History and Future of Software Architecture

A CS 718 Lecture

R K Joshi
IIT Bombay
The Early Culprit

The GOTO Statement
Indispensible Low Level Abstraction in Assembly Language or Machine Language
But
Havoc
in
High Level Programming Languages
How programs looked
Hey, I have seen it in Linux Sources!
It's Solution Was Structured Programming.
But the Programs got bigger and bigger bigger
New Abstractions were needed to Manage them
What Were they?
Functions, Procedures and Modules
Libraries
Classes and Objects
Interfaces and Components
Services
... and so on
The Birth and Growth of Methodologies
Structured Methodology
ER
DFD
Modularity
Cohesion
Coupling

and

the Waterfall Model
The Object Oriented Software Development Methodologies
Peter Coad - Patterns
Yourdon – OOA
Rebecca Wirfs Brock - OOD
Kent Beck and Cunningham's CRC
Booch- OOD
Jacobson- USE CASES
Rumbagh - OMT
...
The UML
Revolved around
Classes
Aggregation
Association
Inheritance
Membership
Then came a Major Contribution to Software Architecture
Design Patterns
Architecture Outgrew Products into Processes and Roles
Patterns in other fields of software design
  Networking patterns
  Distributed computing patterns
  Parallelism patterns
  Documentation patterns
  Coding patterns
  Testing patterns
  Requirements patterns

SA book from CMU, POSA
Pattern-Oriented Software Architecture
A System of Patterns

Frank Buschmann
Regine Meunier
Hans Rohner
Peter Sommerlad
Michael Stal

WILEY

SOFTWARE ARCHITECTURE
PERSPECTIVES ON AN EMERGING DISCIPLINE

MARY SHAW
DAVID GARLAN
PEARSON
Anti-patterns
what is to be avoided

Refactoring patterns
the restructuring
utility in on-the-fly software development (Agile methods)
What really is software architecture?
Then came Zachmann's Model from IBM
**Rule 1:**
Columns have no order

**Rule 2:**
Each column has a simple, basic model

**Rule 3:**
Basic model of each column is unique

**Rule 4:**
Each row represents a distinct view

**Rule 5:**
Each cell is unique

**Rule 6:**
Combining the cells in one row forms a complete description from that view

**Rule 7:**
The logic is recursive
<table>
<thead>
<tr>
<th>Agents</th>
<th>Processors</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects</td>
<td>Machines</td>
<td>Parallelism</td>
</tr>
<tr>
<td>Processes</td>
<td>Networks</td>
<td>Security</td>
</tr>
<tr>
<td>Classes</td>
<td>Connecting Devices</td>
<td>Availability</td>
</tr>
<tr>
<td>Modules</td>
<td>Interfaces</td>
<td>Fault</td>
</tr>
<tr>
<td>Services</td>
<td>Servers</td>
<td>Tolerance</td>
</tr>
<tr>
<td>Components</td>
<td>Cloud</td>
<td>Response</td>
</tr>
<tr>
<td>Files</td>
<td>Databases</td>
<td>Heterogeneity</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td>Speed</td>
</tr>
<tr>
<td>Calls</td>
<td></td>
<td>Look</td>
</tr>
</tbody>
</table>
Usability
Reusability
Adaptability
Configurability
Reconfigurability
Evolvability
Understandability
Tractability
Serviceability
...

Administrators
Programmers
Bug fixers
Owners
Developers
Bug Reporters..

Distribution
Parallelism
Security
Availability
How to you conceptualize all this?

What models do you build?

Will it not soon go out of control if you don't!
Some concerns addressed by Architectural Patterns
Some concerns addressed by
Architectural Description Languages
Some concerns addressed by
Formal Specification Languages and Tools
Some concerns addressed by Architecture Evaluation Frameworks
Remember ..
What?
We are crossing the boundary of one a.out or one exe program
Programming in the Large
But the Bottom Line is ..
The Traditional Wisdom of Cohesion and Coupling
What shall we do in the next lecture?
A Formal Language called CCS (of Robin Milner) to express and build architectures

We learn next, how to express:

- Component behavior
- Components and Connectors
- Non-determinism
- Composition of smaller components into bigger architectures
A Few Examples
2/3 split-join
When the start signal arrives, the bottom agent sends two instances of the same problem to two agents. The left agent and the right agent solve the problem that they receive, and output their solutions to the top agent. The top agent compares the two solutions received from the two solvers and then outputs on either of two ports (right, doubt) indicating whether the solutions compared. However, the top agent acts only after it receives a tick from the bottom agent. The bottom agent ticks the top agent as soon as it sends the problem instances to left and right agents.
<table>
<thead>
<tr>
<th>Types</th>
<th>Start</th>
<th>Intermediate</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-Level</td>
<td>Event Sub-Process</td>
<td>Catching</td>
</tr>
<tr>
<td></td>
<td>Event Sub-Process</td>
<td>Non-Interrupting</td>
<td>Catching</td>
</tr>
<tr>
<td>None</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Message</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Timer</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Error</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Escalation</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Cancel</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>Compensation</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>Conditional</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Link</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Signal</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Terminate</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Multiple</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Parallel Multiple</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
Ponraj
Someshwar

HOMEWORK

Full scoring by

Ponraj
It would be useful to search for services by their workflow patterns.
Base of Semantic Interpretation: Ontologies

- Helps to formally model the structure of a system.
- The output process forms relations that emerge from observation and which are useful in an operational sense.
- An example of such a system can be a company with its employees and their inter-relationships.
- The ontology engineer
  -Formulates a problem to formalize
  -Formulates the formalization using predicates
- Diagram creation Program
- Free
- cross platform
- Open source
- Extensible
- Intuitive UI
- Sheets
- (reasonably) stable
What is IMS?

Instructional Management System (IMS) facilitates the creation of shared learning environments and organizes online learning by making teachers, administrators, district staff and stakeholders view and share educational resources.

IMS provides a unified learning portal that enables schools and districts to communicate, collaborate and share all of their critical learning resources and more.