LECTURE 6:
ENTERPRISE APPLICATION INTEGRATION (EAI), SERVICE-ORIENTED ARCHITECTURE (SOA) & MIDDLEWARE IN ENTERPRISE ARCHITECTURE

Lecture Contents

• Intro to SOA & Middleware in Enterprise Information Systems
  – Need for EAI, Middleware & SOA in WF currently with business trends
  – EAI & Middleware and where this fits in to Business Processes
  – Evolution from EAI to SOA
  – A first mention of Web Services & their role
  – Some theory of SOA with an example of its application

• Types of communication in EIS

• Types of Distributed Communications

• Message-Oriented Middleware: IBM MQ Systems, RabbitMQ

• Middleware Background: Apache Web Servers, Server Clusters & Names

• RESTful Web Services
SECTION 6.1: INTRODUCTION TO SOA & MIDDLEWARE IN ENTERPRISE INFORMATION SYSTEMS

Recap on Workflow

- Workflow (definition from WorkFlow Management Coalition):
  - “The computerised facilitation/automation of a BP, in whole or part”
  - Workflow technology is often an appropriate solution to BPR activities.
  - Traditionally managed by s/w (Workflow Management Systems WFMS)

- Thus workflows involve the coordinated execution of multiple ‘tasks’/’activities’
  - These are performed by different processing entities, nowadays mostly in distributed heterogeneous environments
  - These are very common in enterprises of even moderate complexity
  - Can be performed by humans or machines...

- A workflow system can be defined as a collection of processing steps organized to accomplish some BP
Changes in Context: (cont’d)
Recent Broad Goals/Trends

Goals:
- Low cost
- Streamlined & efficient process
- Monitor & track process execution
- Detect and manage exception
- In-time response, etc
- Solution: IT

Business Trends
- Scalewise:
  - Intra-Enterprise
  - Inter-Enterprise
  - Global Interaction

IT Trends
- Mainframe
- Set of Servers
- Set of Services

Business Trends
- Timewise
  - Manual
  - Electronic
  - Web

The Lie of the Land...

- The diagram shows a layer-wise outline of some of the technologies that will be examined in this lecture & how they relate to each other.
- Up to now, only three layers have been considered (mostly BP layer)
### The Changing Context: Terminology

- Integrating enterprises' existing IS applications to run BPs with many s/w systems has used **Enterprise Application Integration (EAI) technology**:
  - User Interface Integration,
  - Data Integration
  - Method or Function Integration
  - Business Process Integration

- **Middleware** is communication facilitator in EAI and this is handled by the **Enterprise Service Bus (ESB)** (akin to a message router in EAI).

### The Changing Context(/2): EAI & Middleware

- Where in multi-layer architecture should the business-logic be put?
  - Can’t put in client (UI) tier
    - Leads to Fat client, reimplemented for each different client type
    - Redistributing clients after each software update
  - Not Data tier as different applications have different uses for same data
  - Has to reside on Middle Layer

- **Enterprise Application Integration (EAI)**
  - Integrates applications and enterprise data sources so that they can easily share business processes and data
  - Integration done without much changes of applications/ data sources
  - All data conversion, security, comms between computers is seamless
The Changing Context (/3): Challenges to BPM

- Methods of Business Process Management are useful when optimising BPs within an enterprise.

- Some business environments require many different process designs
  - BP Mass-customization => Automatic BP creation (e.g., patient health records)
  - BPs evolve dynamically as they execute, through the exchange of information among participants whose relationships evolve as a result

- But BPM is neither scalable nor adaptive by nature
  - So can use a framework based only on BPM to build business applications but too tightly coupled to adapt to future changes.
  - For each change, business dept must interact with IT dept to change software.

- Still need BPM as processes will need to be optimised (Bajwa et al. 2008)
- Need increased agility in BPs for loosely-coupled business networks

Typical status quo in Many Enterprise IT Architectures

- Functional and technical application monoliths ubiquitous
  - Stovepipe architectures, application scope creep, redundant implementations, data management and many other agility issues
  - Architectural governance or guidance missing

- Development and integration projects costly and long running
  - Proprietary point-to-point connections, often developed from scratch
  - File transfer is a frequently used integration pattern with numerous architectural drawbacks
  - ‘Roll-your-own’ philosophy works short term, but leads to maintenance headaches eventually

- As a result, horizontal initiatives are much harder to implement than they have to be
  - Example: single customer relationship management solution on top of several line-of-business applications (packages and custom developed)
What is Service Oriented Architecture (SOA)?

- Data & BP sharing between applications are EAI’s primary purposes.
  - Links enterprise applications to talk to one another & do “batch” data transfers
  - But EAI also defines principles for linking multiple systems, such as message-oriented middleware (MOM), of which more later.
  - EAI is maybe old with SOA, but still EAI tools useful for large scale integrations
- SOA provides ‘transactional’ data transfer, needs no third-party s/w:
  - It differs from EAI in that it does not depend on a third-party solution.
  - Links interacting & contracted services via comms protocol (i.e. Web Services)
- Services are useful because they:
  - Are reusable in heterogeneous environments at multiple levels, including code, platform, so more flexible in the design of enterprise applications
  - Are implemented by 1/more code components in homogeneous environments
  - Aggregating 1/more components into a service, accessible through asynchronous messaging using open standards.

Different parties (even in the same company) may have different:

- OS, interface, data format, infrastructure, interaction protocols, language, etc
- Automating Supply Chain Mgmt implies bringing all of these together
- As seen, EAI currently solves this but evolution towards SOA
  - Supports flexible s/w dvpt thro ‘loose service coupling’ => no need to talk to IT.
SOA Fundamentals: Modularity, Layering & Loose Coupling

- Multi-step/-user claim handling BP in IBM WebSphere should reuse existing app functionality
- How to integrate new BP with three legacy apps in a flexible, secure & reliable way?
- Existing System Apps: SAP R/3, MS Visual Basic & COBOL for CRM, fraud checking, payments calculations

SOA Principle 1: Modularity (i.e. Separation of Concerns)

- Motivation:
  - Integrating monolithic applications ("stovepipes") is hard (e.g., traditional Enterprise Resource Planning packages)

- Solution
  - Refactor to services, expose service interface only, hide implementation details (a.k.a. encapsulation)

- Consequences
  - Service contracts have to be defined and interpreted (by tools and/or at runtime)
  - Services have to be located and invoked in a coordinated manner
  - No undesired side effects (data mgmt?) service invocations
SOA principle 2: Layering (logical and/or physical)

- **Motivation**
  - Service characteristics such as interface granularity & lifecycle vary: e.g. technical logging service vs. claim checking business process

- **Solution**
  - Organize SOA into 3++ architectural layers

- **Consequences**
  - More abstraction (i.e. services can be composed out of other services leading to composite applications),
  - Requires comms infrastructure (usually Web Services, messaging)
  - 1st law of distribution: “best remote call is the one you don’t make”
  - Remote calls can block waiting for answer!

---

SOA principle 3: Loose coupling through messaging

- **Motivation**
  - Once applications are modularized, dependencies between services occur

- **Solution**
  - Couple services loosely (several dimensions: location-, time-, invocation context-independent)
  - e.g. messaging system decouples in time, location, language dimensions
  - Messaging via any MOM system (eg. HTTP, IBM MQSeries, RabbitMQ)
  - These are a flavour of transport used in an application built to SOA principles,
  - Any transport that can send/receive bags of bytes with your messages will work

- **Consequences**
  - Messaging means single implementation by default (no remote objects)
  - Asynchronous communication complicates systems management
  - e.g. Reliability:- is there any answer coming from the receiver at all???
SOA principle n: Service virtualization and flexible infrastructure

• Motivation
  – “I don’t care about one provider, just pick any that currently is best for me”

• Solution
  – WWW to service bus/cloud
  – Two-level programming

• Consequences
  – Many open issues e.g., trust and privacy, precise semantics, QoS

• Isolated business steps
  – … as a Service (XAAS), e.g. Salesforce.com CRM & Amazon Storage
  – Dynamic matchmaking of services on demand

SOA in Practice: Use of Web Services

• Web Services: the evolution of m/w, EAI platforms to build apps
• WS is server that awaits/replies to REST commands via HTTP
• Web Services is a model for using the Web:
  – To automatically initiate processes via the Web using programs
  – To describe, publish & initiate processes dynamically in distributed envt
• Benefits of Web Services Include:
  – Providing interoperability among different platforms
  – Existing applications can be wrapped as Web services
  – Client & Service can use different platforms & programming languages
• Web services encapsulate business functions:
  – Routine Transactions: Check credit card, process payment, Stock quotes
  – Can compose BPs (e.g. Travel planning, Health care, Etc, etc)
• If you can imagine a way to electronically deliver something:
  – For a customer to solve a problem, or provide some service to them
  – Then you have a viable example of a Web service!
SOA in Practice: Example of SCM

Supply Chain Management
No parts at Plant? ERP system messages HQ -> queries ERP system at other plant for item. None at HQ? HQ sends e-order to supplier’s ERP system

• EA for inventory query/supply order:
  - EA needs 4 systems connected by 3 proprietary interfaces.
  - Mainframe at 1st plant connects to HQ’s Windows servers -> connects to 2nd plant’s IS & supplier Sun box.
  - As seen, this tightly coupled integration is inflexible & costly to modify/maintain.
  - E.g., in the EA, to add new suppliers, competitive bidding on supplier contracts are complex/expensive.

B2B Commerce Facilitation with SOA
• Converting to an SOA allows for B2B commerce without system reworking systems.
• As well as eliminating proprietary interfaces, SOA enables 1st plant to check directly with 2nd plant & place orders without need for HQ’s computer.
• HQ sees transactions with own WS & to-and-fro messages btw 2nd plant & supplier.
SOA in Practice: Example of SCM (/3)

- SOA increases B2B commerce by manufacturer holding competitive bidding system.
- Suppliers bidding to win business required to use WS to connect to bidding system.
- Again, can do with traditional technology but costs are so high that it’s rarely done.
- SOA allows manufacturer to manage suppliers/ costs & suppliers can get business.
- Further, when suppliers are replaced or new suppliers added, IT can now respond quickly and inexpensively to the business decisions.

Solving a SOA Problem: Microservices

- Unfortunately with Services, Goldilocks comes in again – what size?
- SOA have an enterprise scope, apps talking to one another.
- Services are exposed through standardized interfaces between apps.

- Microservices is another approach, breaking apps into smaller, completely independent parts
- This enables them to have greater agility, scalability, and availability.
- The microservices architecture has an application scope, with a focus on the structure and components within an application.
SECTION 6.2: MIDDLEWARE IN DISTRIBUTED SYSTEMS

Role of Middleware

• **Observation**
  – Role: to provide common services/protocols in Distributed Applications
  – Can be used by many different distributed applications

• **Middleware Functionality**
  – (Un)marshalling of data for transport to remote systems/apps
  – Naming protocols: to allow easy sharing, discovery of resources
  – Enforces business rules
  – Security protocols: for secure communication
  – Scaling mechanisms, such as for replication & caching (e.g. decisions on where to cache etc.)
  – Rich set of communications protocols: to allow some applications to transparently interact with others regardless of location.
Introduction to Message Passing

• **Commands in Message Passing**

  • With systems where comms are necessary, message passing is common.
  
  • There are 2 basic message passing primitives, **send** & **receive**
    
    - **send** primitive: sends a message (data) on a specified link from one process to another,
    
    - **receive** primitive: receives a message on a specified link from other processes.

• The send primitive has slightly different form

• This depends on whether message passing is **synchronous** or **asynchronous**.

---

Synchronous Message Passing

• In synchronous message passing each channel forms a direct link between two processes.

• Suppose process A is sending data to process B:
  
  - When process A executes **send** primitive it waits/blocks until process B executes its **receive** primitive.
  
  • Before the data can be transmitted both A & B must ready to participate in the exchange.

• Similarly the **receive** primitive in one process will block until the send primitive in the other process has been executed.
Asynchronous Message Passing

- In asynchronous message passing `receive` has the same meaning/behaviour as in synchronous message passing.
- The `send` primitive has different semantics.
- This time the channel between processes A & B isn’t a direct link but a message queue.
- Therefore when A sends a message to B, it is appended to the message queue associated with the asynchronous channel, and A continues.
- To receive a message from the channel, B executes a `receive` removing the message at the head of the channel's message queue and continues.
- If there is no message in the channel the receive primitive blocks until some process adds a message to the channel.

**SECTION 6.2.2: INTRODUCTION TO COMMUNICATIONS MIDDLEWARE**
A Bit About Socket Communications

• **Sockets**
  - Most network comms use the client-server or (request-response) model.
  - i.e. process (client) connects to other (server), e.g. seeking info.
  - Client needs to know of the existence of and the address of the server,
  - But server needn’t know client’s address/existence before connection.
  - After connection, both sides send & receive info.
  - Making a connection differs for C/S, but both use basic sockets.
  - A socket is one end of an interprocess communication channel.
  - Both processes establish their own socket.

• Unfortunately, they:
  - Divide the message into packets & send via various routings to receiver
  - Reliable but require receiver to be able to interpret the message
  - Can disrupt the order of the packets.

Classification of Middleware

• Classify middleware technologies into the following groups:
  1. **Bog-standard Sockets**
     - The basis of all other middleware technologies.

2. **RPC – Remote Procedure Call (more later)**
   - RPCs provide a simple way to distribute application logic on separate hosts
   - Allow one host to request a service from a host on another computer in a network without having to understand network details.

Stubs are pieces of code that can connect to other network procedures but pretend to be local procedure calls. Have to wrap/unwrap data/results.
Classification of Middleware (/2)

3. **TPM - Transaction Processing Monitors:**
   - TPMs are a special form of MW targeted at distributed transactions.

4. **DAM - Database Access Middleware:**
   - DBs can be used to share & communicate data between distributed applications.

Classification of Middleware (/3)

5. **Distributed Tuple:**
   - Distributed tuple spaces implement a distributed shared memory space.
   - In practice this works like a DB, separating ‘sender’/’receiver’ in time

6. **DOT (Distributed Object Technology):**
   - Here both sender/receiver share an object which they both operate on.
   - Example of this is Enterprise Service Bus
Classification of Middleware (/4)

7. **MOM (Message Oriented Middleware):**
   - In MOM, messages are exchanged asynchronously between distributed applications (senders and receivers).

8. **Web services:**
   - Web services provide access to services via a defined interface, typically accessible through the web protocol HTTP.

---

Summary of Communications
Middleware

- Essentially a range of types of communications middleware
- All can be used to implement others, all are suited to different cases
  - All carry some payload from one side to another (with details)
  - Some of these payloads are ‘active’ and some are ‘passive’
  - Also differ in granularities and whether synchronous or not.
SECTION 6.3: OBJECT-BASED COMMUNICATION IN DISTRIBUTED SYSTEMS

Remote Procedure Call (RPC)

- **Rationale**: Why RPC?
- **Distribution Transparency**:
  - Send/Receive don’t conceal comms at all (requester/client has to know details on the server) – need to achieve access transparency.
- **Answer: Totally New ‘Communication’ System**:
  - Allows programs to communicate by calling another program’s methods.
- **Mechanism**
  - RPC is synchronous so when a process on machine A calls a method on machine B, calling process on A is suspended and
  - Execution of the called procedure takes place on B.
  - No message passing at all is visible to the programmer.
  - Application developers familiar with simple communications model.
**Basic RPC Operation**

1. Client procedure calls client stub
2. Stub builds message, calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters, calls server.
6. Server works, returns result to stub.
7. Stub builds message, calls local OS.
8. OS sends message to client's OS.
9. Client OS gives message to client stub.
10. Stub unpacks result, returns to client.

**SECTION 6.4: MESSAGE QUEUING SYSTEMS**
Message-Oriented Persistent Comms

**Rationale:** Why Another Messaging System?:

**Scalability:**
- Other messaging systems, do not scale well geographically.

**Granularity:**
- Sockets supports messaging O(ms). Distributed messaging can take min/hours.

**What about RPC?:**
- In DS can’t assume receiver is “awake” => default “synchronous, blocking” nature of RPC often too restrictive.

**How about Sockets, then?:**
- Wrong level of abstraction (only “send” and “receive”).
- Too closely coupled to TCP/IP networks – not diverse enough

**Answer: Message Queueing Systems:**
- MQS give extensive support for Reliable Asynchronous Communication.
- Offer medium-term storage for messages – don’t require sender/receiver to be active during message transmission.
- Can store message if Not delivered immediately

Message-Oriented Persistent Comms. (/2)

**Message Queuing Systems:**
- Basic idea: applications communicate by putting messages into and taking messages out of “message queues”.
- Only guarantee: your message will eventually make it into the receiver’s message queue => “loosely-coupled” communications.
- Asynchronous persistent communication thro middleware-level queues.
- Queues correspond to buffers at communication servers.

**Four Commands:**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue.</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message.</td>
</tr>
<tr>
<td>Pull</td>
<td>Check a specified queue for messages, and remove the first. Never block.</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue.</td>
</tr>
</tbody>
</table>
Message-Queuing System Architecture

• **Operation:**
  – Messages are “put into” a *source queue*.
  – They are then “taken from” a *destination queue*.
  – Obviously, a mechanism has to exist to move a message from a source queue to a destination queue.
  – This is the role of the *Queue Manager*.
  – These are message-queuing “relays” that interact with the distributed applications and with each other.
  – Not unlike routers, these devices support the notion of a DS “overlay network”.

Role of Message Brokers

• **Rationale:**
  Often need to integrate new/existing apps into a “single, coherent Distributed Information System (DIS)”.
• **Problem:** different message formats exist in legacy systems
• Can’t “force” legacy systems into single, global message format.
• “Message Broker” allows us to live with different formats
• Centralized component that takes care of application heterogeneity in an MQ system:
  – Transforms incoming messages to target format
  – Very often acts as an application gateway
  – May provide subject-based routing capabilities ⇒ Enterprise Application Integration
**Message Broker Organization**

- General organization of message broker in a MQS – also known variously as an “interface engine”.

**IBM’s WebSphere MQ**

- **Basic concepts:**
  - Application-specific messages are put into, removed from queues
  - Queues reside under the regime of a queue manager
  - Processes can put messages only in local queues, or thru an RPC

- **Message transfer**
  - Messages are transferred between queues
  - Message transfer between process queues requires a channel
  - At each endpoint of channel is a *message channel agent*
  - Message channel agents are responsible for:
    - Setting up channels using lower-level n/w comm facilities (e.g. TCP/IP)
    - (Un)wrapping messages from/in transport-level packets
    - Sending/receiving packets
IBM’s WebSphere MQ (/2)

- Supported Topologies are:
  1. **Hub/spoke** topology (point-to-point queues):
     - Apps subscribe to "their" QM.
     - Routes to hub QM defined in spoke QMs.
  2. **Distributed Publish/Subscribe**:
     - Apps subscribe to topics & publish messages to multiple receivers.
     - 2 Topologies: **Clusters** and **Trees**:
       - **Cluster**: Cluster of QMs connected by channels. Published messages sent to all connected QMs of the published topic.
       - **Tree**: Trees allow reducing number of channels between QMs.

---

IBM’s WebSphere MQ (/2)

- **Principles of Operation**:
  - Channels are inherently unidirectional
  - Automatically start MCAs when messages arrive
  - Any network of queue managers can be created
  - Routes are set up manually (system administration)
Advanced Message Queuing Protocol (AMQP)

**Why AMQP?**

1. Lack of standardization:
   - MOM products unstandardized (mostly proprietary solutions).
     - E.g. 1: JMS Java-dependent only uses an API.
       => different JMS providers can’t ‘talk’ on wire level.
     - E.g. 2: IBM Websphere clunky and expensive

2. Need for bridges\(^1\) for interoperability:
   - For interoperability between different queueing systems, 3rd party vendors offer bridges.
   - Make architecture / topology complex, increase costs & reduce performance (additional delay).

\(^1\)Entities that help in different stages of message mediation

---

**AMQP (/2)**

**Characteristics of AMQP:**

- What is it? Open protocol for enterprise messaging, supported by industry (JP Morgan, Cisco, Microsoft, Red Hat, Microsoft etc.).
- Open/ Multi-platform / language messaging system.
- AMQP defines:
  1. Messaging capabilities (called AMQP model)
  2. Wire-level protocol for interoperability
- AMQP messaging patterns:
  1. Request-response: messages delivered to a specific queue (like C/S)
  2. Publish/Subscribe: messages delivered to a set of receiver queues
  3. Round-robin: message distribution to set of receivers based on availability

**AMQP Model (simplified):**

![AMQP Model Diagram](image.png)