Socket Programming

Proprietary distributed application were initially programmed with sockets (IP and port number). While the host name could be resolved by DNS, the port numbers were hard-wired, restricted flexibility.

Features of Remote Execution

- Parts of the application's code will be executed on a remote computer.

Ideally the code distribution should be transparent – it shouldn't matter where the code is, or which code is distributed(?). Remote code is called as it would be if it were running on the same host.

There are some benefits:

- better use of resources
- load balancing
- if the network's good enough, improved execution time

There are some problems:

- address space
  - separate address spaces on the remote computer and the local one
- heterogeneity of computing platforms
- networks can introduce errors and delays

The transparency requirement raises the question: "How can we move from a simple model (e.g. single memory space) to a more complicated model?"

We want to replace IPC and procedure calls with network communication – how do we do this translation?

Implementation

- Use a client-server model.

To keep the network communication transparent, (e.g.) component A would make calls into a proxy for component B, and vice-versa. These proxies handle the network communication, and the details are hidden from the components.

These proxies are the middleware.

Example: RPC

- Remote execution is not a new idea.

A port mapper service runs on a well-known port number, and applications can address this to get the ports for different services.

- Port mapper doesn't run on the remote machine – there's one on every host that needs to communicate with remote hosts.

Note: no TCP

Example: RMI (Remote Method Invocation)

Servers register services with a lookup service called rmi registry – clients look up the rmi registry for a remote object reference.

- Uses the JVM

Remote object defines methods, registers itself with the rmi registry. Clients send lookup requests to rmi registry to find the remote object, and then call methods on it.

RMI has some benefits:

- can change where things are running
- can hold more information about the service for lookup reasons

Unlike the port mapper, the rmi registry is on an external host.

Uses TCP for reliability, unlike RCP.

RMI Components

There's a skeleton object and a stub object – the skeleton runs on the server and receives RMI requests, sending them on to an appropriate service. (The stub receives the requests from the server?)

These are created by an RMI compiler.

Dormant Service

What if there's no client? The server is still running with no requests coming in, which wastes resources.

Potential solution: Activatable remote services.

We want services to be known about and have the potential to be activated when needed.

- Have a separate registering service which registers inactive services in the directory.

    - Need data: stub proxy to calling to (and reference), as well as an instance identifier to talk to a specific instance of the service (as there may be multiple instances running on different computers)
    - Also need a faulting reference for the service (it's a reference that can't be used yet because the service isn't running, but can be used to activate the service – the faulting reference lookup leads to the daemon (see below), which activates the service)
    - The client receives the faulting reference from the registry, and calling this leads the daemon to activate the service
    - Faulting reference hides the rmi server and rmi daemon from the client
- Have an activation service (rmi daemon – running all the time), which activates the requested service if it is dormant

Once services have been activated, we can group them by their features, etc. Also possibly for load balancing.

Stateful Service

Having a stateful service with respect to the client requires more resources (storage etc.).

Registration of Activatable Objects

A group would specify some features, and all members of the group would have those features. A group would have a group ID.

Each service would also have a service ID.

When you create an activatable service, you first have to create the group object, then you can create the service object.

Faulting Reference

Has an activation identifier, which contains enough info to get the activation server to activate the object. E.g. the activation server needs to know the group ID and the service ID.

Then has a transient reference which is the live reference to the active remote object that can be used to invoke its methods. It is specific to this client and this request.

Activation System

All requests go through the activation monitor which keeps track of which services are active. It activates the services if they need to be activated.

The activation group contains the activatable objects (services). The group is created when the first service is activated.

Summary

Benefits of using several computers:

- can use more resources
- can have clients in different places

Transparency is essential – the client should not be aware of the distribution of the application.