Implementation Architecture
Software Architecture VO/KU (707.023/707.024)

Denis Helic
KMI, TU Graz

Nov 7, 2012
Outline

1. Definition
2. Design
3. Behavior
4. Quality attributes
5. Prototype
Implementation Architecture

- Focuses on how the system is built
- Which technological elements are needed to implement the system
- Software packages, libraries, frameworks, classes, ...
- Addresses non-runtime quality attributes: configurability, testability, reusability, ...
- Comprised of components and connectors
Components and connectors reflect software entities and their relationships at the level of source and binary code.

Typically a number of implementation models.

Each model focuses on one of the concurrent subsystems or processes from the execution view.
Components

- Two types of components: application and infrastructure components
- Application components are responsible for implementing domain-level responsibilities
- These are responsibilities found in a detailed conceptual architecture
- Application components might be realized as binary packages, source packages, and files
Components

- Infrastructure components are needed to make the system run but are not related to the application functionality.
- E.g., HTTP Connection Handler in our sample is a typical infrastructure component.
- Whether a particular component is an application or an infrastructure component depends on the application.
- E.g., if we are building a Web application server then HTTP Connection Handler is an application component.
Components

- Often an infrastructure component acts as a “container” for application components
- A container component provides an execution environment for the contained components
- Typically, the container executes within a process and creates threads for application components
- E.g. a Web application server which runs multiple applications, each of them in their own threads
Component stereotypes

Figure: Implementation stereotypes from Software Architecture Primer
Connectors

- In implementation architecture connectors represent a “uses” relation.
- The arrow depicts the direction of this relation.
- The nature of communication is depicted through the connector styles.
Connectors

- API call: A component calls a method in another component (possibly only if both components are in the same process)
- Callback: The caller passes a reference of an object to the callee. The callee invokes a method on that object later.
- Network protocol: Needed when implementation components reside in different processes on networked machines. Components need to agree on a common protocol or use a standardized protocol
- OS signals: Communication between processes running on the same machine
Connectors

Figure: Implementation connectors from Software Architecture Primer
Connectors

- In some cases we depict ports as endpoints of connectors between components.
- Ports are used to identify a particular component interface.
- E.g. a component might be quite complex but it provides a simple interface for communication.
- E.g. the standard Java library provides an API.
**Example**

**Figure:** Example of implementation architecture from Software Architecture Primer
## Conceptual vs. Implementation

<table>
<thead>
<tr>
<th>Element</th>
<th>Conceptual</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>Domain-level responsibilities</td>
<td>Implementation module</td>
</tr>
<tr>
<td>Connectors</td>
<td>Information flow</td>
<td>“Uses” relationship</td>
</tr>
<tr>
<td>Views</td>
<td>Single</td>
<td>Split</td>
</tr>
</tbody>
</table>
Conceptual vs. Execution vs. Implementation

**Figure:** Conceptual vs. execution vs. implementation from Software Architecture Primer
Implementation architecture design

1. Find application components
2. Find infrastructure components
3. Interface design
4. Behavior design and verification
Application components

- Ideal 1-to-1 mapping of conceptual components onto application components is typically not possible.
- Some conceptual components will become infrastructure components.
  - E.g., persistent storage (databases) are typically infrastructure components.
- Some conceptual components are spread over a number of application components.
  - E.g., conceptual components have complex responsibilities.
A number of conceptual components can be mapped onto a single application component.

E.g. small number and simple responsibilities.

Complex conceptual components might map on additional application components.

UI components map onto one application component (i.e. HTML UI).

All other map onto a single application component on the server.
Infrastructure components

- Off-the-shelf components
- Frameworks (e.g. application framework)
- Servers (web, application, database, file)
- Generic clients (browser)
- In our sample at least two infrastructure components: browser, application server
Sample infrastructure components

- Web browser
- Web application server
- Apache Tomcat: http://tomcat.apache.org/
- Reference implementation of the Servlet API
- Servlet API abstracts the HTTP protocol from programmers and allows to write Java classes that handle HTTP requests
Sample infrastructure components

```java
public class SomeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        ...
        PrintWriter out = response.getWriter();
        ...
    }
}
```
Sample infrastructure components

- With a servlet we can read parameters from the user
- E.g. dataset name, dataset file
- We can respond with HTML
- E.g a new input form for uploading a dataset
Sample infrastructure components

- Servlets are still very low level
- We want to operate on a higher level of abstraction than reading/handling user parameters
- There are frameworks which abstract e.g. HTTP request/response cycle
- For example, for general UI applications MVC frameworks
- Component-based frameworks such as GWT
- Also, Java servlets are just one possibility (Python, PHP, etc..)
Infrastructure selection

- Research
- Conceptual issues, e.g. MVC, component-based, service-based, ...
- Execution issues, e.g. spawning of external processes
- Implementation issues, e.g. programming language
- Contextual issues, e.g. commercial products, open source, etc.
- Organizational issues, e.g. know-how, team, ...
Infrastructure selection

- Trough research you will obtain a list of several infrastructures
- Identification of criteria set (what is important)
- Weighting of criteria (to what extent is some feature important)
- Evaluation of modules
- Selection
There are \( n \) alternatives \( A_1, A_2, \ldots, A_n \)

There are \( m \) different criteria \( C_1, C_2, \ldots, C_m \)

Each alternative is for each criterion with score \( S_{ij} \)

Each criterion has a weight relative to its importance \( W_1, W_2, \ldots, W_m \)

The final score for the alternative \( A_i \):

\[
S(A_i) = \sum_{j=1}^{m} S_{ij} W_j
\] (1)
Weighted Scoring Method

For uniform distribution of weights:

\[ S(A_i) = \frac{1}{m} \sum_{j=1}^{m} S_{ij} \]  (2)
## Weighted Scoring Method

<table>
<thead>
<tr>
<th>Criteria group</th>
<th>Criterion</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Login and access system</td>
<td>F0</td>
</tr>
<tr>
<td>Functionality</td>
<td>Access rights and permission management</td>
<td>F1</td>
</tr>
<tr>
<td>Functionality</td>
<td>Content management system</td>
<td>F2</td>
</tr>
<tr>
<td>Functionality</td>
<td>'Product basket’ feature</td>
<td>F3</td>
</tr>
<tr>
<td>Functionality</td>
<td>Product catalogue management</td>
<td>F4</td>
</tr>
<tr>
<td>Functionality</td>
<td>Product comparison feature</td>
<td>F5</td>
</tr>
<tr>
<td>Functionality</td>
<td>Product management</td>
<td>F6</td>
</tr>
<tr>
<td>Functionality</td>
<td>Product tagging feature</td>
<td>F7</td>
</tr>
<tr>
<td>Functionality</td>
<td>Search feature</td>
<td>F8</td>
</tr>
<tr>
<td>Functionality</td>
<td>Reporting feature</td>
<td>F9</td>
</tr>
<tr>
<td>Quality</td>
<td>Adaptability/Source code quality</td>
<td>Q0</td>
</tr>
<tr>
<td>Quality</td>
<td>Reliability</td>
<td>Q1</td>
</tr>
<tr>
<td>Quality</td>
<td>Security</td>
<td>Q2</td>
</tr>
<tr>
<td>Quality</td>
<td>Technology/Programming language</td>
<td>Q3</td>
</tr>
<tr>
<td>Usability</td>
<td>Error reporting</td>
<td>U0</td>
</tr>
<tr>
<td>Usability</td>
<td>User interface/Ease of use</td>
<td>U1</td>
</tr>
<tr>
<td>Vendor</td>
<td>References</td>
<td>V0</td>
</tr>
<tr>
<td>Cost</td>
<td>Installation and implementation cost</td>
<td>C0</td>
</tr>
</tbody>
</table>
### Weighted Scoring Method

<table>
<thead>
<tr>
<th>Code</th>
<th>AVG¹</th>
<th>RW²</th>
<th>WSM³</th>
<th>Magento</th>
<th>OSCommerce</th>
<th>Zen Cart</th>
<th>PHPB2B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>6%</td>
<td>3</td>
<td>8%</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>F1</td>
<td>6%</td>
<td>3</td>
<td>8%</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>F2</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>F3</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>F4</td>
<td>6%</td>
<td>3</td>
<td>8%</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>F5</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F6</td>
<td>6%</td>
<td>3</td>
<td>8%</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>F7</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F8</td>
<td>6%</td>
<td>2</td>
<td>5%</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>F9</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Q0</td>
<td>6%</td>
<td>5</td>
<td>13%</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Q1</td>
<td>6%</td>
<td>2</td>
<td>5%</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Q2</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Q3</td>
<td>6%</td>
<td>4</td>
<td>10%</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>U0</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>U1</td>
<td>6%</td>
<td>2</td>
<td>5%</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>V0</td>
<td>6%</td>
<td>1</td>
<td>3%</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>C0</td>
<td>6%</td>
<td>5</td>
<td>13%</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

| AVG Score | 4.44 | 3.06 | 2.94 | 2.67 |
| Norm. AVG Score | 34% | 23% | 22% | 20% |
| WSM | 4.33 | 3.05 | 2.95 | 2.65 |
| Norm. WSM | 33% | 24% | 23% | 20% |

Table 10.4: Scores of software packages based on WSM.
Sample infrastructure selection

- Apply the same methodology for your projects
- We select Open Source Java with Tomcat
- We select component-based framework
- Apache Wicket as a Web application framework
- We select SNAP library for network analysis
- We select gnuplot for diagrams
Interface design

- For all application and infrastructure components we need to define interfaces (ports)
- Helps in clarifying the responsibilities of a component
- Some interfaces are also standardized
- E.g. HTTP, SQL, File I/O
Sample interface design

- UI: Combination of HTML/HTTP
- It is standardized!
- Web application server: HTTP/Servelt API
- It is standardized
- Component separation
- Wicket components are specified by the framework
Sample interface design

- We need to specify interface between Wicket and our application logic.
- A new interface is needed.
- We need to tell to the application logic that there is a request for an 
  e.g. calculation.
- e.g. `executeCalculation(user, dataset)`
Sample interface design

- We should also tell to the system where to write calculations, datasets, users, etc.
- Typically these are only files
- We can store users in an e.g. XML file
  
ed.g. users.storeUser(user);
- calcs.storeCalc(calc);
Behavior design

- Now we need to go into details
- Use-case maps are not enough anymore
- We need to investigate behavior at the operation level
- Thus, we need a sequence diagram
Sample implementation Architecture

Figure: Detailed sample implementation architecture
Figure: Sample implementation architecture
Figure: Sample sequence diagram
Non-runtime quality attributes

- Since implementation view addresses build structure
- It is the right place to consider non-runtime quality attributes
- E.g. maintainability, extensibility, reusability, ...
- We can use a mechanism similar to use-case maps
- Impact-maps: try to investigate what parts of the system need to change if “something” happens
Impact-maps

- Map 1: new external system - interface to external system needs to be changed
- Map 2: new calculation - network analysis application component needs to be changed
- Map 3: new UI - UI component needs to be changed
- Goal: as few component changes as possible
To show that the architectural solution is feasible we implement prototypes.

For each identified application component we provide implementation.

Deploy it within the infrastructure components.

Test it and check correctness, functionality, quality-attributes.