] = '\0';
    printf("Read block:\n%s\n", data->buffer);
    clnt_destroy(clientHandle); /* closes socket */
}
```
## Summary

<table>
<thead>
<tr>
<th></th>
<th>Java RMI</th>
<th>CORBA</th>
<th>Sun RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multilingual</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>External Data Representation</strong></td>
<td>Object Serialization</td>
<td>CDR</td>
<td>Sun XDR</td>
</tr>
<tr>
<td><strong>Data format</strong></td>
<td>Binary</td>
<td>Binary</td>
<td>Binary</td>
</tr>
<tr>
<td><strong>IDL</strong></td>
<td>Java Interfaces</td>
<td>CORBA IDL</td>
<td>Sub XDR</td>
</tr>
<tr>
<td><strong>Type Support</strong></td>
<td>Objects</td>
<td>Primitive &amp; Aggregated</td>
<td>Primitive &amp; Aggregated</td>
</tr>
<tr>
<td><strong>Distributed Garbage Collection</strong></td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Binder</strong></td>
<td>RMIRegistry</td>
<td>CORBA Naming Service</td>
<td>Port Mapper</td>
</tr>
<tr>
<td><strong>Bootstrapping</strong></td>
<td>Registry Look-Ups</td>
<td>IOR / Registry Look-Ups</td>
<td>Registry Look-Ups</td>
</tr>
<tr>
<td><strong>Call Semantics</strong></td>
<td>At-most-once</td>
<td>At-most-once / Maybe</td>
<td>At-most-once</td>
</tr>
</tbody>
</table>
Today
Middlewares
Remote Method Invocation
Java RMI
Corba
Remote Procedure Call
Sun RPC

Next Time

- SOA and Web Services