Evolution of Service Oriented Architectures

Rushikesh K. Joshi

Department of Computer Science & Engineering
Indian Institute of Technology Bombay

Email: rkj@cse.iitb.ac.in
The Plan

What changes have taken place in typical application architectures?

What is service orientation?

Where did it originate?

How did it evolve?

How does it connect to web services?

What new technologies are needed?
Architecture of Your Data-structures Assignment

- A single process
- One main
- Manipulate data structure
- Data is live till execution
- Next run is fresh execution
A Database Assignment

- Create Tables
- Manipulate them
- Use a query language
- Data is live across queries
Processes + Data + Interfaces

• Three tiered application
  – GUI
  – Business Logic
  – Database

A Single user application based on persistence
Processes + Data + Interfaces + Concurrency

- A GUI
- Tasks: Concurrent Activities
- Shared Database
- Concurrency Control

<table>
<thead>
<tr>
<th>UI</th>
<th>T1</th>
<th>T2</th>
<th>T..</th>
<th>Tn</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A More complex Single user application using persistence
Processes + Data + Interfaces + Parallelism → Concurrency

• A GUI
• Tasks: Concurrent and parallel Activities
• Shared Database
• Distribution of control
• Concurrency Control
• Inter-task communication

A More complex Single user parallel application using persistence
Add more user roles

• Views
• Viewers are located at different places
Add more data sites

- Views
- Viewers are located at different places
- Data is distributed
- Tasks are parallel
Add control replication

• Views
• Viewers are located at different places
• Data is distributed
• Tasks are parallel
  – Possible Control replication
Add data replication

• Views
• Viewers are located at different places
• Data is distributed
  – And replicated
    • Multiple copy consistency
• Tasks are parallel
  – Control replication
Links may fail!

- Alternate routes?
Tasks may fail!

- Failure Semantics
- Recovery possible?
Users are scattered over the internet
Some users are local
You need security policies
All sites have all the layers and they collaborate.
Types of Applications

- Compute centric, parallel
- Data centric, huge data
- Computations and data centric
- Long vs. short transactions
- Distributed and Networked
- High volume, large users
- Event driven
- Peer-to-peer, client-server, service oriented
- Multi-concern
Service orientation

What is a service?

Service provider
and the service user

Service contracts
Origins of service orientation

- Functions and procedures
- Input parameters --> output results
- What is important is an interaction between the client and the server
- Call to the service
How do you express a service?

• Syntactic Contracts
• Behavioral Contracts
  – Functional behavior
  – Non-functional behavior
From local call to Remote Procedure Call

- The procedure may be located out of the calling process
- Procedure may also be located at a remote machine
RPC (early 80s)

- A very primitive service paradigm in multiprogramming systems
  - e.g. unix servers
  - Rup
  - Rusers
  - Fingerd

Machines and processes
Issues considered in RPC

• How do you describe RPC?
• What's the process of development?
• How do you find, bind and use?
• What are the non-functional semantics such as fault tolerance semantics?
Technologies for RPC

- Description: e.g. XDR format
- Marshalling and De-marshal-ling
- Specification Language
- Stub generator
- Portmapper (directory service)
DLL Technologies

- Dynamically linked libraries
- A function or procedure may be loaded and dynamically linked
- Available on the same machine
Remote Method Invocations on Objects

• The Proxy Pattern
Technologies for RMI

• Object specification
• Server development, Proxy generation
• Object discovery
• Object binding and use
Middleware Technologies for Interoperable Networked Services

• Typical of Applications in Lan
• Hide and Provide Principle
A View of CORBA-based Middleware

Distributed Objects

Middleware

Distributed Objects

SYSTEMS (os/n/w)
Concerns addressed by Middleware

- Connectivity and Communication
- Interoperability
- Repositories
- Activation
- Services
- Service Descriptions and discovery Mechanisms
- Development Process and Tools
- Standardization
Model Driven Approach

Model first

Then generate Implementation

Existence of Middleware does not make an exception to this.
Interfaces

Interface I {
    typeR func f (type1, type2 ..)
    ...
}

Interface description languages: vendor specific
Implementation in an object paradigm

Skeleton-Implementation inherits I {
...
}

Actual-Implementation inherits Skeleton-Implementation {
....implement here...
}

Register your implementation

Create an instance and register

- obj = new Actual

- register it with a registry

- start accepting requests
Alternative Registration Mechanism

Do not Create instance directly

Register only the implementation

The middleware will create an instance when needed
Find, Bind and Invoke

• Find your component
• Create a placeholder for it
• Invoke as if it is a local component

Proxy Design Pattern
Dynamic Invocation

• Unlike procedure/function calls in your day to day programming

• Make a request as a message and send it to remote component
One way calls and callbacks

• One way calls do not block the caller

• Callbacks from server needs that the caller supports an exportable interface

![Diagram showing one way calls and callbacks]
Multiple interfaces sharing an implementation

- Different users may use different interfaces
- Roles can be associated with interfaces
- Single implementation realizes all
- Interface navigability
Generic Services

• Type independent services
• Based on object orientation (polymorphism)
• E.g. Naming: name → object mappings
• E.g. events: publisher-subscriber
  – Events of any kind
Component Reuse: e.g. Containment in COM/DCOM

Interface A

Interface IUnknown

IUnknown

X

Y

contained

container
Component Reuse: Aggregation in COM/DCOM
Creation through Factory Objects

- First find and bind to factory
- Factory creates objects
- DLL based factories

Client | COM Library | DLL | component instance
---|---|---|---
CoGetClassObject | DllGetClassObject | create factory instance | return factory instance
return factory instance CreateInstance | return factory instance | create component instance
use component | return component instance | create

R.K.Joshi
Creation through Factory Objects

- First find and bind to factory
- Factory creates objects
- DLL based factories

<table>
<thead>
<tr>
<th>Client</th>
<th>COM Library</th>
<th>DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoGetClassObject</td>
<td>DllGetClassObject</td>
<td>function instance</td>
</tr>
<tr>
<td>return factory instance</td>
<td>return factory instance</td>
<td>create</td>
</tr>
<tr>
<td>CreateInstance</td>
<td>create</td>
<td>create</td>
</tr>
<tr>
<td>use component</td>
<td>return component instance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IIT Bombay</td>
</tr>
</tbody>
</table>
Web Services

• A service is available through http-based protocols.
• You have to cross the firewalls
• A different set of technologies, but the issues are the same
• You need to rework out all the issues of communication, typing, discovery, descriptions, security
Example Web Service Standards and Technologies

- WSDL – web service description language
- SOAP – Simple object access protocol for exchanging XML based messages
- UDDI – universal description discovery and integration
Some examples of service orientation

• Telephonic services
• Web Services
• Services for mobile devices
• Peer to peer applications
• Aggregates and Compositions
Web Server: A case study (program developed by 2 students)

Handover to applications

process some requests e.g. home pages

HTTP Requests
First, a Simple Socket Server
An example code

- `simple_server_main.cpp`
A simple webserver code
how does it look like?

web server listens on port xxxx

client->webserver->local
server i.e. http/perl/java..
Implementation

- instances of local server as threads
- Another web server thread listens to incoming requests
- request type determined and the request dispatched
- communication between main thread and local threads through pipes, one per local thread
CLASS: CWebServer

CWebServer()
Input: int port, int p_http, int p_java, int p_pl
creates an instance of CWebServer. (this is the constructor)

Service_function()
Input: CWebServer * _webserver Output: nothing
The main service loop which infinitely listens to the port for incoming requests and then classifies them and dispatches them to the appropriate LocalServer thread.

GetRequestType()
Input: std::string line Output: int request_type
Classifies the incoming request as one of the following HTTP_REQ, JAVA_REQ, PL_REQ, BAD_REQ which then helps the service loop to decide which thread to send the request to.
**CLASS : LocalServer**

this is an abstract class which only provides the interface to the CWEBServer object to call the functions on the appropriate LocalServer objects

CreateInstance()
Input : int pipe_d
Output : static LocalServer * Â
This is the function which emulates the SINGLETON pattern to ensure that only one instance of each of the LocalServers is created.

HandleRequest()
Input : int sock, string & strFirstLine
This function is supported by all the LocalServer objects. This is the function called by the CwebServer object to dispatch the request. Basically this reads the header and then determines the actions to be taken and then returns the file back to client.
CLASS : CHTTP

CLASS: CHTTP

CHTTP

Input:int pipe_d

Description: This is the constructor which creates a pipe for this object which communicates with the main service loop which diverts the appropriate incoming request to this object.

GetHtml ()

Input : http_request * r

Description : This function searches for the requested file and then sends it to client with the help of LoadFile function.

LoadFile ()

Input : string & strFilePath, string & strFileContents

Description: This function when called by the GetHtml, gets the contents of the requested file and then sends it to the client (bypassing the CwebServer object).

R.K.Joshi

IIT Bombay
CLASS : CPerl

CPerl
Input: int pipe_d
Description: This is the constructor which creates a pipe for this object which communicates with the main service loop which diverts the appropriate incoming request to this object.

GetPerl ()
Input: perl_request * r
Description: This function searches for the requested file and then sends it to client with the help of LoadFile function.

LoadFile ()
Input: string & strFilePath, string & strFileContents
Description: This function when called by the GetPerl, gets the contents of the requested file and then sends it to the client (bypassing the CwebServer object).
CLASS : CJava

CJava
Input: int pipe_d

Description: This is the constructor which creates a pipe for this object which communicates with the main service loop which diverts the appropriate incoming request to this object.

GetJava()
Input: java_request * r

Description: This function searches for the requested file and then sends it to client with the help of LoadFile function.

LoadFile()
Input: string & strFilePath, string & strFileContents

Description: This function when called by the GetJava, gets the contents of the requested file and then sends it to the client (bypassing the CwebServer object).
Natural Extensions

How are logins and permissions and communication security to be handled?

How should requests be logged?

What support is assumed from the browser at client end?

What support is available for application architectures?

Grids at the backend?

How are services themselves described?

What formats are used to exchange data, requests, results?

More meaningful or semantically rich applications designed as services
Once again, what's a service?

Agreed protocol between client and a server

Web services are provided on the web

Servers may use sophisticated available technologies, middleware, parallelism, shared spaces, shared resources

At client side, some support is provided by the browser, but stand-alone applications can also exist
So Explore and Enjoy your workshop!