Institute of
Public Administration

COMP41530 - Web Services in Cloud Computing

Barry Corish
Associate Lecturer, IPA

Lecture 02

Institute of Public Administration | 57-61 Lansdowne Road | Dublin 4 | Ireland | Ph. +353 1 2403600 | www.ipa.ie

Overview

- Review of last week
- Distributed Computing
- Client Server Model
- Internet Protocols and WebServices
- Messaging
- Middleware

Overview

- **Review of last week**
- Distributed Computing
- Client Server Model
- Internet Protocols and WebServices
- Messaging
- Middleware

Review: Problems with large Information Systems

- Information systems tend to become:
  - larger, more complicated
  - harder to change or replace
  - more expensive

Review: Change

- Organisational change requires
  - More interconnection
  - More change
  - Faster
- Traditional systems are failing to keep up, and are costing too much

Review: Connections

- Inter System Connections tend to be:
  - Complicated
  - Unreliable
  - Expensive
  - “Tightly Coupled”

Review: Service Oriented Architecture

- A way of designing systems
- Break the system down into small functions (services)
- Make the services:
  - Modular
  - Interoperable
  - Re-usable
  - Flexible

Review: WebServices

- Software modules
- Discreet functions
- Accessed by other Systems
- Defined external interfaces – “Black Boxes”
- Follow standards for their external interfaces
- Follow standard ways of communicating, based on Internet and Web technologies and standards

Review: Concepts

- Simple v. Complex
- Stateless v. Stateful
- Synchronous v. Asynchronous
- Tightly Coupled v. Loosely Coupled
- Orchestration

Overview

- Review of last week
- **Distributed Computing**
- Client Server Model
- Internet Protocols and WebServices
- Messaging
- Middleware

What is a distributed computing system? (1/2)

- An information system
- Composed of several operational components
- Each components supports one or more function
- Functions must co-operate to complete business process
- To co-operate, they need to share information to each other

What is a distributed computing system? (2/2)

- Typically, the functions are running in several components, and the components are spread over several computers
- Components can be distributed across a LAN (e.g. 5-500 metres)
- Can be spread across a WAN (opposite sides of the world?)
- But multiple functions and/or components can also be within the same computer
- The components communicate by sending messages over a network
- That the functions are spread over multiple components should be transparent to the user, and ideally, to the developer

Overview

- Review of last week
- Distributed Computing
- **Client Server Model**
- Internet Protocols and WebServices
- Messaging
- Middleware

Client Server Model

- Most distributed information systems follow the Client Server Model
- There are alternatives for specialised applications
- Everything we'll be doing assumes Client/Server

Server

- Program sits running on a system
  - Runs “all” the time
  - Listens, waiting for a message to arrive
- On receipt of message, it does something
  - Process, store, decide, pass on, etc.
- Often sends a response (not always!)

Client

- Runs intermittently
  - When there is something to do
- Locates a server
- Send a message
- If appropriate, waits for response
- Deals with any response

Which is which? (1/2)

- In a distributed system, two communicating components are typically not peers
- One is "server", one is "client"
- Normally obvious which should be which
  - "One" vs "Many"
  - One Server, many clients

Which is which? (2/2)

- If it's not obvious which component should be which:
  - Base on "who knows when something needs to be done"
    - The one who knows is the client
- If still not obvious:
  - either choose one arbitrarily
  - or, if appropriate, make each computer both a client and a server

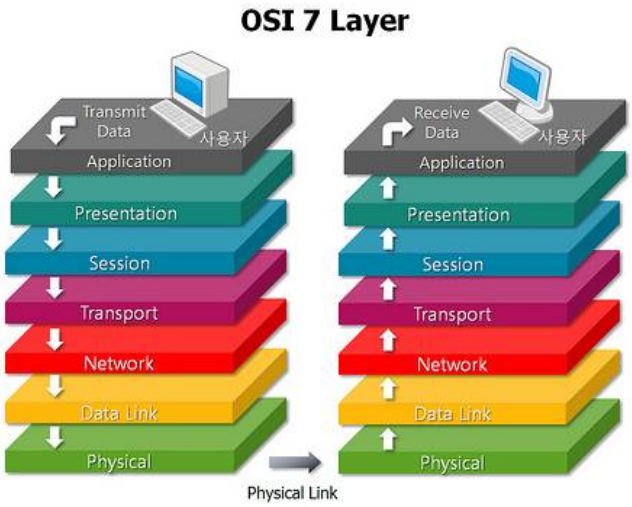
Overview

- Review of last week
- Distributed Computing
- Client Server Model
- **Internet Protocols** and WebServices
- Messaging
- Middleware

OSI Reference Model

- Thinking about how components communicate:
- Open Systems Interconnection Reference Model
- A framework for how to think about communication between components
- 7 layer conceptual model
  - Possible to have fully functional systems without all 7 layers
  - Sometimes not obvious which layer a given technology lives in
- A good way to think about connections

OSI 7 Layer Model



OSI Reference Model (1/2)

- Host Layers (4-7):

Layer	Name	Description
7	Application (Support)	Supports Application and End User Processes
6	Presentation	Translates application to network formats, incl. Encrypt & Decrypt
5	Session	Create, manage and terminate connections
4	Transport	Flow control, error recovery

OSI Reference Model (2/2)

- Media Layers (1-3):

Layer	Name	Description
3	Network	Switching, routing, formatting
2	Data Link	Packets – sets of bits
1	Physical	Bit Stream over physical link

- Mnemonic : “All People Seem To Need Data Processing”

Internet Protocols

- Provides all the basic services to support a point-to-point web services call
- ..but in practice, we want more than the basics.

TCP/IP

- Core of the Internet
- Two protocols:
- IP - Internet Protocol
- TCP - Transport Control Protocol

IP - Internet Protocol

- Gets single packets from network point A to network point B
- Defines format of the network addresses
- Unreliable - delivery not guaranteed
- One packets may arrive multiple times, or never!
- Packets may arrive out of sequence

TCP - Transport Control Protocol

- Looks after the connection
- Get all the packets
- Get them in the right order
- Re-order any missing packets

IPv4 Addresses

- a.b.c.d
- ...where each is number 0-255
- Approx. 4.3 billion address
- Nearing exhaustion!

Domain Name System (DNS) (1/2)

- Why don't we type the IP address of a server into Browser to go there?
- We can, but they're hard to remember.
- DNS provides a giant lookup table of hostnames to IP addresses
- Consists of Top Level Domains (TLDs)
- e.g. .com, .net, .ie etc.
- Creation of TLDs administered by ICANN
- Many more TLDs "under consideration" by ICANN

Domain Name System (DNS) (2/2)

- TLDs administered by "authorities“
- Authorities delegate the provision of names under the TLD to providers
- Provides an important abstraction layer
- Can change IP address without losing "name"

"Local Addressing"

- No need for a "public" IP address for every computer in a LAN.
- Hide "lots" of computers in the LAN behind a single public address.
  - Addresses beginning 10.*.*.* or 192.168.*.* are reserved for use away from the public internet
- Likewise, normally an internal DNS within a LAN
  - Hostnames not published to the “public” DNS infrastructure

IPv6

- New addressing scheme
- 8 sets of 4 hex numbers (0-F)
- Successor to IPv4.
- 128 bits long
- 3.4×10^38 addresses
- Lots!

IPv4 to IPv6 changeover (1/2)

- Change over beginning
- Difficult - every step must be IPv6 enabled for end to end communication
- Likely that LANs will remain IPv4 for some time

IPv4 to IPv6 changeover (2/2)

- Edge components where LAN connects outwards to Public Internet will do translation from IPv4 to IPv6
- Public Internet Infrastructure will gradually shift to dual IPv4 and IPv6
- Long term, IPv4 may drift out of use in Public Infrastructure

Ports

- So that we can have many servers running on a single server, each program uses a different port.
- Only one program can use (“bind”) to a port on a given system at any one time
- IP Address - phone number for the building (which computer?)
- Port Number - extension number within the building (which service within the computer?)

Services on Ports (1/2)

- Ports under 1024 are generally used for "well defined" services.
  - Telnet - 23
  - FTP - 21
  - SMTP - 25
  - HTTP - 80
  - LDAP - 389
- Not always:
  - Microsoft SQL Server - 1433
  - IBM DB2 server - 50,000

Connections between Components

- When a client calls a service:
- Locates the service (DNS)
- Selects a free “high number” port on itself.
- Calls from <client ip address>:<free port number>
- Calls to <server ip address>:<defined port number>
- Response from server is sent back to <client ip address>:<free port number> , where client program is waiting

Overview

- Review of last week
- Distributed Computing
- Client Server Model
- **Internet Protocols and WebServices**
- Messaging
- Middleware

Why "Web"Services? (1/3)

- Why use the Internet/Web stack of protocols and technologies for inter – component messaging?
  - Availability and near universal implementation
  - Vendor neutral
  - Simple
  - Reliable
  - Cheap
  - Fast (enough!)
  - Well understood

Why "Web"Services? (2/3)

- More…
  - Logging
  - Resilient
  - Flexible
  - Hidden (we work at the top of the stack only!)

Why "Web"Services? (3/3)

- Tools available:
  - Firewalls
  - Load Balancers
  - IDS
  - IPS
  - Traffic monitoring/analysis
  - Encryption
- Most problems already solved for Websites
- Reuse the best for WebServices
  - This exists, and it works.
  - Don't re-invent the wheel.

Overview

- Review of last week
- Distributed Computing
- Client Server Model
- Internet Protocols and WebServices
- **Messaging**
- Middleware

Messaging

- Information is sent between Components as messages.
- Messages consist of:
  - Header - Addressing, Message Type etc.
  - Body - the actual data
- NB: Distinction in book between Properties and Body

Synchronous Messaging (1/2)

- Send the message, stop processing until I get an answer
- e.g. RPC - Remote Procedure call
- The next function in the process is located in another component of the distributed system
- This is normally transparent to the code before and after the call

Synchronous Messaging (2/2)

- The call is the same as if the function was located on this component
- Tends to be tightly coupled
- Example: Java RMI (remote method invocation)

Asynchronous Messaging (1/2)

- "Fire and Forget" the message
- AKA "store and forward"
- Generally based on a Message Queue
- Put the message on the queue
- Once on queue, forget about it - not your problem any more

Asynchronous Messaging (2/2)

- Queue stores the message, looks after routing, delivery etc.
- Gets the message there eventually
- But, if you expect an answer, or ack. - must watch the queue.
- Tends to be more loosely coupled
- Examples: IBM MQ, Tibco, Mule

Overview

- Review of last week
- Distributed Computing
- Client Server Model
- Internet Protocols and WebServices
- Messaging
- **Middleware**

How to avoid rebuilding chaos in SOA?

- Are we just replacing existing connections one-for-one with new connections based on Web technologies?
- What prevents the complexity from creeping in again?

What is Middleware? (1/2)

- Generally more software components
  - Might be implemented as a hardware appliance
- Acts as "plumbing" and "glue"
- Sits between the components of a distributed system
- Provides pre-written services connected with communication

What is Middleware? (2/2)

- Not part of the business process in itself
- A set of tools to make connecting/running the business software easier
- Does the "donkey work“
- Think: If business software is “trucks and trains”, middleware software is “roads and train tracks”

What basic services can Middleware provide?

- Location
- Abstraction
- Reliability
- Fail Over
- Load balancing
- Scalability
- Authentication
- Authorisation

Middleware provides services we need...

- ..so we don't have to write them ourselves.
- Less handwritten code is better.
- If you find yourself writing code to provide any of these services, you may be doing:
  - the wrong things
  - the right things, but wrong way

Connecting systems is all about messages, right?

- We are passing chunks of information between systems.
- Each time information is sent, that's a message
- Both sender and receiver must "understand" the message, and interpret it's meaning in the same way:
  - In terms of how it is written (syntax)
  - In terms of what it means (content with the syntax)

Implementing Messaging at Enterprise Level

- To integrate the systems in an Enterprise (or between Enterprises):
  - Choose a standard for messages
  - Implement a system for processing the messages
  - Using bought-in middleware, unless you like pain…

Enterprise Messaging System (EMS) (1/3)

- Enterprise Wide Standards for messages
  - Careful definition of messages
  - Standards, not implementation!
- There should only be one standard within the organisation
  - More normally, there are several
  - These are normally hard to map to each other

Special Middleware 1: Message Oriented Middleware - "MOM“ (1/2)

- Sits in between the client and the server
  - Client never directly contacts the server, and vice versa
  - Normally Asynchronous messaging
  - Provides additional services:
    - Understands internals structure of the message
    - Can make decisions based on any aspect of the message
    - Load balancing/failover
    - Content based routing

Enterprise Messaging System (EMS) (2/3)

- Defines:
  - Security
  - Routing/Addressing
  - Metadata (message type, timestamps etc.)
  - Message Structure
    - Data Dictionary

Enterprise Messaging System (EMS) (3/3)

- Vital to making SOA work effectively.
- What common language are we going to use for our messages?
- Keep the standards “open” to allow for future change

Special kinds of Middleware

- So far: General purpose middleware providing general purpose services
- We also need specialist services for dealing with Messaging.
- We’ll discuss 4 kinds of message-specific middleware
  - These are loose and overlapping categories
  - Vary in how much they do to or with the message(s)

Special Middleware 1: Message Oriented Middleware - "MOM“ (2/2)

- More…
  - Message transformation
  - Message prioritisation
  - Message buffering/overflow facilities
  - Guaranteed delivery
  - Scalability
  - Publish & Subscribe
  - Message duplication to multiple recipients

Special Middleware 2: Integration Broker

- High-end "MOM" = "Integration Broker"
- Add in business rules engine
- Generally more powerful/faster
  - Often running on dedicated appliance (hardware)
  - Can handle higher rates of transactions (100s / second)
- More flexibility

Special Middleware 3: Enterprise Service Bus (ESB) (1/2)

- Messaging Backbone
- Implementation of the standards defined in the EMS
- "All" systems in the enterprise can connect to send/receive messages

Special Middleware 3: Enterprise Service Bus (ESB) (2/2)

- Each system connects once to the bus
  - Reduced number of connections
- Looks after routing of messages
- Can transform, translate and re-map messages as required

Special Middleware 4: Transaction Oriented Middleware (1/2)

- Moving beyond single transactions:
  - Transaction Processing ("TP") Systems
  - Monitor the progress of a business function across multiple components
  - Allows rollback of failed/abandoned transactions
  - Allows restart of failed/timed out transactions

Special Middleware 4: Transaction Oriented Middleware (2/2)

- Assures ACID principles on a distributed system:
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Otherwise, these are very hard to implement on distributed systems

Review

- Distributed Computing
- Client Server Model
- Internet Protocols and WebServices
- Messaging
- Middleware

Questions?

?