Architectural Design
of
Distributed Business Systems

Tutorial M3.
TOOLS Europe 2001, 13 March 2001

Trygve Reenskaug
Mogul Norway, Oslo
Architectural Design of Distributed Business Systems

Tutorial M3, TOOLS Europe 2001
13 March 2001

Trygve Reenskaug
Mogul Norway AS, Oslo

http://www.ifi.uio.no/~trygver

The Autokon system
for the Computer-Aided Design of Ships

1960 - Architecture created
1962 - First deployed
1965 - First international
1997 - Converted to PC
Architecture still valid!!

The Autokon architecture
Why the long life?

- Shipbuilding essentially unchanged
- Luck - random access mass storage devices enabled database solution
- Reality reflected into
  Clean, intelligible modular structure
- Expressed user needs as symptoms for required abstractions
The Autokon architecture
Was it worth it?

Y E S !
¤ System integrity maintained through major revisions
¤ System reliability maintained through major revisions
¤ System is comprehensible to users and maintainers
¤ Architecture spans decades of development projects

Exercise 1: What do WE mean by System Architecture?

• ........................................................................
• ........................................................................
• ........................................................................
• ........................................................................
• ........................................................................
• ........................................................................

Highlights
• Why architecture is important & What it is
• Habitable Architectures created for people
  – Enterprise models
  – Personal information environments
  – System models, the heavy part
• UML Collaboration, a powerful abstraction for architectural topology
  – Collaboration (role model) specify many instances
  – CollaborationInstance depicting a concrete topology
• Patterns, a literary style for sharing experience
  – Wrap Entity Beans with Session Beans
  – Caterpillars Fate: Smooth transition from analysis to design
• Summary and Conclusion
WHY ARCHITECTURE?

- Help create habitable systems
- Help create long-lived, robust systems

i.e., aim for simplicity:

...simple systems are easy to master
...simple systems are easy to build - correctly
...simple systems are easy to maintain

Theory X vs. Theory Y (MacGregor)

Theory X:
- Dislike work
- Lack ambition
- Are irresponsible
- Are resistant to change
- Prefer to be led rather than to lead

Theory Y:
- Are willing to work
- Are capable of self-control
- Are willing to accept responsibility
- Are imaginative and creative
- Are capable of self-direction

Authoritarian leadership

Rigid, controlling environment

Bored, passive workers

Inspiring leadership

Flexible, supportive environment

Interested, creative workers
Vision of Distribution
anno 1973

Mary's System
Jonas System
Yngvar's System
Kari's System
Anton's System

The Importance of the User’s Mental Model

- A good conceptual model allows us to predict the effects of our actions.
- Without a good model we operate by rote, blindly.
- We do operations as we are told; we cannot fully appreciate why, what effects to expect, or what to do if things go wrong.

Donald A. Norman: The Design of Everyday Things

Make IT so simple that even a programmer can understand IT

Edsger Dijkstra:

- Testing can only show the presence of bugs
- Testing can never show the absence of bugs
- so, the number of bugs in the system when you deliver it is proportional to the number of bugs found during testing
- The only way to avoid bugs is not to put them in in the first place
- I.E., Keep It Simple, Stupid (KISS)

60.000 bugs removed from Windows2000 during testing
Architectural styles

Architecture for:
Chaos-Beauty-Flexibility?

Architecture Example

First Priority: Make it Habitable!
Second Priority: Make it Buildable!
Third Priority: Make it Cost Effective!

RATIONAL's Definition of Architecture
"Principles of Architecting Software Systems"

• Software architecture encompasses
the set of significant decisions
about the organization of a software system
  – Selection of the structural elements and their interfaces
  – Behavior as specified in collaborations among those elements
  – Composition of these structural and behavioral elements into larger subsystems
  – Architectural style that guides this organization

• Software architecture also involves
  – Functionality, Usability, Resilience, Performance, Reuse, Comprehensibility
  – Economic and technology constraints and tradeoffs
  – Aesthetic concerns
Highlights

- Why architecture is important & What it is
- Habitable Architectures created for people
  - Enterprise models
  - Personal Information environments
  - System models, the heavy part
- UML Collaboration, a powerful abstraction for architectural topology
  - Collaboration (role model) specify many instances
  - CollaborationInstance depicting a concrete topology
- Patterns, a literary style for sharing experience
  - Wrap Entity Beans with Session Beans
  - Caterpillars Fate: Smooth transition from analysis to design
- Summary and Conclusion

Personal, Integrated Information Environments

Enterprise level
- Work processes
- UML Object Model

Personal level
- User's mental model
- UML Collaboration

System level
- Design models
- All of UML

Enterprise level Example: Travel Expense

Enterprise level
- Work processes
- UML Object Model
Two Approaches to Object Orientation

"East Coast Approach":
Object orientation is a smart programming artifact.
An object is an instance of a class.
Simula; Master complexity;
Classification Theory

"West Coast Approach":
Object Orientation is a powerful modeling paradigm.
An object encapsulates state and behavior so that it can collaborate with other objects.
Smalltalk; Personal Information Systems;
Computer Augmentation + Lisp + Simula (+ Operating Systems)

We achieve a separation of concern: a collaboration describes the objects involved in one or more functions, describing the objects involved only and focusing on the relevant object properties.

In the collaboration, we study patterns of interacting objects:
What are the objects, what are their responsibilities, and how do they collaborate to achieve the system functionality.

The classifierRole represents the object’s position in the structure and its responsibility for performing its part of the system function.
Behavior: Action-Object Flow Diagram
Travel Expense Model

Who
/ Traveler
<Plan trip>
<Buy tickets>
<Write exp. Report>

/ Authorizer
Travel perm. request
Travel permission
<Decide>
<Check OK>
<Authorize>

/ BookKeeper
Expense Report
<Check>
<Authorize>

/ Paymaster
Bookkeeper
Payment Request
<Pay out>

What
Activity
A Swimlane for each Object or ClassifierRole

When
Who
What
When

Data Object
Action

Pipes and Filters:
Travel Expense Model

Who
/ Traveler
<Plan trip>
<Buy tickets>
<Write exp. Report>

/ Authorizer
TravelRecord [request]
TravelRecord [permission]
<Decide>
<Check OK>
<Authorize>

/ BookKeeper
TravelRecord [report]
<Check>
<Authorize>

/ Paymaster
TravelRecord [authorized]
<Pay out>

Data Object changes state over time
Personal Information Environment

Task: Decide permission

Travel Expense Model

// Traveler
// Authorizer
// BookKeeper
// Paymaster

<Plan trip>
<Buy tickets>
<Write exp. Report>
<Travel>

<Travel request>
<Travel permission>
<Travel>
<Expense Report>
<Check OK>
<Check OK>

<Check OK>
<Authorize>

<Check>
<Authorize>

<Pay out>
<Payment Request>
<Check OK>
<Authorize>

<Authorize>
<Expense Report>
<Check OK>
<Authorize>

<Check>
<Bookkeeping>
<Check OK>
<Authorize>

© Trygve Reenskaug 2001
Travel Permission Decision Tool

Travel Expense System

Traveler: Peter
Period: week 11
Planned cost: USD 2000

Purpose: Attend TOOLS Europe 2001

Current plan for Peter

<table>
<thead>
<tr>
<th>Activity</th>
<th>Week</th>
<th>Budget + Commitments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Item</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Travel</td>
</tr>
</tbody>
</table>

Permissions: Permit, Reject

Required Topology

Architecting Distributed Systems

System level Design models
All of UML
Highlights

- Why architecture is important & What it is
- Habitable Architectures created for people
  - Enterprise models
  - Personal information environments
  - System models, the heavy part

- UML Collaboration, a powerful abstraction for architectural topology
  - Collaboration (role model) specify many instances
  - CollaborationInstance depicting a concrete topology

- Patterns, a literary style for sharing experience
  - Wrap Entity Beans with Session Beans
  - Caterpillars Fate: Smooth transition from analysis to design

- Summary and Conclusion

Enterprise Production Planning & Control

The (toy) sample user interface is a Java Applet that is run from a web browser. The interface interacts with a background service that is also written in Java.

The top bars in the bar chart shows the earliest times when the activities can be performed. The bottom bars show when the activities all have to be done by the same resource, and this resource can only serve one activity at the time.
**Use Case 2: Frontloading**

**UML Instance Interaction Diagram**

```
Use Case 2: frontloading
UML Instance Interaction Diagram

fL(t) = public void
frontload (int time)

1: Project
2: Activity-A
3: Activity-B
4: Activity-C
5: Activity-D
6: Activity-E
7: Activity-F

Project
0
20

Activity-A
1:6
6

Activity-B
7:4
10

Activity-C
7:7
13

Activity-D
14:3
16

Activity-E
14:4
17

Activity-F
18:2
19
```

**The UML Collaboration**

**and the ClassifierRole**

```
The UML Collaboration and the ClassifierRole

1: mess
2: mess

A collaboration describes how an operation, like a use case, is realized by a set of classifiers and associations used in a specific way. The collaboration defines a set of roles to be played by instances, as well as a set of interactions that define the communication between the instances when they play the roles.

What are the objects, what are their responsibilities, and how do their collaborate to achieve the system functionality.
```
Interfaces in the Basic scheduling collaboration

The interfaces specify minimal requirements to receiving classes

<table>
<thead>
<tr>
<th>Predecessor</th>
<th>Activity</th>
<th>Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>fL(t1)</td>
<td></td>
<td>fL(t2)</td>
</tr>
</tbody>
</table>

«Interface»
FrontloadIntf
frontLoad(time)

The interfaces specify minimal requirements to receiving classes.

Interface Definitions

Java

```java
public interface FrontloadIntf {
    public void frontLoad (int t);
}
```

The UML Class playing a role in a Collaboration

:ClassName

1: mess1
2: mess2

An Instance of a given class playing the specified role at run time
Assign classes to roles
(Optional)

```
public class Activity
    implements FrontloadIntf {
    private FrontloadIntf[] successors;
    ... ...
    public void frontload (int t) {...}
    ... ...
}

public class Project
    implements FrontloadIntf{
    private FrontloadIntf[] startActivities;
    ... ...
    public void frontload (int t) {...}
    ... ...
```

Class Definitions
Java

Highlights

- Why architecture is important & What it is
- Habitable Architectures created for people
  - Enterprise models
  - Personal information environments
  - System models, the heavy part
- UML Collaboration, a powerful abstraction for architectural topology
  - Collaboration (role model) specify many instances
  - CollaborationInstance depicting a concrete topology
- Patterns, a literary style for sharing experience
  - Wrap Entity Beans with Session Beans
  - Caterpillars Fate: Smooth transition from analysis to design
- Summary and Conclusion
Separate Presentation and Business Objects

An Unrealistic Implementation

Java RMI Remote Method Invocation

Exercise 3: Suggest an implementation for this system
A **subsystem** is a grouping of model elements that represents a behavioral unit in a physical system. A subsystem offers interfaces and has operations.

The Subsystem hides detail to simplify the drawing, but is neither visible at compile time nor at run time.
High level Collaboration with UML Components

Tool

Responsibility

PlanningService

Responsibility

«Interface»

Project

The Component simplifies the system. It supports separate compilation and deployment; the separation is visible at run time.

Assign Components to Physical Nodes

Tool

PC

Application Server

Planning Service

The Node is a container of components. In itself, it does not offer operations to its environment, so it cannot be a classifierRole.

Some UML Definitions

- A Collaboration describes how an operation, like a use case, is realized by a set of classifiers and associations used in a specific way. The collaboration defines a set of roles to be played by instances, as well as a set of interactions that define the communication between the instances when they play the roles.

- The Instance is encapsulated. It has identity, interface and state.

- An instance of a given Class can play one or more roles at run time. A ClassifierRole can be implemented by many ways.

- A Subsystem is a grouping of model elements that represents a behavioral unit in a physical system. A subsystem offers interfaces and has operations.

- A Component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.

- A Node is a run-time physical object that represents a computational resource.
**UML Diagrams**  
*(Architecturally Significant)*

### Build-time Diagrams (Code and Code Management)
- Class Diagrams
- Deployment Diagrams
- Component Diagrams

### Run-time Diagrams
- Use Case Diagrams
- Collaboration Diagrams
  - Objects, ClassifierRoles, Subsystems, Components
- Interaction & Sequence Diagrams
- Object Diagrams
- Statechart Diagrams
- Activity Diagrams / Action-Object Flow Diagrams

---

**Highlights**

- Why architecture is important & What it is
- Habitable Architectures created for people
  - Enterprise models
  - Personal information environments
  - System models, the heavy part
- UML Collaboration, a powerful abstraction for architectural topology
  - Collaboration (role model) specify many instances
  - CollaborationInstance depicting a concrete topology

- Patterns, a literary style for sharing experience
  - Wrap Entity Beans with Session Beans
  - Caterpillars Fate: Smooth transition from analysis to design

- Summary and Conclusion

---

**Design Patterns**

Christopher Alexander, Architect:

*Each pattern describes a problem that occurs over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice.*

Jim Coplien and Doug Schmidt, Software Engineers:

- A clear statement of a problem and its context.
- Offering a concrete solution addressing the problem
- A clear statement of the forces that motivate the solution.

---

*A pattern* tells you how to solve a problem.

*A framework* solves the problem for you.
Highlights

- Why architecture is important & What it is
- Habitable Architectures created for people
  - Enterprise models
  - Personal information environments
  - System models, the heavy part
- UML Collaboration, a powerful abstraction for architectural topology
  - Collaboration (role model) specify many instances
  - CollaborationInstance depicting a concrete topology
- Patterns, a literary style for sharing experience
  - Wrap Entity Beans with Session Beans
  - Caterpillars Fate: Smooth transition from analysis to design
- Summary and Conclusion

Wrap Entity Beans with Session Beans
Ed Roman, TheServerside.com, August, 2000

"Artist's Impression"

Business API
Business Logic

Persistent Data
Data Logic

A Real Example Application Execution Architecture
Highlights

• Why architecture is important & What it is
• Habitable Architectures created for people
  – Enterprise models
  – Personal information environments
  – System models, the heavy part
• UML Collaboration, a powerful abstraction for
  architectural topology
  – Collaboration (role model) specify many instances
  – CollaborationInstance depicting a concrete topology
• Patterns, a literary style for sharing experience
  – Wrap Entity Beans with Session Beans
  – Caterpillars Fate: Smooth transition from analysis to design
• Summary and Conclusion

Caterpillar’s Fate - 1
Norman Kerth, Coplien’s book

The main patterns

Highlights

• Why architecture is important & What it is
• Habitable Architectures created for people
  – Enterprise models
  – Personal information environments
  – System models, the heavy part
• UML Collaboration, a powerful abstraction for
  architectural topology
  – Collaboration (role model) specify many instances
  – CollaborationInstance depicting a concrete topology
• Patterns, a literary style for sharing experience
  – Wrap Entity Beans with Session Beans
  – Caterpillars Fate: Smooth transition from analysis to design
• Summary and Conclusion
Personal, Integrated Information Environments

- Habitable Systems
- Theory X for inspiring organizations
- Explicit Users' Mental Models

Control Components through their Responsibilities and Interfaces

Plan Behavior Who - What - When

Traveler
- Plan trip
- Authorize ticket
- Travel
- Write expense report

Authorizer
- Authorize ticket
- Authorize expense report

BookKeeper
- Check expense report
- Check OK

Paymaster
- Pay out
- Pay stub

PlanningService
- Responsibility

Tool
- Responsibility

Project
Instruct implementors and maintainers

Questions ? Comments ?

THE END
More details

- http://www.ifi.uio.no/~trygver
- trygve.reensgaug@ifi.uio.no
  The reference work. This book is out of print. A .pdf version can be downloaded free from above website.
  The theory of role modeling
  ftp://ftp.nr.no/pub/egil/ConceptualModelingOO.ps.gz
  http://cgi.omg.org/cgi-bin/doc?ad/01-02-13
- http://hillside.net/patterns/
- http://theserverside.com/patterns/
- IBM WebSphere: http://www.redbooks.ibm.com/abstracts/sg246161.html