Topics

Last 3 lectures:

1. Process outline
2. Architecture design:  (a) choosing the architecture framework
   (b) allocating components
3. Validation:          (a) Using scenarios   (b) prototyping
4. Documenting a software architecture
5. ICDE case study      6. Attribute-Driven design

This lecture:

1. ICDE case study      (continued)
2. Attribute-Driven design (continued)

3. Class exercise:  Discussions & presentations
**Steps of ADD vs Gorton**

**Gorton:**

1. **Functional requirements**
   - **Step 1:** Confirm there is sufficient requirements info
2. **Design constraints**
   - **Step 2:** Choose an element of system to decompose
3. **Quality attribute requirements**
   - **Step 3:** Identify candidate architectural drivers
   - **Step 4:** Choose a design concept that satisfies architectural drivers
   - **Step 5:** Instantiate architectural elements & allocate responsibilities
   - **Step 6:** Define interfaces for instantiated elements
   - **Step 7:** Verify & refine requirements & make them constraints for instantiated elements

**Software architecture design**

**Architectural Requirements**

**Choose Architectural Framework**

**Allocate Components**

**Architecture Views**

**Architecture Document**
Case study: ICDE (1)

- Information Capture and Dissimination Environment (ICDE)

Requirements overview:

- A system to automatically capture & store data on actions performed by a user at a workstation.

  e.g. of data for Google search:
  - search query string
  - copies of pages returned by Google

- Data is to be analysed by third-party tools to provide ‘intelligent’ help to users
Case study: ICDE (2)

Project context

Application architecture for ver 1.0 (single-user, 2-tier)

- Data collection
  - a number of loosely coupled processes
  - transparently track user’s activities & stores them in data store
  - activities relate to Internet access, documents opened & browsed, etc.

- Data store
  - A COTS Relational database system
  - time-stamped data on user activities is stored in several tables

Component diagram for ver 1.0 application architecture (2-tier architecture)

- Data analysis
  - GUI-based tool which supports queries to the DB
  - tool was created for testing purposes
Overview of key objectives for ICDE ver 2.0:

(1) To provide an infrastructure to support a programming interface (API) for 3rd party client tools to access the ICDE data store. Must offer:
   - flexibility in terms of deployment,
   - plug into ICDE environment to obtain or provide info on users activities
   - provide simple read / write access to ICDE data store

(2) Evolve the ICDE architecture to support 100-150 users at low cost
3. Solution

3.1 Relevant patterns
3.2 Architecture overview
3.3 Structural views
3.4 Behavioural views
3.5 Implementation issues

Step 2: choose whole system

Step 3: Some candidate architectural drivers:
(a) Ease of data access
(b) Heterogeneous platform support
(c) Instant event notification
(d) Insulate 3rd party tools from the details of data store
(e) Provide < 5 sec response times for retrieval of <= 1000 rows
### Case study: ICDE (3)

#### ADD Step 3: (in detail)

- **a.** Ranking of requirements is done by stakeholders (High, Med, Low)
- **b.** **Second ranking** is done by software architect **based on architecture impact** (High, Med, Low)

#### Requirement Table

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Stakeholder priority</th>
<th>Architecture impact</th>
<th>Priority category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of data access</td>
<td>High</td>
<td>medium</td>
<td>(H, M)</td>
</tr>
<tr>
<td>heterogeneous platform support</td>
<td>High</td>
<td>High</td>
<td>(H, H)</td>
</tr>
<tr>
<td>Instant event notification</td>
<td>High</td>
<td>High</td>
<td>(H, H)</td>
</tr>
<tr>
<td>Insulate 3rd party tools from the details of data store</td>
<td>High</td>
<td>High</td>
<td>(H, H)</td>
</tr>
<tr>
<td><strong>Performance:</strong> provide &lt; 5 sec response times for retrieval of &lt;= 1000 rows</td>
<td>High</td>
<td>High</td>
<td>(H, H)</td>
</tr>
<tr>
<td><strong>Simplicity:</strong> flexible architecture design that is easy to evolve, extend</td>
<td>High</td>
<td>High</td>
<td>(H, H)</td>
</tr>
</tbody>
</table>
ADD Step 3: (in detail)

b. Second ranking is done by software architect based on architecture impact (High, Med, Low)

c. Select 5 or 6 in the (H, H) category. These are called:
   **Candidate architectural drivers**

### Requirement

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Priority category</th>
</tr>
</thead>
<tbody>
<tr>
<td>heterogeneous platform support</td>
<td>(H, H)</td>
</tr>
<tr>
<td>Instant event notification</td>
<td>(H, H)</td>
</tr>
<tr>
<td>Insulate 3rd party tools from the details of data store</td>
<td>(H, H)</td>
</tr>
<tr>
<td><strong>Performance:</strong> provide &lt; 5 sec response times for retrieval of &lt;= 1000 rows</td>
<td>(H, H)</td>
</tr>
<tr>
<td><strong>Simplicity:</strong> flexible architecture design that is easy to evolve, extend</td>
<td>(H, H)</td>
</tr>
</tbody>
</table>
ADD Step 4: (in detail)

Use candidate architectural drivers to determine types of elements, relationships & their interactions

4.1. Identify design concerns associated with architectural drivers,

e.g. (1) driver: Insulate 3rd party tools from the details of data store

design concerns:
modifiability: changes to data store should not affect 3rd party tools implementation

4.2. For each design concern, create a list of design patterns that address the concern.

(1) n-tier client-server: YES
(2) messaging: YES /NO???
(3) publish-subscribe: YES / NO???
(4) broker: NO (API provided)
(5) process coordinator: NO (only one DB)
ADD Step 4: (in detail)

Use candidate architectural drivers to determine types of elements, relationships & their interactions.

4.1. **Identify design concerns** associated with architectural drivers,

   e.g. (2)

   **driver:** provide < 5 sec response times for retrieval of <= 1000 rows from data store

   **design concerns:**
   - **Performance:**
   - **Scalability:**

4.2 For each design concern, **create a list of design patterns** that address the concern.

   (1) n-tier client-server: YES
   (2) messaging: YES / NO ??
   (3) publish-subscribe: YES / NO ???
   (4) broker: NO (API provided)
   (5) process coordinator: NO (only one DB)
ADD Step 4: (in detail)

Use candidate architectural drivers to determine types of elements, relationships & their interactions

4.1. Identify design concerns associated with architectural drivers,

e.g. (3)

driver:
Simplicity: flexible architecture design that is easy to evolve, extend

design concerns: Modifiability:

4.2 For each design concern, create a list of design patterns that address the concern.

(1) n-tier client-server: YES
(2) messaging: YES / NO ???
(3) publish-subscribe: YES / NO ???
(4) broker: NO (API provided)
(5) process coordinator: NO (only one DB)
ADD Step 4: (in detail)

Use candidate architectural drivers to determine types of elements, relationships & their interactions

4.1. **Identify design concerns** associated with architectural drivers,

   e.g. (3) **driver:**
   Instant event notification

   **design concerns:**
   Performance

4.2 For each design concern, create a list of design patterns that address the concern.

   (1) n-tier client-server: YES / NO ???
   (2) messaging: YES / NO ???
   (3) publish-subscribe: YES
   (4) broker: NO (API provided)
   (5) process coordinator: NO (only one DB)
ADD Step 4:

(in detail)

4.2 For each design concern, create a list of design patterns that address the concern.

e.g. (2)

**driver:** provide < 5 sec response times for retrieval of <= 1000 rows from data store

design concerns:
Performance & Scalability:

**design patterns** that address the concern:

(1) n-tier client-server: YES
(2) messaging: YES

For each pattern:

a. **identify pattern’s discriminating pmts**
   (1) n-tier client-server:
   (2) messaging:
   HOMEWORK

b. **estimate values of discriminating pmts**
   values will depend on technology used
   HOMEWORK
ADD Step 4: (in detail)

4.2 For each design concern, create a list of design patterns that address the concern.

4.3 Select patterns most appropriate for satisfying candidate architectural drivers. Use a matrix:

<table>
<thead>
<tr>
<th>Architecture driver</th>
<th>Pattern 1: n-tier client-server</th>
<th>Pattern 2: messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pros</td>
<td>Cons</td>
</tr>
<tr>
<td>Modifiability:</td>
<td>Each tier is encapsulated</td>
<td></td>
</tr>
<tr>
<td>Insulate 3rd party</td>
<td>=&gt; changes to internal logic of</td>
<td></td>
</tr>
<tr>
<td>tools from the</td>
<td>one tier do not have a ripple</td>
<td></td>
</tr>
<tr>
<td>details of data store</td>
<td>effect. =&gt; high modifiability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pros</td>
<td>Cons</td>
</tr>
<tr>
<td></td>
<td>Messaging is loosely coupled i.e.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>senders &amp; receivers are not</td>
<td></td>
</tr>
<tr>
<td></td>
<td>directly bound thro’ an</td>
<td></td>
</tr>
<tr>
<td></td>
<td>interface =&gt; high modifiability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>changes to message formats may</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cause ripple effects</td>
<td></td>
</tr>
</tbody>
</table>
## ADD Step 4: (in detail)

### 4.2 For each design concern, create a list of design patterns that address the concern.

### 4.3 Select patterns most appropriate for satisfying candidate architectural drivers. Use a matrix:

<table>
<thead>
<tr>
<th>Architecture driver</th>
<th>Pattern 1: n-tier client-server</th>
<th>Pattern 2: messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pros</td>
<td>Cons</td>
</tr>
<tr>
<td>Performance &amp; Scalability</td>
<td>Performance: Has proven high performance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scalability: Servers in each tier can be replicated =&gt; easy to scale</td>
<td>BUT: data management tier can be a bottleneck.</td>
</tr>
</tbody>
</table>
### ADD Step 4: (in detail)

4.3 **Select patterns most appropriate** for satisfying candidate architectural drivers.

(a) **Use a matrix**

(b) **choose patterns that together satisfy architectural drivers**

<table>
<thead>
<tr>
<th>Architecture driver</th>
<th>Pattern 1 <em>messaging</em></th>
<th>Pattern 2 <em>publish-subscribe</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>can deliver thousands of messages per second =&gt; high performance</td>
<td>(1) can deliver thousands of msgs per second =&gt; high performance</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td></td>
<td>(2) <strong>Many-to-many messaging</strong>: many publishers can publish on same topic. many subscribers can listen to same topic</td>
</tr>
<tr>
<td><strong>Performance:</strong> Instant event notification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A mechanism is needed to distribute ICDE generated events (for availability of new analysis results) as they occur in the data store</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ADD Step 4: (in detail)

4.3 Select patterns most appropriate for satisfying candidate architectural drivers.

(a) Use a matrix

(b) Choose patterns that together satisfy architectural drivers

(c) Combine selected patterns. Combination results in a new pattern

Textbook solution has the following relevant architectural patterns:

(1) 3-tier client-server pattern

3rd party tools are clients, communicating with the API implementation in the middle tier, which queries the data store.

(2) Publish-subscribe pattern

The middle tier contains a publish-subscribe capability

(3) Layered pattern

Both the client and middle tiers employ layers internally to structure the design.
Stakeholder architectural requirements for ICDE ver 2.0:
(1) Ease of data access
(2) heterogeneous platform support

Non-functional requirements for ICDE ver 2.0:
(1) Reliability: resilience to failures induced by 3rd party tools

Constraints for ICDE ver 2.0:
(1) ICDE ver 1.0 database schema must be used.
(2) Must run on MS Windows

CLASS EXERCISE:
Can the selected patterns satisfy the above requirements and constraints? If yes, why. If no, why not?