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**

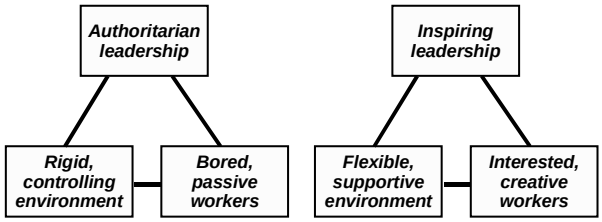
WHY ARCHITECTURE?

- **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**

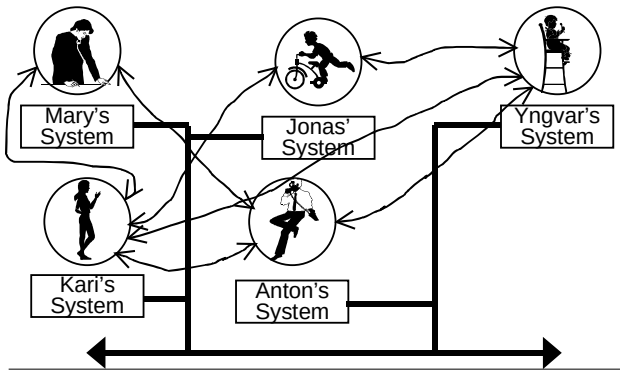
Theory X vs. Theory Y (MacGregor)

Theory X:	Theory Y
• Dislike work	• Are willing to work
• Lack ambition	• Are capable of self-control
• Are irresponsible	• Are willing to accept responsibility
• Are resistant to change	• Are imaginative and creative
• Prefer to be led rather than to lead	• Are capable of self-direction

Theory X vs. Theory Y (MacGregor)



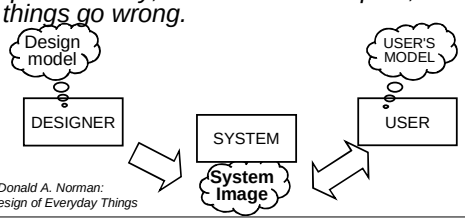
Vision of Distribution anno 1973



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 10

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

Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 11

Make IT so simple that *even a programmer can understand IT*

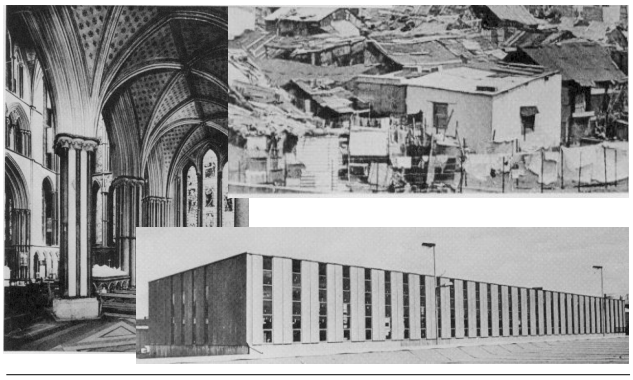
Edsger Dijkstra:

- **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 Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 12

Architectural styles



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 13

Architecture for:
Chaos-Beauty-Flexibility?

Architecture Example

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

Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 14

First: Architect describes end result.
Second: Architect describes how to build.

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***

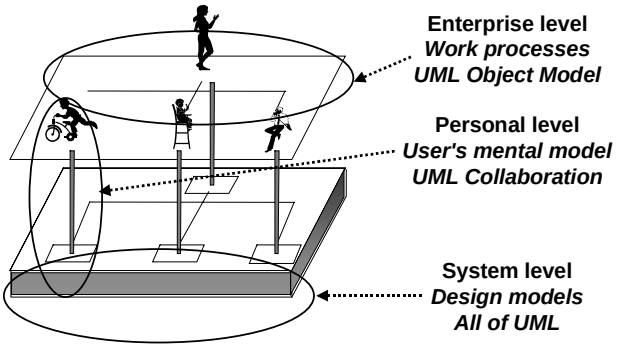
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 15

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**

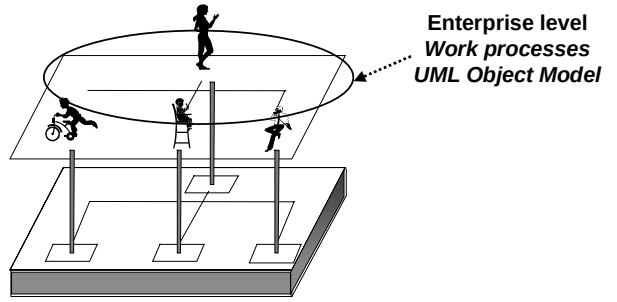
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 16

Personal, Integrated Information Environments



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 17

Enterprise level *Example:Travel Expense*



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 18

Two Approaches to Object Orientation

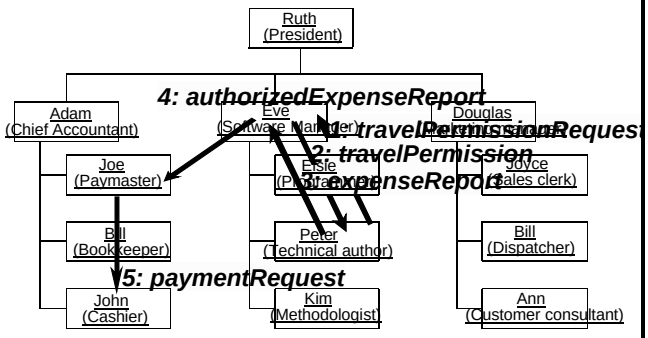
"East Coast Approach":
Object orientation is a smart programming artifact.
An object is an instance of a class.

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)

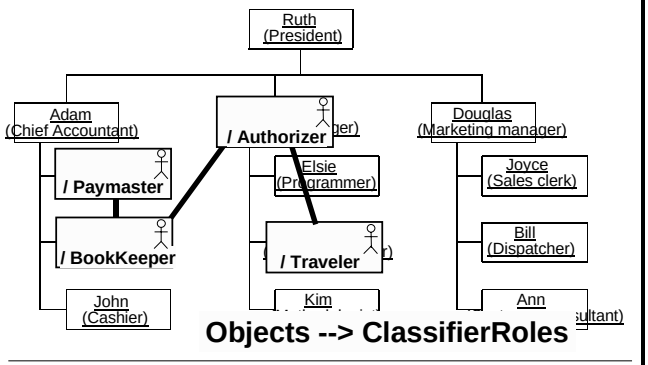
UML Instance Interaction Diagram *Travel Expense Model*



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.

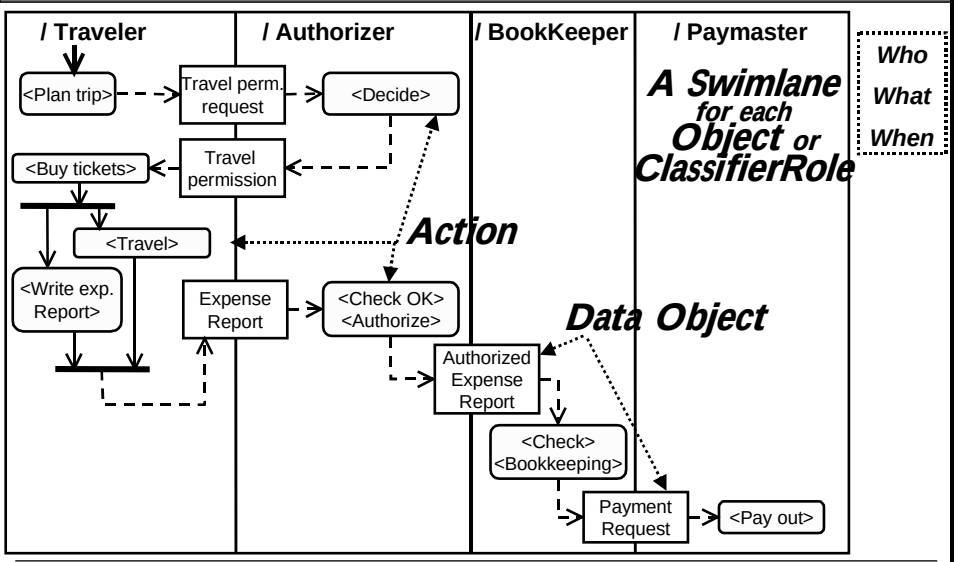
In the collaboration, we study patterns of interacting objects: What are the objects, what are their responsibilities, and how do they collaborate to achieve the system functionality.

UML Collaboration Diagram *Travel Expense Model*



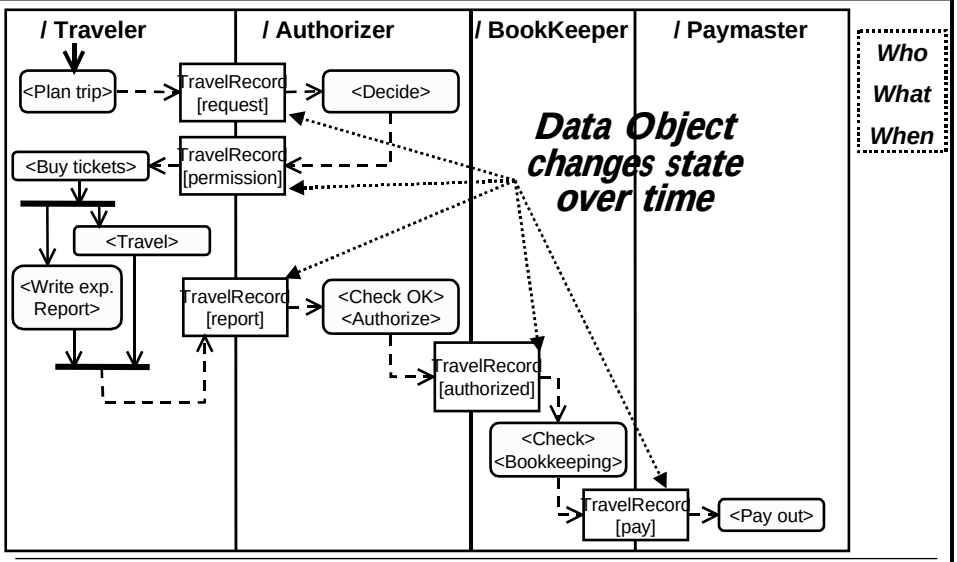
The *classifierRole* represents the object's position in the structure and its responsibility for performing its part of the system function.

Behavior: Action-Object Flow Diagram
Travel Expense Model



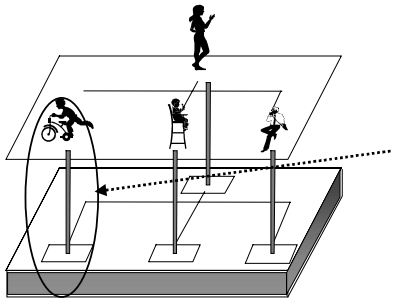
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 22

Pipes and Filters:
Travel Expense Model



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 23

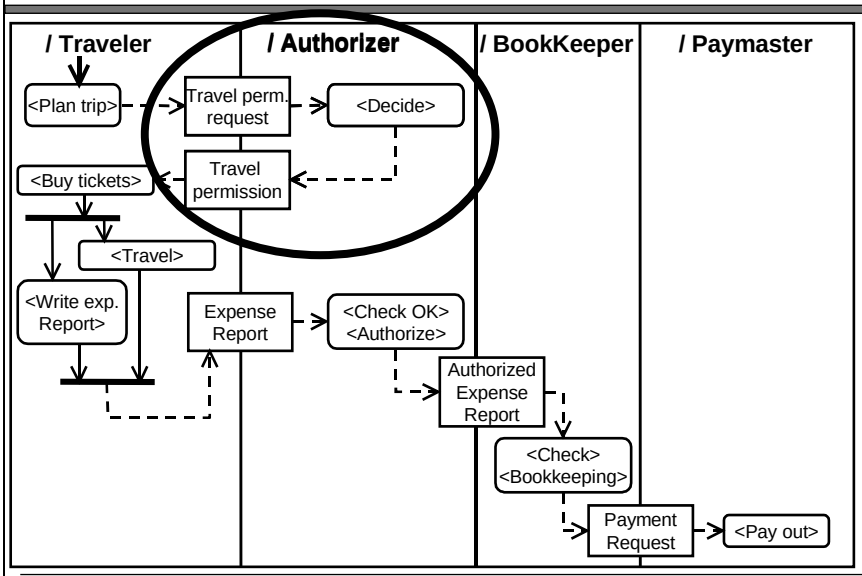
Personal Information Environment



Personal level
User's mental model
UML Collaboration

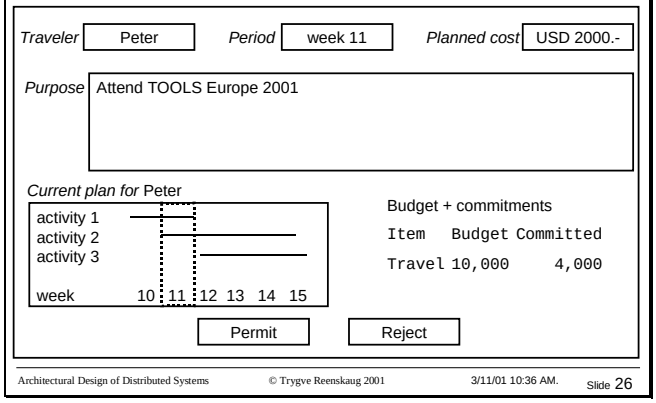
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 24

Task: Decide permission
Travel Expense Model



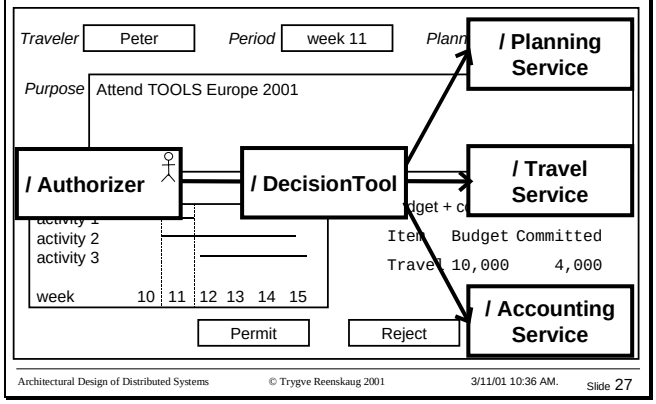
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 25

Travel Permission Decision Tool
Travel Expense System



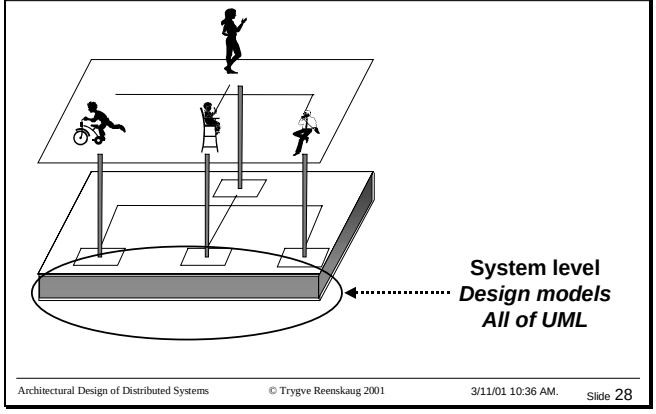
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 26

Travel Permission Decision Tool
Required Topology



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 27

Architecting Distributed Systems



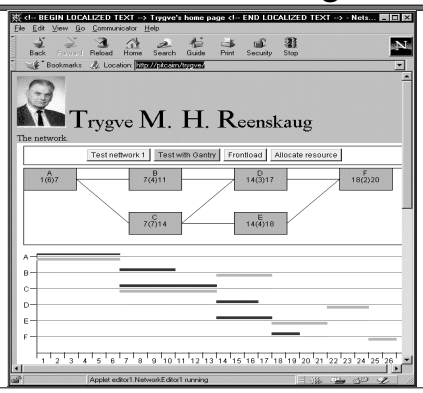
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 28

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**

Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 29

Enterprise Production Planning & Control

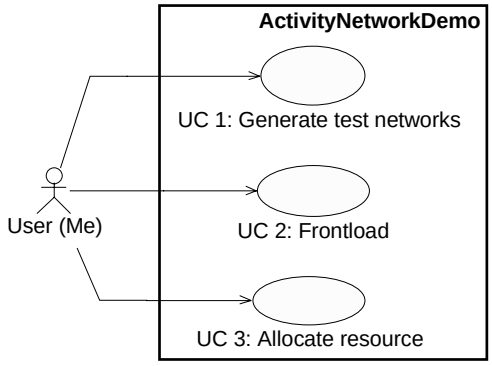


Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 30

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.

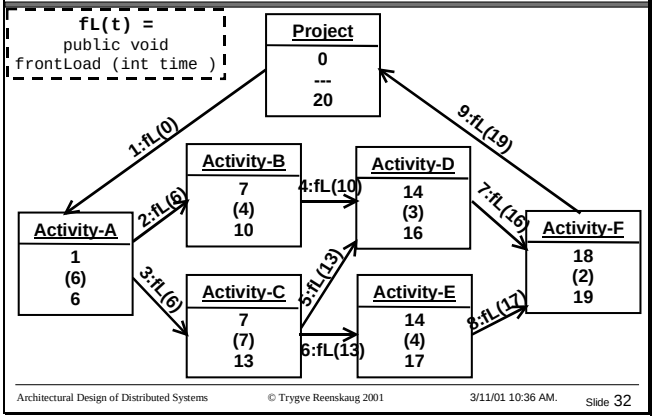
The top bars in the bar chart shows the earliest times when the activities can be performed. The bottom bars show when the activities all have to be done by the same resource, and this resource can only serve one activity at the time.

UML Use Case Notation *Demo Example*

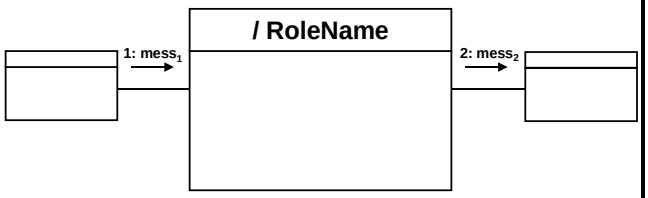


Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:36 AM. Slide 31

Use Case 2: Frontloading UML Instance Interaction Diagram

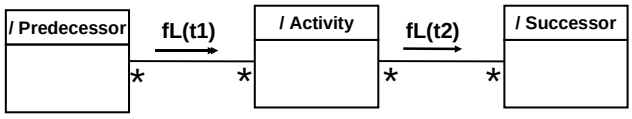


The UML Collaboration and the ClassifierRole



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 essence of frontloading Collaboration with Interaction



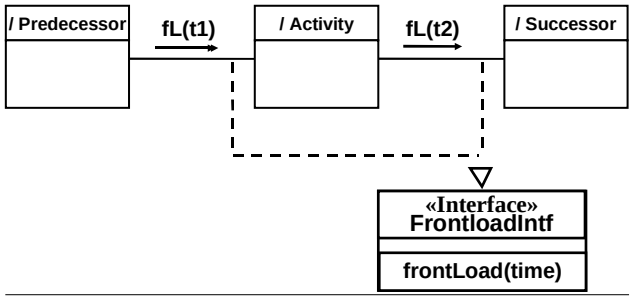
"In an instance of a collaboration, each ClassifierRole maps onto at most one object."

A many-relation (*) means that the role represents a typical member of the set All other members behave correspondingly

What are the objects, what are their responsibilities, and how do their collaborate to achieve the system functionality.

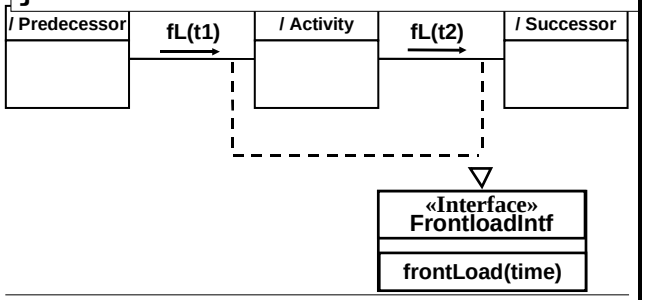
Interfaces in the Basic scheduling collaboration

The interfaces specify minimal requirements to receiving classes

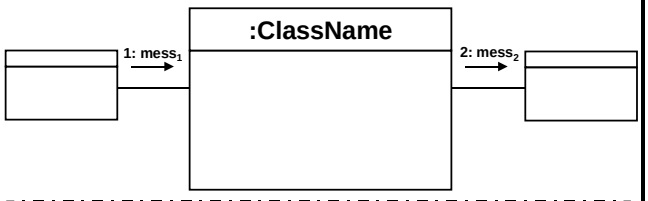


Interface Definitions *Java*

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

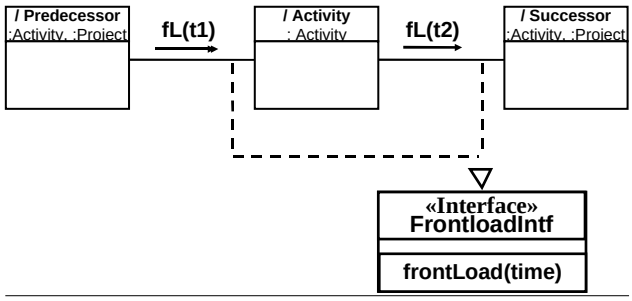


The UML Class *playing a role in a Collaboration*



An **Instance** of a given class playing the specified role at run time

Assign classes to roles *(Optional)*



Class Definitions *Java*

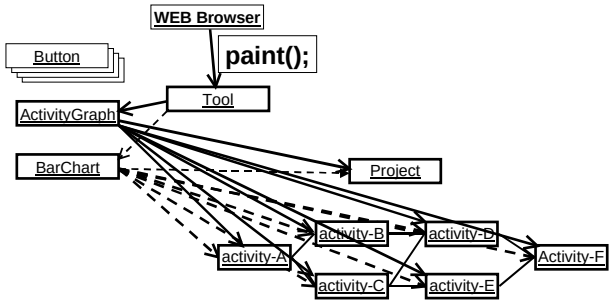
```
public class Activity
   implements FrontloadIntf {
   private FrontloadIntf[] successors;
   … … …
   public void frontload (int t) {…}
   … … …
}
```

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


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*

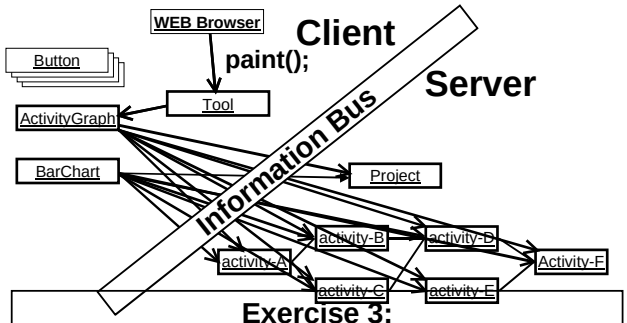


An Unrealistic Implementation

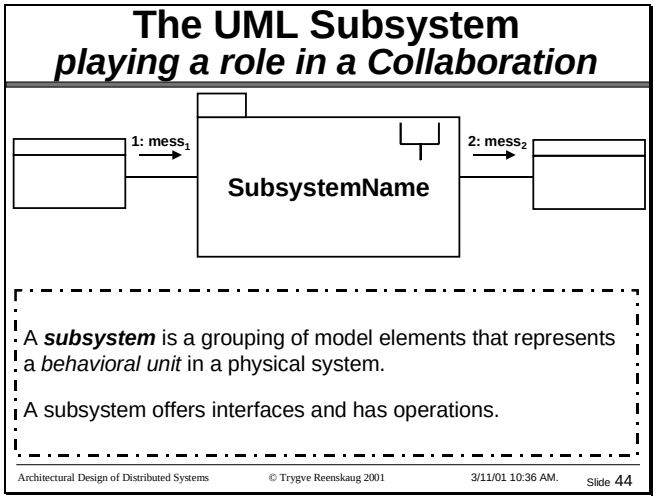
```
class ActivityGraph extends Canvas {. . .
public void paint(Graphics g) {
       . . . // paint frame etc.
       Graphics ng = g.create(. . .);
       tool.getProject().paintGraph(ng);
}
```

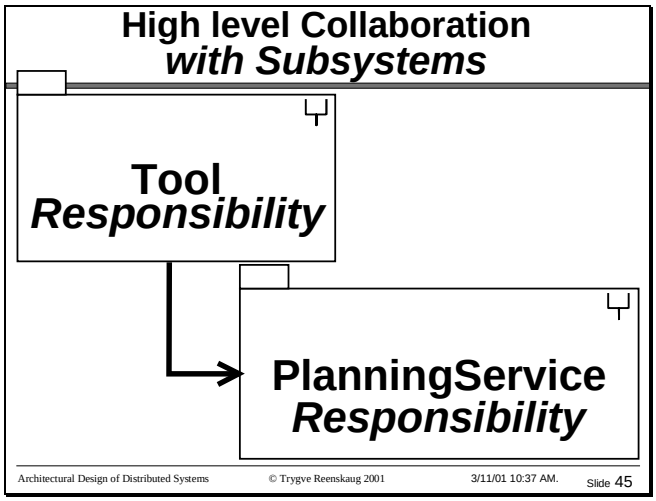
```
public class Project {. . .
public void paintGraph(Graphics g) {
     // collect ranked activities
     Activity[][] bucket = rankedActivities();
     // Plot activities in each column.
     for (int i=0; i < bucket.length; i++) {
          for (int j=0; j < bucket[i].length; j++) {
               bucket[i][j].paintSymbol(g, new Point (…), grid);
          }
     }
     // draw connections . . .
```


Java RMI *Remote Method Invocation*

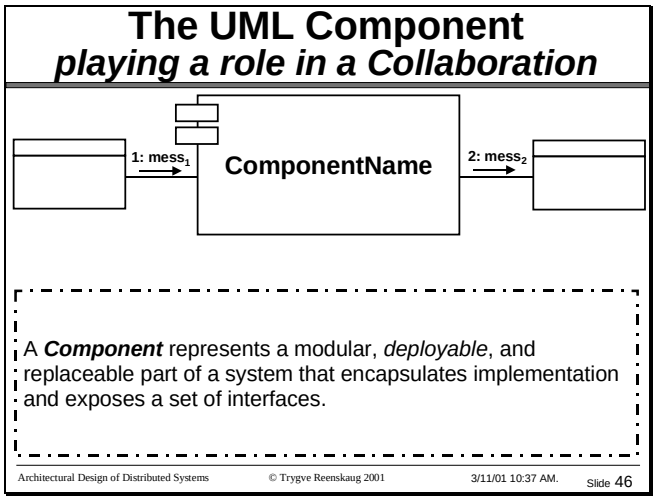


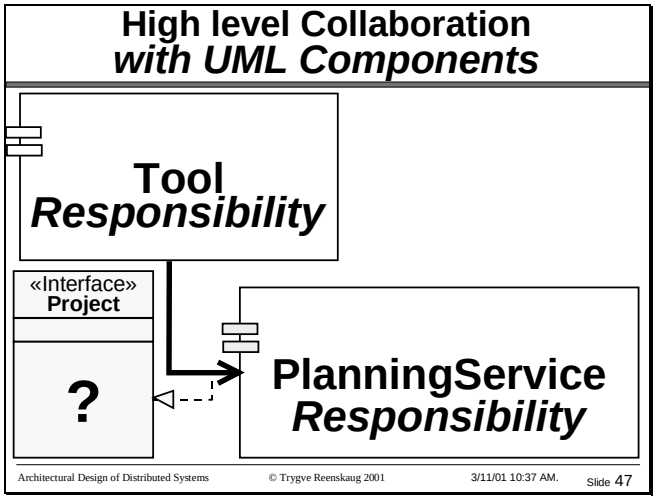
Exercise 3:
Suggest an implementation for this system



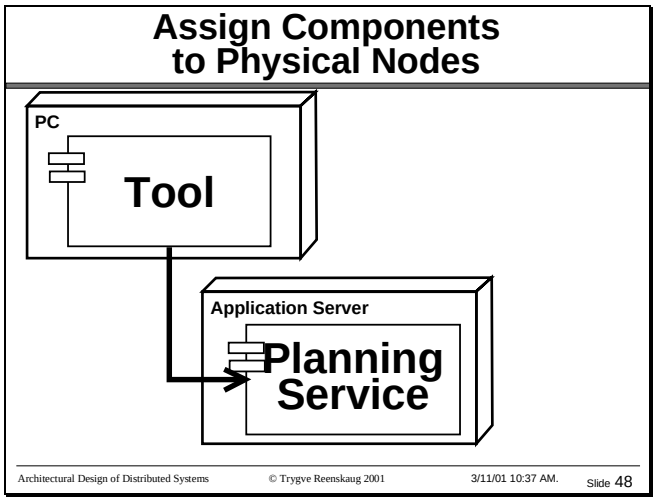


The Subsystem hides detail to simplify the drawing, but is neither visible at compile time nor at run time.

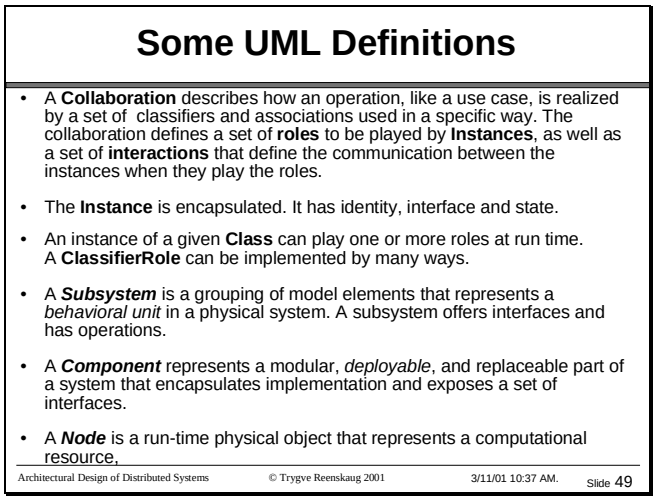




The Component simplifies the system. It supports separate compilation and deployment; the separation is visible at run time.



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



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

ChristopherAlexander, 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

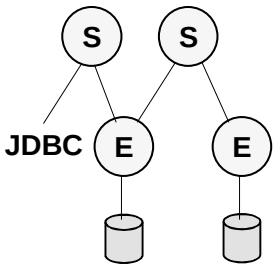
- **Why architecture is important & What it is**
- **Habitable Architectures created for people**
  - *Enterprise models*
  - *Personal information environments*
  - *System models, the heavy part*
- **UML Collaboration, a powerful abstraction for architectural topology**
  - *Collaboration (role model)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

Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:37 AM. Slide 53

Wrap Entity Beans with Session Beans-1

Ed Roman, TheServerside.com, August, 2000

"Artist's Impression"

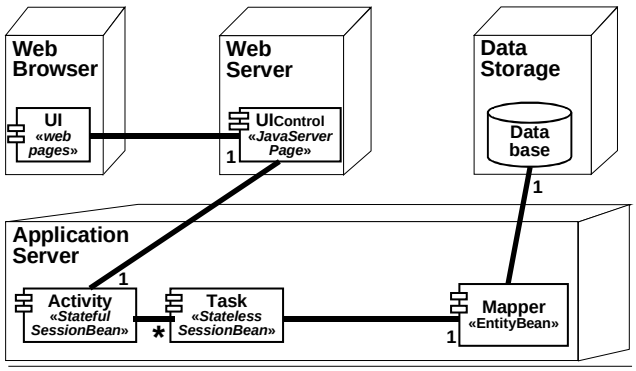


Business API
Business Logic

Persistent Data
Data Logic

Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:37 AM. Slide 54

A Real Example Application Execution Architecture

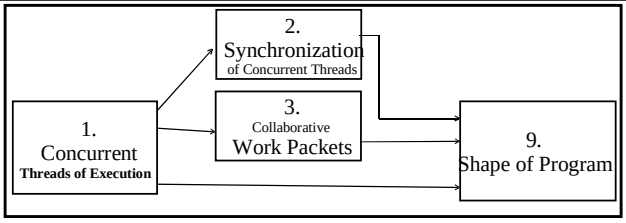


Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:37 AM. Slide 55

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

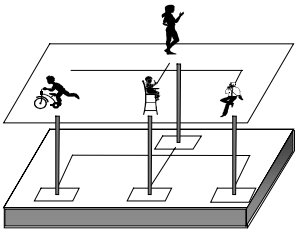


The main patterns

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**

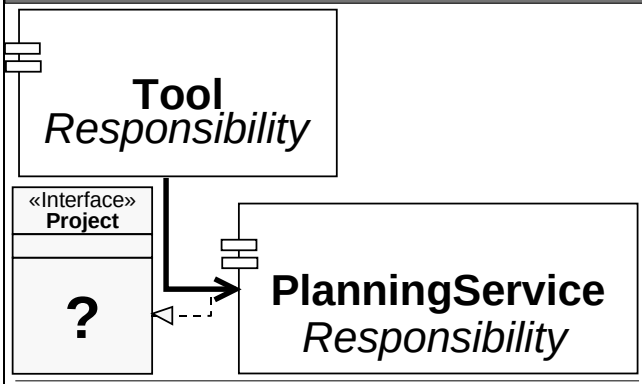
Personal, Integrated Information Environments



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

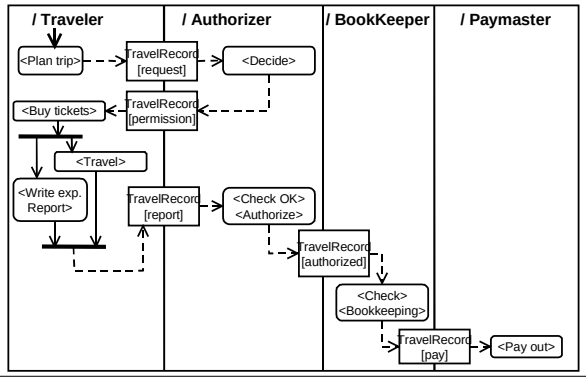
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:37 AM. Slide 59

Control Components through their Responsibilities and Interfaces



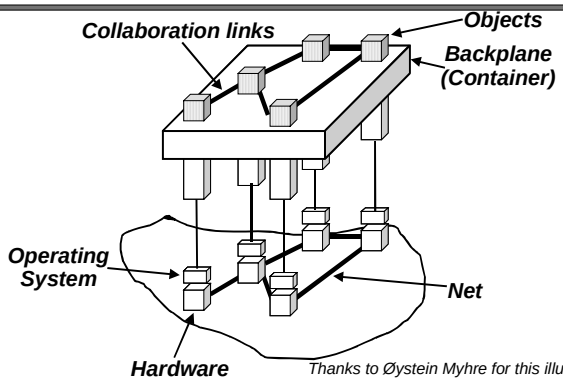
Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:37 AM. Slide 60

Plan Behavior Who - What - When



Architectural Design of Distributed Systems © Trygve Reenskaug 2001 3/11/01 10:37 AM. Slide 61

Instruct implementors and maintainers



Thanks to Øystein Myhre for this illustration

Questions ?
Comments ?

THE END

More details

- http://www.ifi.uio.no/~trygver
- trygve.reenskaug@ifi.uio.no
- Coplien, James O. and Douglas C. Schmidt, ed. Pattern Languages of Program Design, Addison-Wesley, 1995.
- Reenskaug, Wold, Lehne: *Working With Objects.* Manning/Prentice Hall 1996. ISBN 0-13-452930-8
  The reference work. This book is out of print. A .pdf version kan be downloaded free from above website.
- Theory: Egil P. Andersen: *Conceptual Modeling of Objects. A Role Modeling Approach.* Dr Scient thesis. Dept. of Informatics, University of Oslo. 4 November 1997.
  The theory of role modeling
  ftp://ftp.nr.no/pub/egil/ConceptualModelingOO.ps.gz
- *Unified Modeling Language (UML).* Object Management Group. UML Draft Specification v. 1.4. UML Draft Specification v. 1.4. OMG DOC#: ad/01-02-13.
  http://cgi.omg.org/cgi-bin/doc?ad/01-02-13
- Donald A. Norman: "The Design of Everyday Things." Doubleday/Currency 1990. ISBN 0-385-26774-6.
- Douglas MacGregor: “Human Side of Enterprise : 25th Anniversary Printing” McGraw-Hill 1985; ISBN: 0070450986
- Alexander, Christopher, et al. A Pattern Language, Oxford University Press, New York, 1977.
- http://www.c2.com/cgi/wiki?WelcomeVisitors
- http://hillside.net/patterns/
- http://theserverside.com/patterns/
- http://www.c2.com/cgi/wiki?EjbRoadmap
- Frank Buschmann: *Pattern-Oriented Software Architecture.* Wiley 1996; ISBN: 0471958697
- IBM WebSphere: http://www.redbooks.ibm.com/abstracts/sg246161.html