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

Lecture Contents

• The world of Enterprise Information Systems (EIS)
• Intro to SOA & Middleware in EIS
  — EAI & Middleware and where this fits in to Business Processes
  — Evolution from EAI to SOA
  — Some theory of SOA
  — A SOA SCM example
• Types of communication in EIS
• Types of Distributed Communications
• Message-Oriented Middleware:
  — IBM MQ Systems,
  — RabbitMQ
Enterprise Information Systems in Brief

- **Enterprise information systems:**
  - Integrated ISs that support core BPs and functions.
  - e.g. Marketing, Accounting, Finance, Info security, HR, Compliance, Production, Purchasing, and Logistics.

- Know them from terms e.g.
  - ERP: Enterprise resource planning
  - SCM: Supply chain management
  - CPFR: Collaborative planning, forecasting, and replenishment
  - CRM: Customer relationship management
  - KM: Knowledge management
  - BI: Business Intelligence, Data Analytics etc

- Main EIS need is for data integration (data sharing/ exchange):
  - e.g., ERP & SCM improve SC; KM & CRM for (un)profitable customers
  - All these are facilitated by communication!

The Lie of the Land...

- A layer-wise outline of some technologies & how they interrelate.
- 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 often implemented by the **Enterprise Service Bus (ESB)** (like EAI message router).

The Changing Context(/2): EAI & Middleware

- Where in multi-layer architecture is business-logic?
  - 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 & enterprise data sources to share BP & data
  - 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 BPM 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
  - BPM-based f/w to build business applications is too tightly coupled to adapt.
  - For each change, business dept must interact with IT dept to change software.

- Still need BPM as processes will need to be optimised
  - Need increased agility in BPs for loosely-coupled business networks

- SOA is a solution for this

**Service-Oriented Architecture (SOA)**

**Ecosystem Evolution**

- Component-Based Development
  - Messaging Backbone
  - Point-to-Point connection between applications
  - Simple, basic connectivity

- Enterprise Application Integration (EAI)
  - EAI connects applications via a centralized hub
  - Easier to manage larger number of connections

- Service-Oriented Architecture
  - Integration of services through an Enterprise Service Bus (ESB)
  - Flexible connections with well-defined, standards-based interfaces

- Different parties (even in the same company) may have different:
  - OS, interface, data format, infrastructure, interaction protocols, language, etc

- Automating Supply Chain Mgmt => bring all these together

- As seen, EAI currently solves this but evolution has been to SOA
  - Supports flexible s/w devp thru ‘loose service coupling’ => no need to talk to IT.
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.

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 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, new/changed suppliers , IT responds quickly/ cheaply to the business decisions.
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.
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.

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
  - Some of these payloads are ‘active’ and some are ‘passive’
  - Also differ in granularities and whether synchronous or not.

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.
Message-Oriented Persistent Comms

- **Rationale:** Why Another Messaging System?:
  - Other messaging systems, do not scale well geographically.
- **Scalability:**
  - Sockets supports messaging O(ms). Distributed messaging can take min/hours.
- **Granularity:**
  - In DS can’t assume receiver is “awake” => default “synchronous, blocking” nature of RPC often too restrictive.
- **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.

Message-Oriented Persistent Comms. (/2)

- **Message Queueing 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>Poll</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”.

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 Example Diagram](image)

**AMQP Example: RabbitMQ Model**

![AMQP Example Diagram](image)
References


• Klauser, M., The use of Web Services in Improving Business Processes, Private Communication, 2012