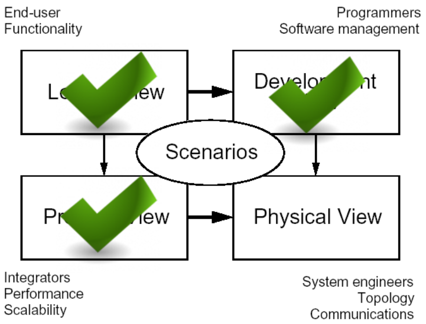
Documenting Software Architecture

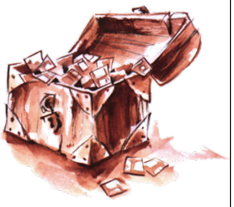
4+1 view model continue…

4+1 View model

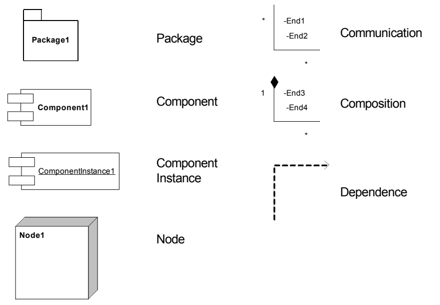


The Physical View: System engineers

- √ Focus on non-functional requirements:
  - Availability (Reliability), Performance (Scalability).
- √ Software executes on a network of processing nodes.
  - Networks, processes, tasks and objects need to be mapped onto nodes.
- √ Usual with different physical configurations:
  - In development phase, in deployment phase for various sites and customers.
  - Architecture must be highly flexible and minimal impact on source code.



Notation for Physical view (UML)



Package

Component

Component Instance

Node

Communication

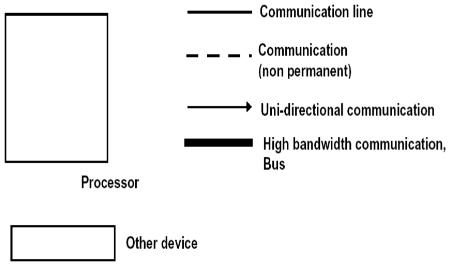
Composition

Dependence

Notation for Physical View

Components

Connectors



Communication line

Communication (non permanent)

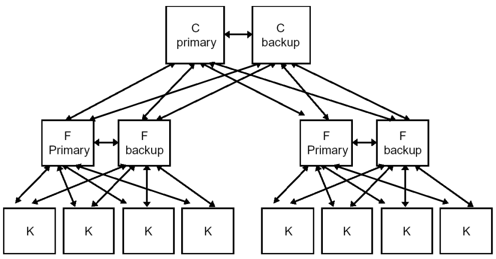
Uni-directional communication

High bandwidth communication, Bus

Processor

Other device

Example: Physical View of Phone control



C primary

C backup

F Primary

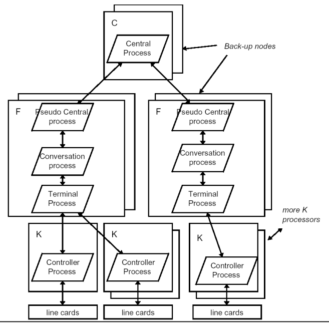
F backup

F Primary

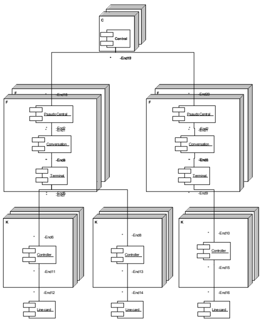
F backup

K K K K K K K K

Example (UNAS): Process View mapped on Physical View of Phone control

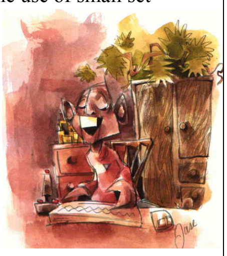


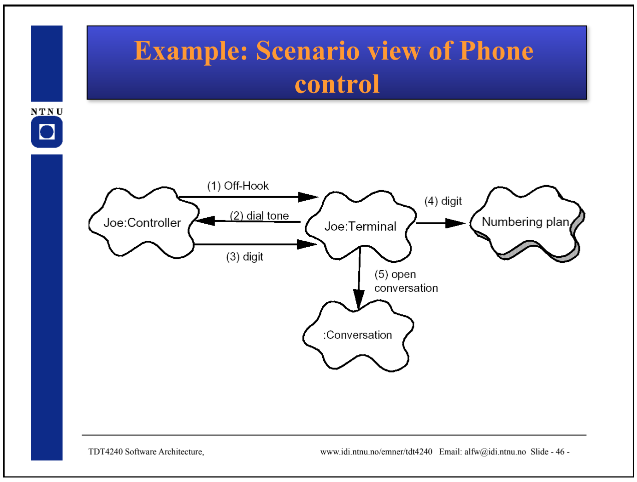
Example (UML): Process View mapped on Physical View of Phone control



Scenarios

- √ Four views can work together by the use of small set of scenarios.
- √ The scenarios are an abstraction of most important requirements.
- √ Diagrams: Use case / interaction.
- √ Scenario view is an add-on, and serves two main purposes:
  - Driver to discover the architectural elements during architecture design.
  - Validation and illustration role when architecture design is complete.





Example: Scenario view of Phone control

Joe:Controller

(1) Off-Hook

(2) dial tone

(3) digit

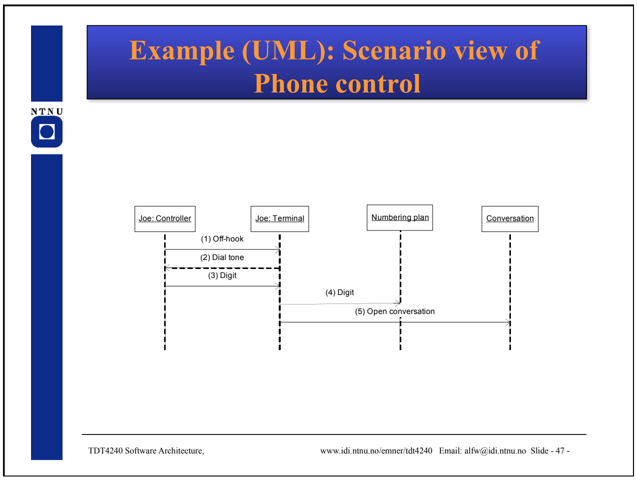
Joe:Terminal

(4) digit

Numbering plan

(5) open conversation

:Conversation



Example (UML): Scenario view of Phone control

Joe: Controller

Joe: Terminal

Numbering plan

Conversation

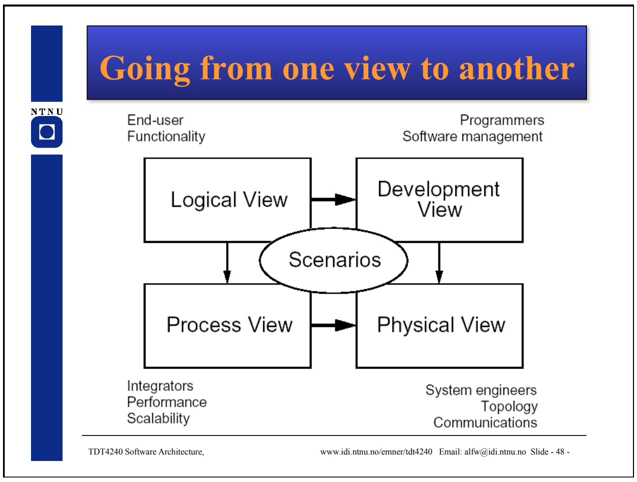
(1) Off-hook

(2) Dial tone

(3) Digit

(4) Digit

(5) Open conversation



Going from one view to another

End-user
Functionality

Programmers
Software management

Logical View

Development View

Scenarios

Process View

Physical View

Integrators
Performance
Scalability

System engineers
Topology
Communications

From the logical to the process view

√ Identify several important characteristics of classes:
- **Autonomy:** Are the objects active, passive, protected?
- **Persistence:** Are the objects transient, permanent?
- **Subordination:** Do an object depend on another object?
- **Distribution:** Are the state or operations of an object accessible from many nodes in the physical architecture?

√ **In logical view:** Consider an object as active and concurrent.

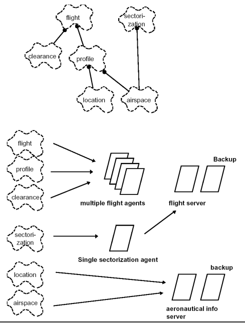
√ **In process view:** Must determine the “right” amount of concurrency and define the set of processes needed.

II: From the Logical to the Process View

√ **Inside-out:**
1. *Define agent tasks* which multiplex a single thread of control across multiple active objects of a class.
2. *Subordinated object* to active objects are executed on the same agent.
3. Mutual exclusive or light-weight classes share a single agent.
4. Continue clustering until reasonably small number of processes that allows distribution and use of physical resources.

√ **Outside-in:**
1. Identify *external stimuli* (requests) to the system.
2. Define *client processes* to handle the stimuli.
3. Define *server processes* that only provide services.
4. Use data integrity and serialization constraints of the problem to *define the right set of servers*.
5. *Allocate objects* to the client and server agents.
6. Identify which objects must be *distributed*.

Example: From Logical to Process View



From Logical to Development View

- √ A Class is usually implemented as a package.
- √ Large classes are decomposed into multiple packages.
- √ Collections of closely related classes are grouped into subsystems.
- √ Constraints considerations that distinguish the logical from implementation views:
  - Team organization.
  - Expected magnitude of code.
  - Degree of expected reuse and commonality.
  - Strict layering principles.
  - Release policy.
  - Configuration management.

From Process to Physical View

- √ Processes and process groups are mapped onto available physical hardware.
- √ Must consider various configurations for testing or deployment.

Process for Architecture Design: A scenario-driven approach:

- √ Start:
  - Choose small number of scenarios based on risk and criticality.
  - Create a strawman architecture.
  - Describe the architectural elements discovered in four views.
  - Implement, test and measure architecture prototype.
  - Capture lessons learned.
- √ Loop: Start by reassessing risk, adding scenarios.
  - Discover additional architectural elements or architectural changes.
  - Update 4 views: Logical, process, development, physical.
  - Revise existing scenarios based on changes.
  - Upgrade, test and measure architecture prototype.
  - Review/revise five views.
  - Design guidelines and rationale are updated.
  - Capture the lessons learned.
- √ End loop

4+1 View Model Summary

View	Logical	Process	Development	Physical	Scenarios
Components	Class	Task	Module, Subsystem	Node	Step, Scripts
Connectors	Association, inheritance, containment	Redez-vous, Message, Broadcast, RPC, etc.	Compilation dependency, "with" clause, "include"	Communication medium, LAN, WAN, bus etc.	
Containers	Class category	Process	Subsystem (library)	Physical subsystem	Web
Stakeholders	End-user	System designer, integrator	Developer, manager	System designer	End-user, developer
Concerns	Functionality	Performance, availability, SW fault-tolerance, integrity	Organization, reuse, portability, line-of-product	Scalability, performance, availability	Understand-ability

Conclusion

- √ *Quality attribute* of non-trivial systems are *determined by its architecture.*
- √ Software architecture is a *vehicle for communication* among the stakeholders.
- √ Architecture can be described through *several views.*
- √ *Views represent the position* from where stakeholders see the architecture.