Modeling Real-Time System Architectures with UML 2.0

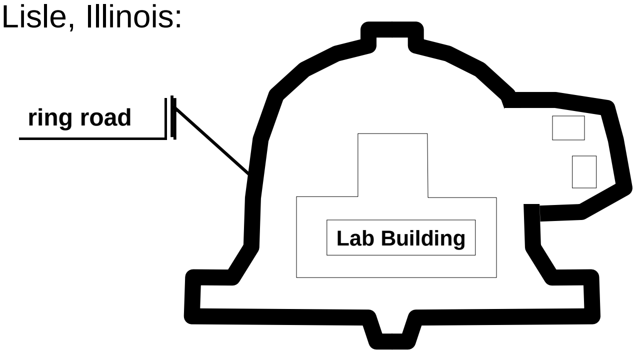
Bran Selic
Principal Engineer
IBM Software Group –
Rational Software
bselic@ca.ibm.com

Objectives

- Define architecture and its role in software design
- Identify requirements for modeling software architectures
- Describe how UML 2 and MDA can be used for modeling software architectures of embedded systems

A Parable

Lisle, Illinois:



- “Architectural” decay, caused by:
  - Lack of high-level (architectural) view (“the forest vs the trees”)
  - Difficulties in enforcing architectural decisions

3

Presentation Overview

- Software architecture and its role
- Requirements for architectural modeling
- The role of UML 2 and MDA in architectural modeling
- Example system
- Architectural patterns for embedded software

4

(Run-Time) Architecture

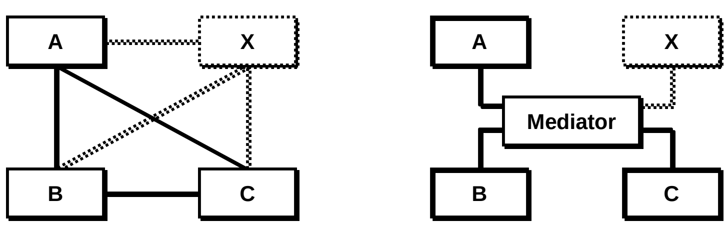
- An abstract view of a system that identifies only the important elements and relationships
- We will focus only on run-time architectures:

  The run-time organization of <u>significant</u> software components interacting through interfaces, those components being composed of successively smaller components and interfaces

5

Why Architecture is Important

- Enables communication between stakeholders
  - exposes how individual requirements are handled
- Drives system construction
  - decomposition into units of responsibility and parallel development
- Determines a system's capacity for evolutionary growth



6

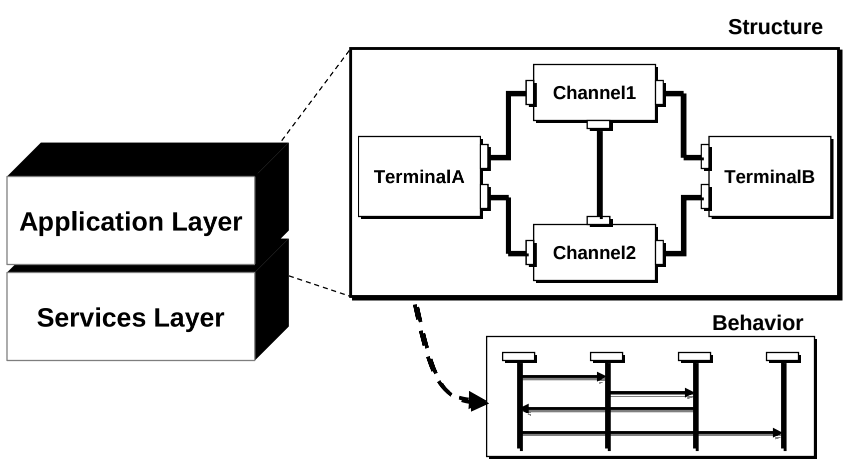
Presentation Overview

- ♦ Software architecture and its role
- ♦ Requirements for architectural modeling
- ♦ The role of UML 2 and MDA in architectural modeling
- ♦ Example system
- ♦ Architectural patterns for embedded software

7

Example Complex Architecture Spec

- ♦ Example telecom system architecture



8

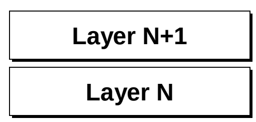
Basic Run-Time Architectural Patterns

- Peer-to-peer communication:

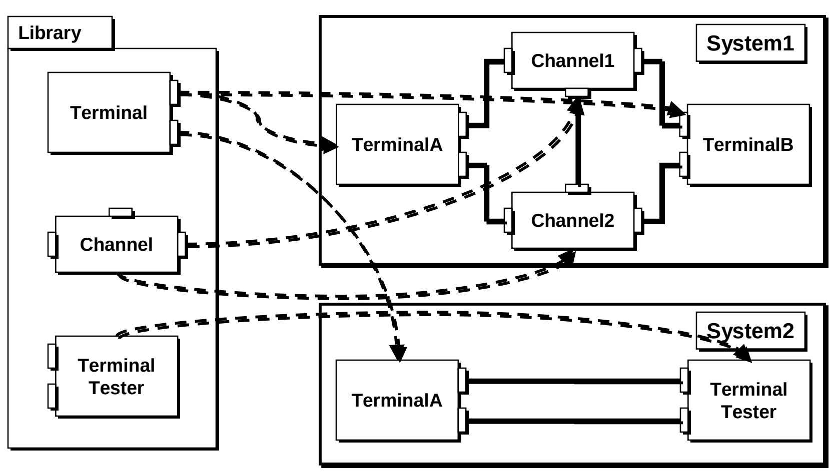
- Containment:

aggregation (information hiding)

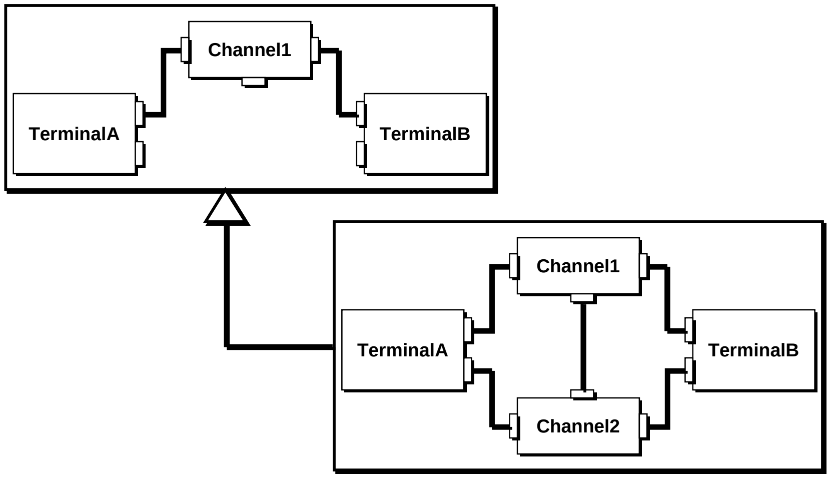
- Layering



Architectural Component Design



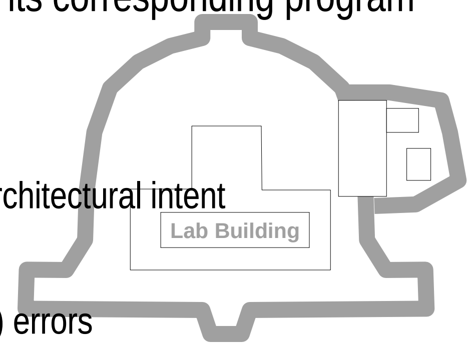
Refining Architectures (Reuse)



11

Architectural Decay

- The (usually) gradual divergence between an architectural model and its corresponding program implementation
- Caused by:
  - Misunderstandings of architectural intent
  - Design disagreements
  - Implementation (coding) errors
- Often occurs during low-level maintenance work



12

Summary of Requirements

- Need the ability to specify software architectural patterns in a direct and and expressive way
- Need to provide reuse of architectural-level components
- Need to prevent architectural decay

13

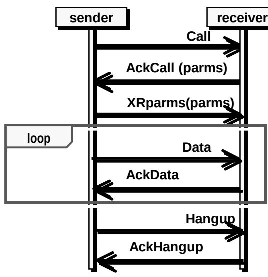
Presentation Overview

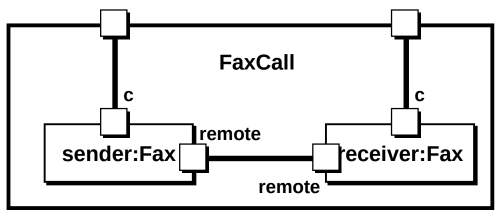
- Software architecture and its role
- Requirements for architectural modeling
- The role of UML 2 and MDA in architectural modeling
- Example system
- Architectural patterns for embedded software

14

UML 2.0 and Architectural Modeling

- UML 2.0 adds two fundamental capabilities for modeling software architectures
  - *Structured classes* for modeling structural aspects:
    - parts, ports, and connectors
    - reusable architectural components
  - *(Complex) interactions* for modeling behavior





What About Architectural Decay?

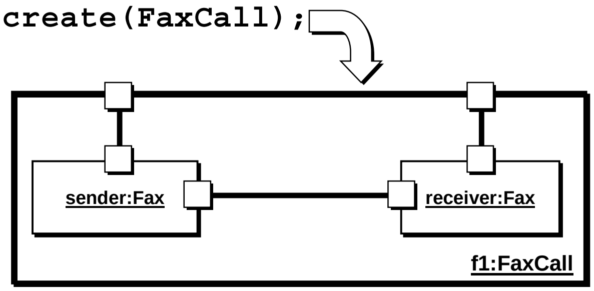
- Ensure visibility and enforcement of architectural intent
- Achieved by:
  - Requiring that *all* design work to take place at the model level
  - Automatically generating implementations directly from models

…i.e., use *model-driven development*

Structured Object Semantics

- Run-time assertion: the complete internal structure of a composite is automatically created (recursively, if necessary) when the object is created

```
f1 := create(FaxCall);
```



17

Benefits of Run-Time Assertion

- *Architectural enforcement:* only explicitly prescribed architectural structures can be instantiated
  - it is not possible to bypass (corrupt) the architecture by low-level programming
- *Simplification:* low-level program code that dynamically creates (destroys) components and the connections between them is eliminated
  - in some systems this can be as much as 35% of all code
- Major net gain in productivity and reliability

18

Summary: UML as an ADL

- UML 2.0 seems to have the necessary level of expressive power to specify software architectures directly (= architectural description language)
- By supporting the class/object paradigm at the architectural level it enables definition of reusable architectural components
- Combined with Model-Driven Development (MDD) methods, it can also ensure architectural enforcement

Presentation Overview

- Software architecture and its role
- Requirements for architectural modeling
- The role of UML 2 and MDA in architectural modeling
- Example system
- Architectural patterns for embedded software

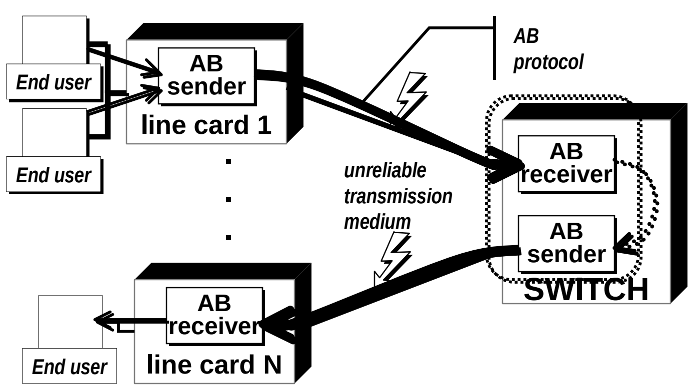
Design Patterns

- *A **design pattern** is a proven generalized solution to a generalized problem that can be used to derive a specific solution to a specific problem*
- Represent distilled reusable experience
- Major benefits of using patterns:
  - Simplify and speed-up design
  - Reduce risk
  - Facilitate communications between designers

21

Example System

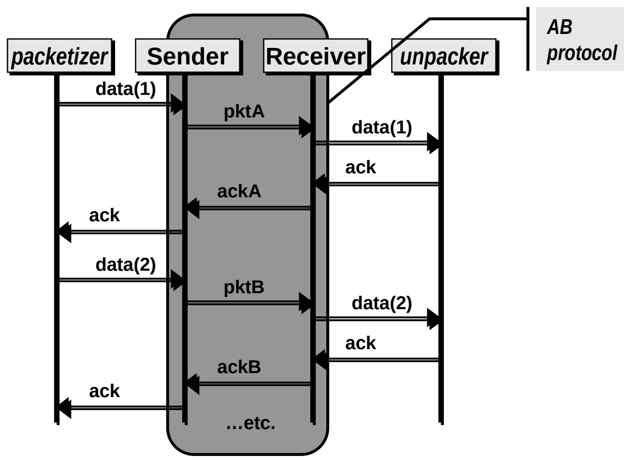
- A multi-line packet switch that uses the alternating-bit protocol as its link protocol



22

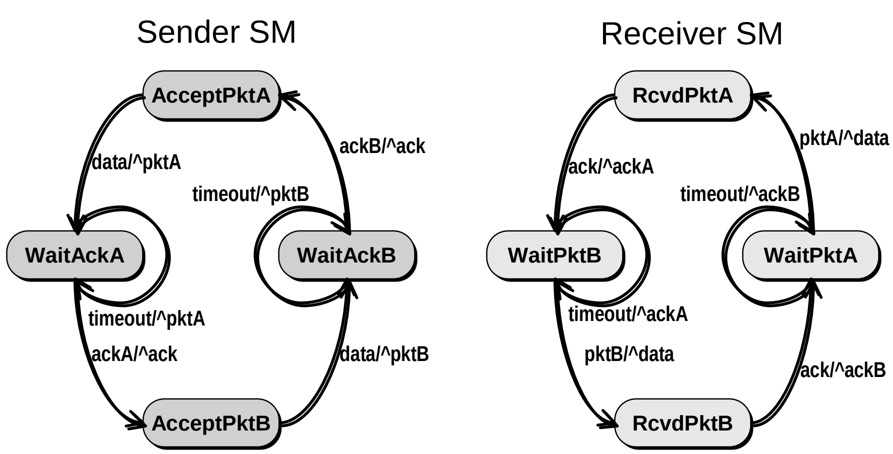
Alternating Bit Protocol (1)

♦ A simple one-way point-to-point packet protocol



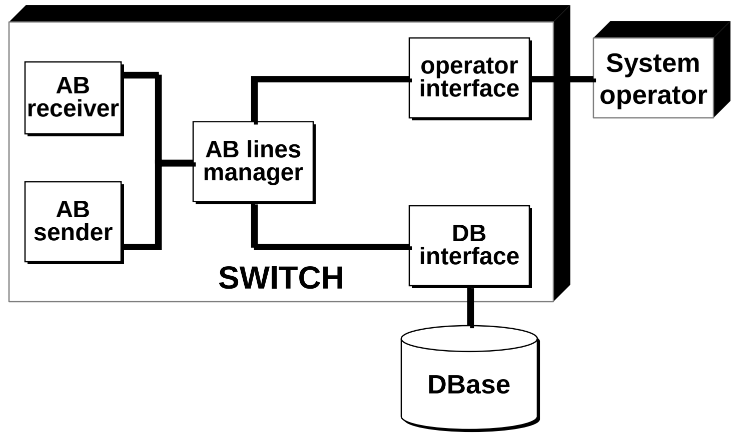
Alternating Bit Protocol (2)

♦ State machine specification



Additional Considerations

- Support infrastructure



25

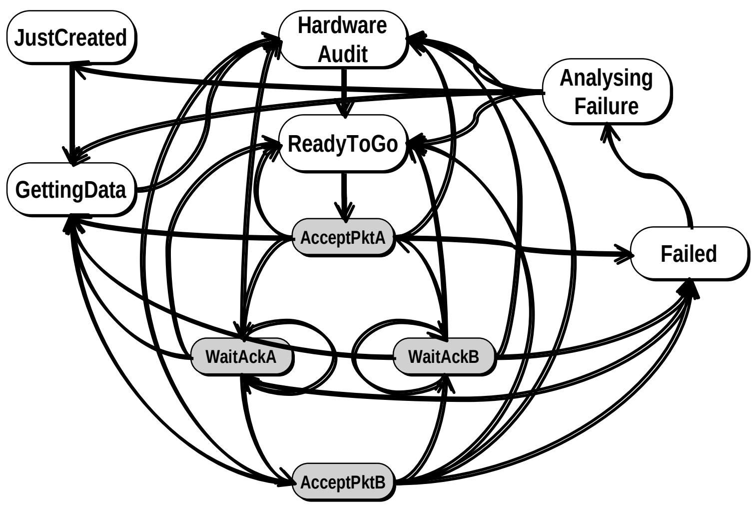
Control

The set of (additional) mechanisms and actions required to bring a system into the desired operational state and to maintain it in that state in the face of various planned and unplanned disruptions

- For software systems this includes:
  - system/component start-up and shut-down
  - failure detection/reporting/recovery
  - system administration, maintenance, and provisioning
  - (on-line) software upgrade

26

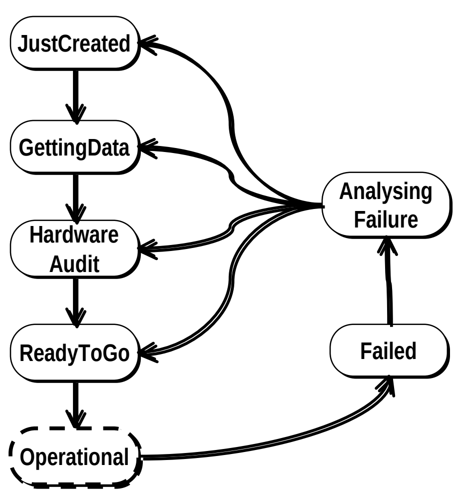
Retrofitting Control Behavior



27

The Control Automaton

- In isolation, the same control behavior appears much simpler



28

Control versus Function

- Control behavior is often treated in an ad hoc manner, since it is not part of the primary system functionality
  - typically retrofitted into the framework optimized for the functional behavior
  - leads to controllability and stability problems
- *However, in highly-dependable systems as much as 80% of the system code is dedicated to control behavior!*

29

Some Important Observations

- *Control predicates function*
  - before a system can perform its primary function, it first has to reach its operational state
- *Control behavior is often independent of functional behavior*
  - the process by which a system reaches its operational state is often the same regardless of the specific functionality of the component

30

Presentation Overview

- ♦ Software architecture and its role
- ♦ Requirements for architectural modeling
- ♦ The role of UML 2 and MDA in architectural modeling
- ♦ Example system
- ♦ Architectural patterns for embedded software

31

The Recursive Control Architectural Pattern

32

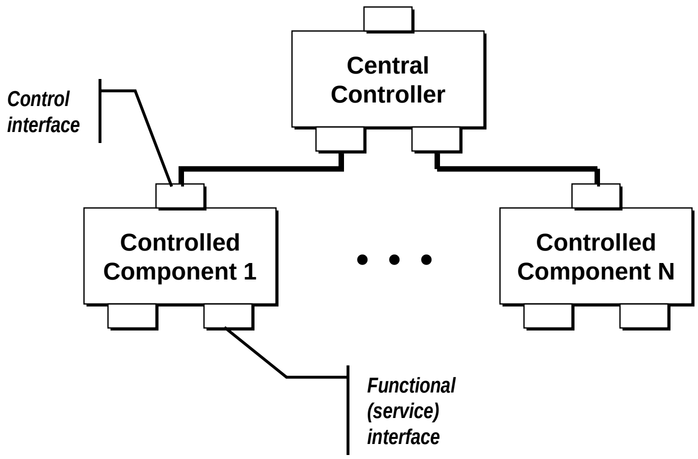
Basic Design Principles

- *Separate control from function*
  - separate control components from functional components
  - separate control interfaces from functional interfaces
  - imbed functional behavior within control behavior
- *Centralize control (decision making)*
  - if possible, focus control in one component
  - place control policies in the control components and control mechanisms inside the controlled components

33

The Basic Structural Pattern

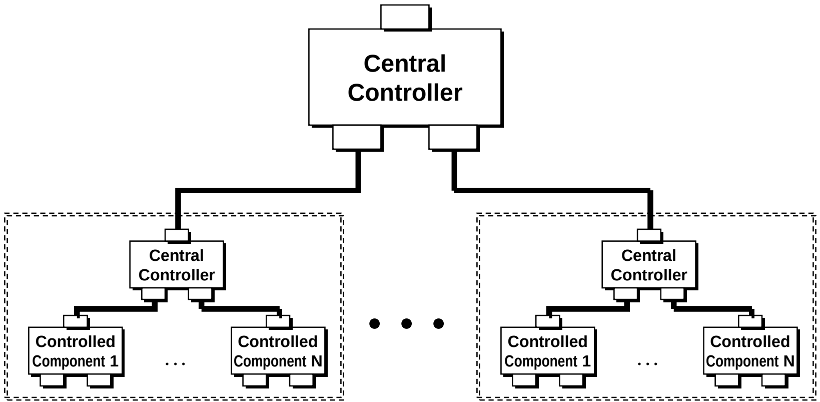
- Set of components that need to be controlled in a coordinated fashion



34

Recursive Application

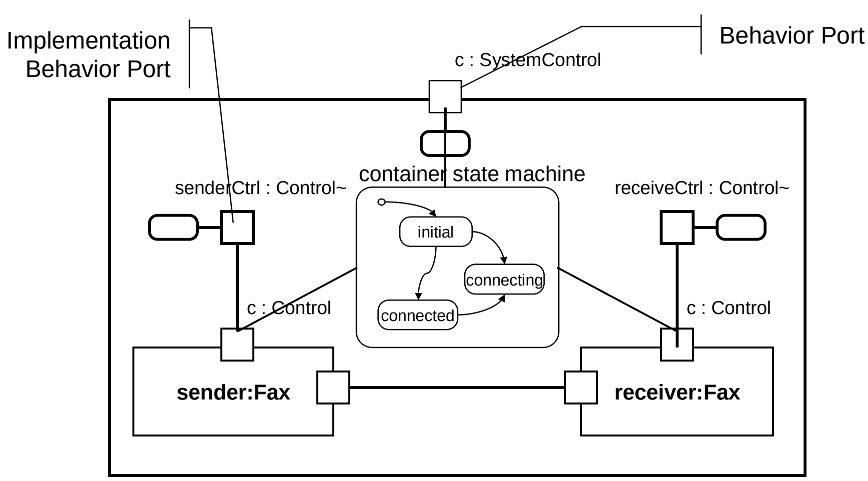
- Hierarchical control
  - scales up to arbitrary number of levels



35

Behavior Ports

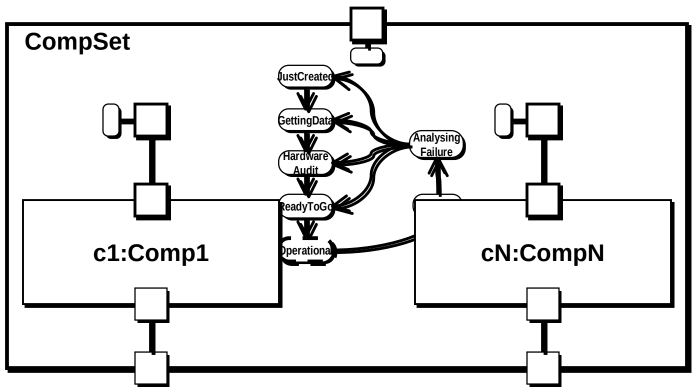
- Ports directly connected to the state machine



36

Realization with Ports and Objects

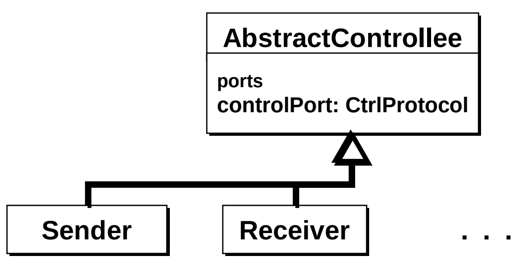
- Composite plays role of centralized controller



37

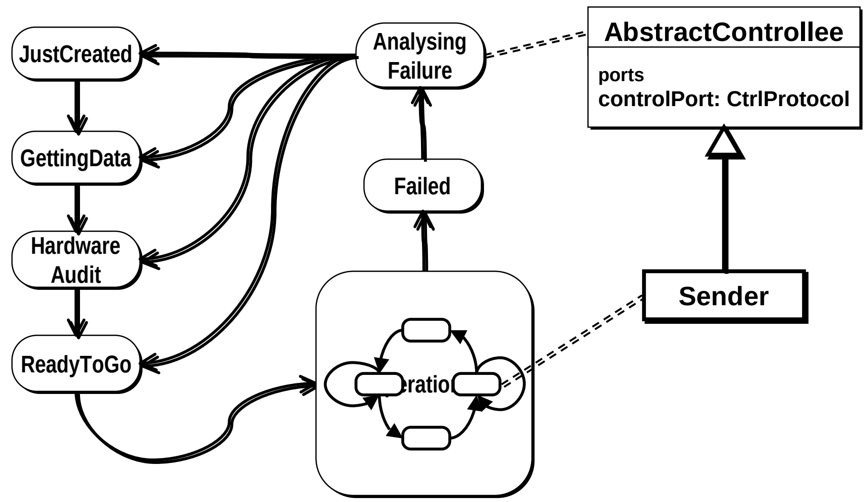
Exploiting Inheritance

- Abstract control classes can capture common control behavior and structure
- Different subclasses capture function-specific behavior



38

Exploiting Hierarchical States



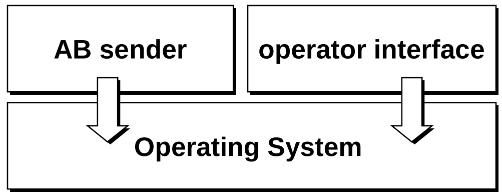
39

The Run-Time Layering Architectural Pattern

40

Semantics of Layering (1)

- A fundamentally different type of structural relationship

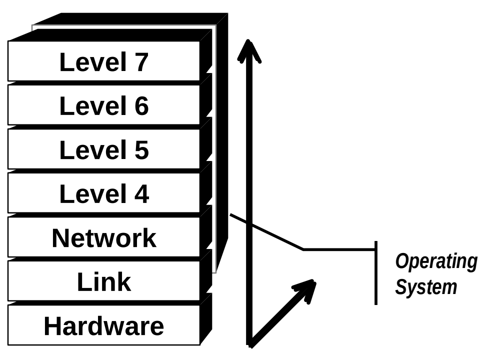


- Layering is different from containment
  - Higher-layers do not contain lower layers
  - Formally, the lower layers "contain" the higher layers (existence dependency) but they do not encapsulate them

41

Semantics of Layering (2)

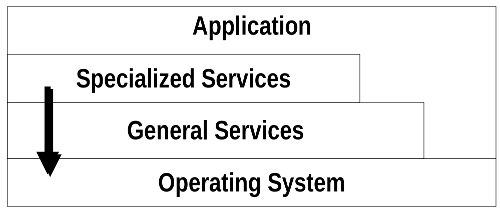
- In complex systems, layering is a complex multidimensional relationship
  - e.g., 7-layer model of Open System Interconnection (OSI)



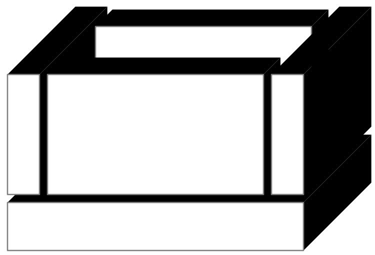
42

Inadequate Representations of Layering

- Staircase model



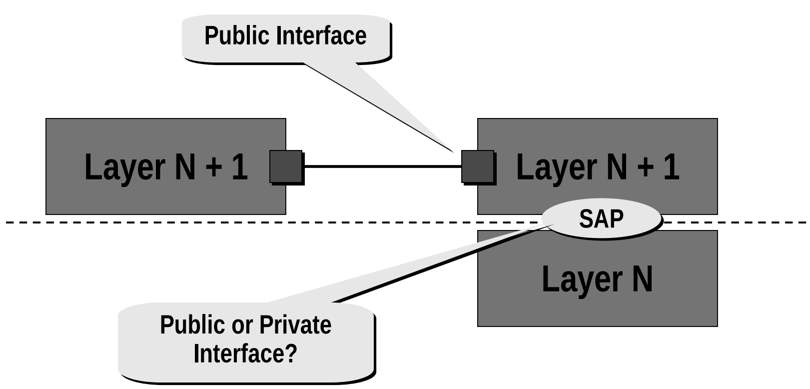
- Toaster model



43

More on the OSI Model

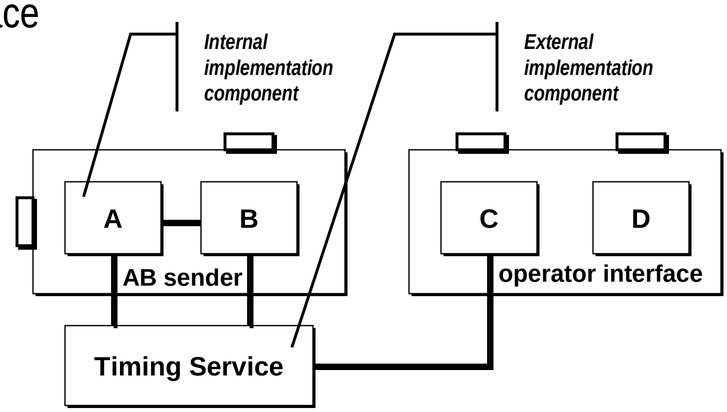
- Two distinct kinds of interfaces: peer and SAP



44

Implementation Components

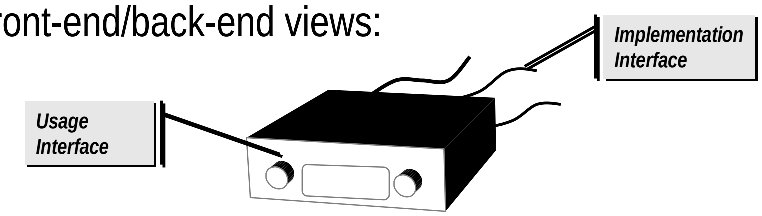
- Private sub-components required to realize the functionality offered by component through its public interface



45

Interface Types for Layering

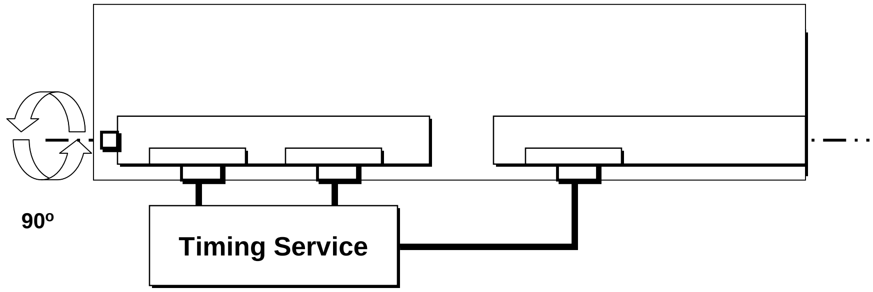
- Need to differentiate two interface types:
  - *Usage interface*: implementation-independent interface through which a component provides its services (function and control)
  - *Implementation interface* (service access point): implementation-specific interface through which a component accesses an external service
- Front-end/back-end views:



46

Implementation Interfaces

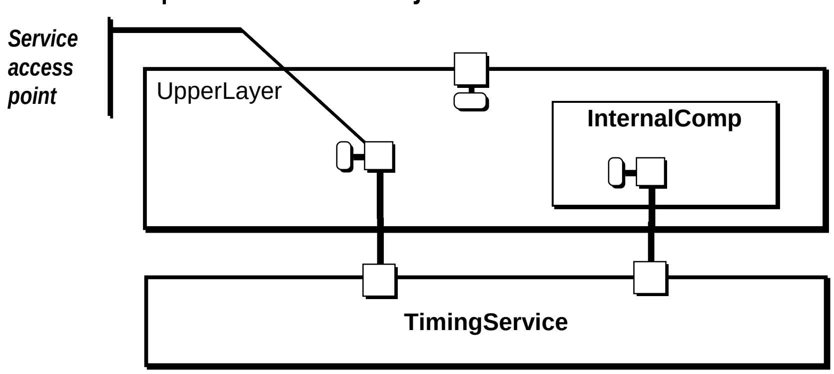
- Implementation interfaces are public interfaces but can be viewed as being in a different "plane" (dimension) from service interfaces



47

Modeling Layers with Ports and Objects

- Implementation interfaces are modeled by implementation end ports that can be connected directly to service ports of other objects



48

Summary: Architectural Patterns

- Architecture plays a fundamental role in software design and evolution
- To prevent architectural decay we need to
  - Specify architectures directly, clearly, and easily
  - Enforce architectural specifications
- The combination of UML 2 and MDA enables us to meet these objectives
- We have demonstrated this ability by showing how these concepts can be used to model two high-level architectural patterns common in embedded system design

49