COS 731
Software Engineering II

Lecture 4
Software Architecture Process (1)

17 August 2015
Topics

1. Process outline
   a. determining architectural requirements
   b. prioritising architectural requirements

2. Architecture design
   a. choosing the architecture framework
      i. N-tier client-server
      ii. messaging
      iii. publish-subscribe
      IV. broker
      v. process coordinator

3. Class exercise: Discussions & presentations
Software architecture addresses **non-functional requirements**

- **Functional requirements:**
  - define **what** an application does.

- **Non-functional requirements:**
  - define **how** the application provides the required functionality.

- **3 distinct areas of non-functionality requirements**
  - (i) technical constraints
  - (ii) business constraints
  - (iii) quality attributes

- Software architect must create a platform to support functional requirements & satisfy non-functional requirements.
Software architecture addresses non-functional requirements cont'd

3 distinct areas of non-functionality requirements

(i) **technical constraints** constrain design options
- e.g. only Java developers are available, so the application must be developed in Java

(ii) **business constraints** constrain design options
- e.g. in order to widen our customer base we must interface with XYZ tools

(iii) **quality attributes** define applic. requirements in terms of:
- performance, scalability, modifiability, security, availability, portability, usability, testability, supportability, etc.
What does a software architect do?

(1) Work with requirements team
   in order to understand overall system needs & ensure that appropriate quality attributes are understood

(2) Work with various application stakeholders:
   clients (business users), system administrators, etc.

(3) **Lead the technical design team**
   Architect leads a design team of:
   (i) system designers
   (ii) other architects
   (iii) technical leads etc.
   to produce architecture blueprint

(4) Work with project management
   provide help with project planning, estimation, scheduling etc.
1. Process outline:
A 3-step architecture design process

(1) **Determine architectural requirements**
create a statement or model of requirements that will drive the architecture design

(2) **Architecture design**
define the structure and responsibilities of the components

(3) **Validation**
test the architecture against existing and (possible) future requirements
2. Architectural requirements (1)

Architectural requirements
aka
architecturally significant requirements
aka.
architecture use cases

are the non-functional requirements (quality attributes & constraints)
2. Architectural requirements (2)

Prioritisation of requirements:

High => must have
Medium => must be supported at some stage
Low => part of wish list

Rqmnt examples:

<table>
<thead>
<tr>
<th>category</th>
<th>Architecture requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality attribute</td>
<td>security: All communication must be authenticated &amp; encrypted using certificates.</td>
</tr>
<tr>
<td></td>
<td>scalability: Application must be able to handle peak load of 500 concurrent users.</td>
</tr>
<tr>
<td>Constraint</td>
<td>business: Minimal expenditure on COTS</td>
</tr>
<tr>
<td></td>
<td>development: The system must be implemented in Java</td>
</tr>
</tbody>
</table>
2. Architecture design (1)

**Architecture design**

- most difficult task
- requires many years of SE and design experience

**A. Inputs for architectural design**

- architecture rqmnts documents

**B. Architecture design steps**

1. choose an overall strategy for the architecture based on proven architecture patterns.
2. specifying components of the application, how they fit into framework, & allocating component responsibilities

**C. outputs of architectural design**

1. **Views**: capture the design
2. **Documents**: explain design, reasons for design decisions & associated risks
2. Architecture design (2)

B. Architecture design steps

1. choose an overall strategy (to satisfy key requirements) for the architecture based on proven architecture patterns.

2. specifying components ...........

- 5 commonly used architecture patterns:

  1. n-tier client-server pattern
  2. messaging pattern
  3. publish-subscribe pattern
  4. broker pattern
  5. process coordinator pattern

are commonly used because:

- they are well understood & they work

Architectural design incorporates one or more patterns depending on complexity of application.
2. Architecture Patterns (1)

**N-tier client-server pattern**

- Commonly a suitable choice for ‘small’ (simple) applications

- **Key properties are:**
  
  (1) Separation of concerns
      - clear partitioning of logic into different tiers
  
  (2) Synchronous communication
      - request direction is:
        - client->web server-> app. server->database
      - each tier waits for response before proceeding
  
  (3) Flexible deployment
      - all tiers could be on same machine or
      - each tier could be on different machine
      - OR other combinations in-between
## 2. Architecture Patterns (2)

### N-tier client-server pattern

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Issues</th>
</tr>
</thead>
</table>
| Availability      | Servers in each tier can be replicated  
|                   | => high availability |
| Handling failures | Most web servers & application servers implement transparent fail-over.  
|                   | => if a server fails while a client is communicating with the server, client request is transparently re-directed to a live replica server  
|                   | => fail-safe capability |
| Modifiability     | Each tier (web, business logic, data management) is encapsulated  
|                   | => changes to internal logic of one tier do not have a ripple effect.  
|                   | => high modifiability |
| Performance       | Has proven high performance.  
|                   | Key considerations:  
|                   | (1) no. of concurrent threads in each server (web, application)  
|                   | (2) communication speed between tiers  
|                   | (3) amount of data transferred |
| Scalability       | Servers in each tier can be replicated  
|                   | => easy to scale  
|                   | BUT data management tier can be the bottleneck. |

**N-tier client-server example**

- **Client Tier**
  - Web Client
  - Web Client
- **Web Server Tier**
  - Web Server
- **Application Server Tier**
  - Application Server
- **Data Management Tier**
  - Databases

17/08/2015

COS731 Lecture 4
2. Architecture Patterns (3)

**Messaging pattern**

- **Key properties:**

  1. **Asynchronous communication**
     - A client sends a request (message) to the queue where it is stored until a server removes it.
     - Client continues without waiting for response to message.

  2. **Configurable QoS**
     - Queue can be configured for (a) high-speed non-reliable delivery or (b) slower reliable delivery.

  3. **Loose coupling**
     - No direct binding between clients & servers. Client is not aware of which server receives message & vice versa.
##Messaging pattern contnd

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Issues</th>
</tr>
</thead>
</table>
| **Availability**   | Physical queues with the same logical name can be replicated across different msg server instances  
=> high availability |
| **Handling failures** | If a client is communicating with a queue that fails, it can find a replica queue  
=> fail-safe capability |
| **Modifiability**  | Messaging is loosely couples i.e. senders & receivers are not directly bound thro’ an interface  
=> high modifiability  
**BUT:** changes to message formats may cause ripple effects. |
| **Performance**    | Technology can deliver thousands of messages per second  
=> high performance |
| **Scalability**    | Queues can be replicated across clusters of messaging servers hosted on single or multiple machines  
=> high scalability |
2. Architecture patterns (5)

**publish-subscribe pattern**

- **Key properties:**
  
  1. **Many-to-many messaging**
     - many publishers can publish on same topic. many subscribers can listen to same topic
  
  2. **Configurable QoS**
     - queue can be configured for (a) high-speed non-reliable delivery or (b) slower reliable delivery.
     - underlying communication style may be point-to-point or broadcast.
  
  3. **Loose coupling**
     - similar to messaging: no direct binding between clients & servers.
### 2. Architecture patterns (6)

#### publish-subscribe pattern contnd

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Issues</th>
</tr>
</thead>
</table>
| Availability      | Topics with the same logical name can be replicated across different server instances managed as a cluster.  
                   => high availability |
| Handling failures | If a publisher is communicating with a topic hosted by a server that fails, it can find a live replica server  
                   => fail-safe capability |
| Modifiability     | Publish-subscribe is loosely coupled.  
                   => high modifiability  
                   New publishers & subscribers can be easily added.  
                   BUT: changes to message formats may cause ripple effects. |
| Performance       | can deliver thousands of msgs per second  
                   => high performance |
| Scalability       | Topics can be replicated across clusters of servers hosted on single or multiple machines  
                   => high scalability  
                   Multicast/broadcast solutions scale better than point-to-point solutions. |
2. Architecture patterns (7)

Broker pattern

**Key properties**

(1) **Hub-and-spoke architecture**
   - broker acts as messaging hub, senders & receivers connect as spokes.
   - connections to broker are via ports associated with specific message format.

(2) **Performs message routing**
   - broker has logic for routing of messages from input to output ports
   - delivery path may be hard-coded or part of the message

(3) **Performs message transformation**
   - broker logic transforms source (input port) message type to destination (output port) message type
2. Architecture patterns (8)

Broker pattern contnd

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Availability</strong></td>
<td>Replication can be done in similar fashion to messaging and publish-subscribe =&gt; high availability</td>
</tr>
<tr>
<td><strong>Handling failures</strong></td>
<td>Input ports validate &amp; discard messages sent in wrong format. With replication, senders can fail-over to a live replica broker.</td>
</tr>
<tr>
<td><strong>Modifiability</strong></td>
<td>Changes to message transformation &amp; routing logic do not affect senders &amp; receivers. =&gt; high modifiability</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Brokers can become a performance bottleneck due to transformation &amp; routing activities.</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>High request loads can be handled thro’ clustering of broker instances.</td>
</tr>
</tbody>
</table>
Process coordinator pattern

Key properties

(1) Process encapsulation
- encapsulates sequence of steps needed for a business process.
- Receives process start request, calls servers in defined order & emits results.

(2) Loose coupling
- Server components are unaware of: (a) their role in business process (b) order of steps in process

(3) Flexible communication
- Communication can be synchronous or asynchronous
2. Architecture patterns (10)

Process Coordinator

Start process request → Process Coordinator → Process results

Step 1

Server-1 → Server-2 → Server-3 → Server-4

Step 4

process coordinator pattern

Quality attributes: Homework for the student
5. Class activities (20 minutes)

Given the following architecture patterns:

1. n-tier client-server pattern
2. messaging pattern
3. publish-subscribe pattern
4. broker pattern
5. process coordinator pattern

Select one application that you are very familiar with and determine which architectural pattern (and possibly a combination of patterns) you consider to be appropriate for the application. Give reasons for your answer.

Some application examples:
- Online retailer (e.g. Amazon.com)
- Internet banking
- Google searching
- Twitter
- WhatsApp
References