Robust Design Strategies

Gruia-Catalin Roman

October 2014

Department of Computer Science

University of New Mexico

Course Overview

1. Software Architecture Fundamentals
2. Software Architecture Specification
3. Robust Design Strategies

Robust Design Strategies

Chapter Overview

[3.1] Motivating factors

[3.2] Design principles

[3.3] Architectural styles

[3.4] Design patterns

3.1 Motivating Factors

- Systems are being delivered late and over budget
- Coding and testing dominate development time
- Maintenance and upgrade costs are too high
- Performance levels are inadequate
- Problem: software is treated differently from other kinds of products
  - an engineering mentality is needed
  - the intellectual tools used need to be specific to software development

An Engineering Perspective

- Analysis
  - we seek to understand the function and constraints of the product first
- Experimentation
  - we focus on risk factors and the unknown early in the process
- Planning
  - we use the product design to drive the planning process
- Problem solving
  - we approach the design in a systematic and predictable fashion

An Engineering Perspective

- Tradeoffs
  - we see the design as the art of making the right technical compromises
- Reuse
  - we reuse procedures, designs, and components and build things for reuse
- Evaluation
  - we subject both designs and products to targeted and extensive evaluations

Answers From the Past

An emphasis on programming and requirements

- High-level languages
- Structured programming
- Information hiding
- Top-down functional decomposition
- Structured analysis and design
- Rapid prototyping
- Formal specifications

Modern Thinking

Object-orientation as an enabling technology

- Facilitates conceptualization
  - powerful abstraction mechanisms
- Facilitates model construction
  - direct mapping to the physical world
- Promotes encapsulation
  - programming by contract
- Reduces development effort
  - programming by differences
- Embodies modern programming language thinking
- Promotes reuse

Object-Oriented Analysis

- OOA is a process of discovery and modeling
- Essential software requirements are captured in terms of the following key concepts

  object class relation

- Additional procedural abstractions are included to complete the model
- Graphical notation is commonly used to depict the structural aspects of the model
- It is often expected that the analysis will transfer to design and implementation

Object-Oriented Design

- OOD is a process of invention and adaptation
- The principal concern is the architecture of the software
- The approach is (generally) language independent
- The emphasis is on achieving certain structures having particular desirable properties
- The same concepts are used but the interest is in modularity
- Simple projection of real-world objects to software does not guarantee a good design
  - different domains of knowledge
  - constraints
  - concept versus module emphasis

Object-Oriented Programming

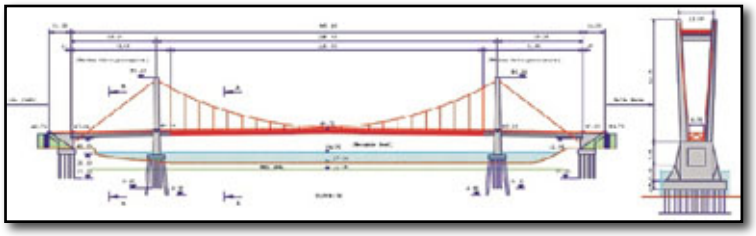
- OOP is a construction process
- It relies on the use of a programming language having many of the following features
  - objects as dynamically created instances of classes
  - classes (instance variables and methods)
  - inheritance (single and multiple)
  - generic classes (templates)
  - abstract classes and virtual methods
  - polymorphism
  - dynamic binding
  - exception handling

Beyond Object Orientation

- Object orientation thinking
  - is dominated by the concern with modular design
  - gained an emphasis on design reusability (patterns)
  - is seeking to address quality of service concerns
- Software architecture stresses two classes of issues
  - modular design
    - functionality coverage, structuring, and distribution
  - system level constraints
    - meeting non-functional requirements
- Constraints determine design complexity

Architecture Revisited

- The software architecture plays a critical role in placing software development on an engineering basis
- Software architecture design is the phase most akin to the design process taking place in other engineering disciplines
- It is the gap between paper and concrete that demands precision



3.2 Design Principles

- General design guidelines
  - modularity
  - encapsulation
  - protection against changes
  - constraint satisfaction
  - constraint evaluation
- Robust design
  - risk assessment
  - risk reduction
  - defensive design
  - defendable design
  - critical constraint driven

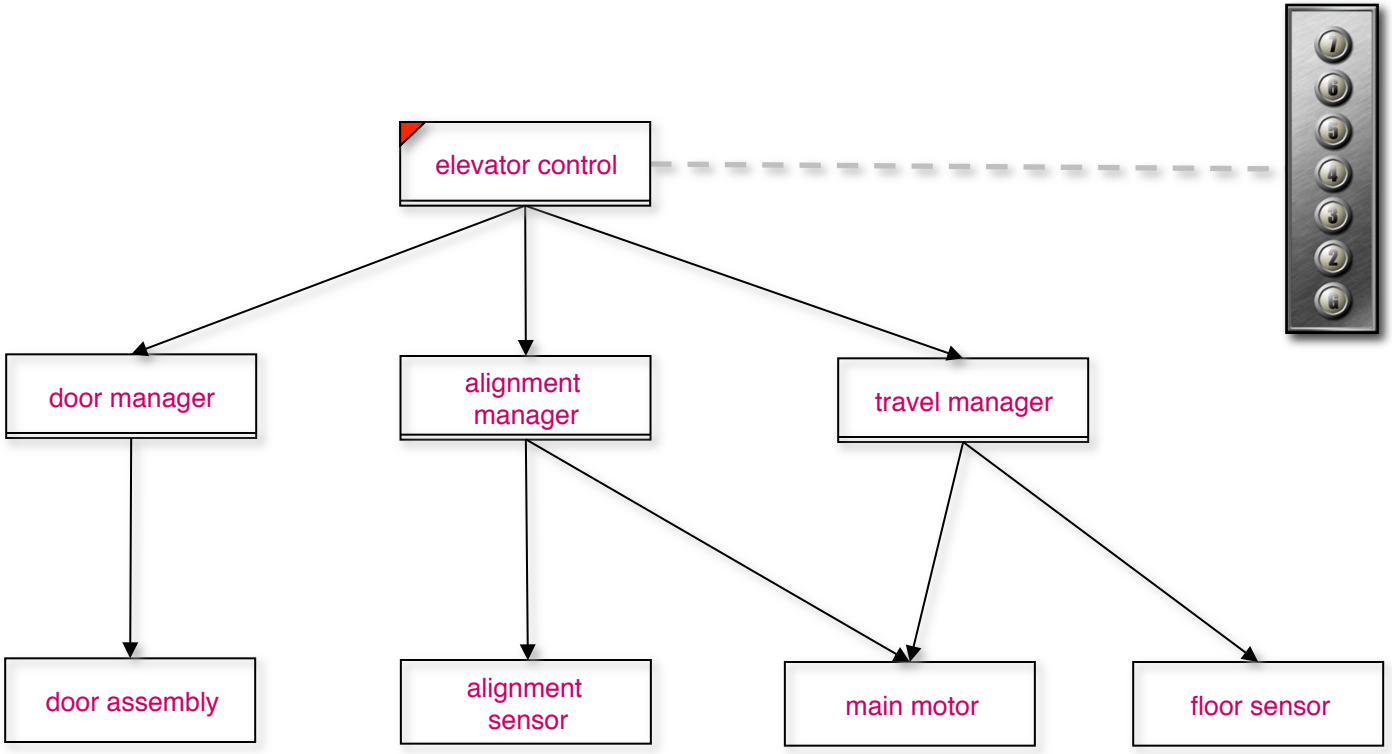
Design Strategy

- Identify the critical constraints
  - assess risk level
  - understand cost implications
- Separate the critical constraints
  - structurally
    - encapsulate at the level of module
  - behaviorally
    - encapsulate as a cross cutting constraint (aspect)
    - ensure independent analyzability
- Build and test along the critical path

Design Technique

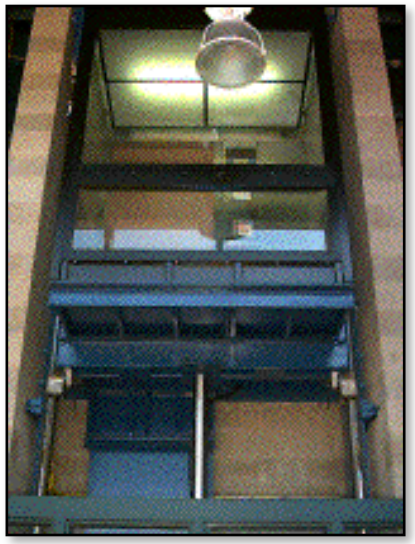
- Facilitators
  - static structures
  - simplicity where it counts
- Core concerns
  - high level of decoupling
  - integrated analysis and design

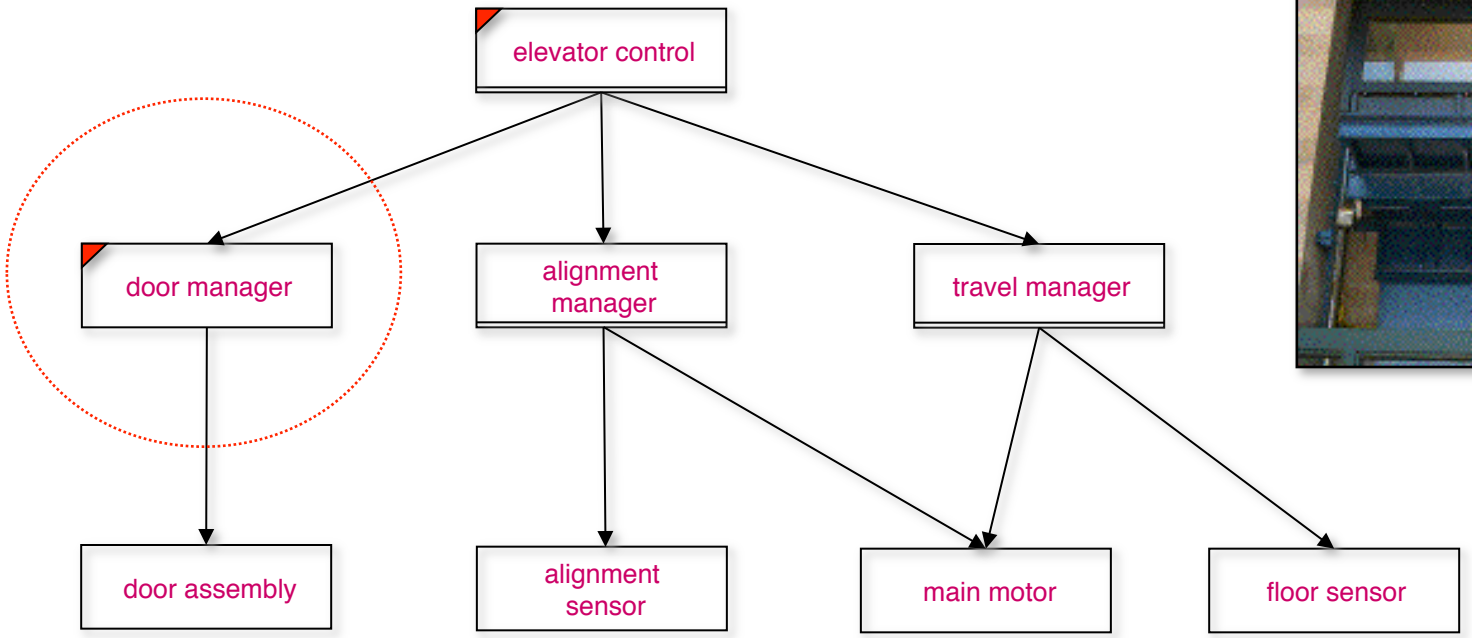
Modular Design



Responsive Design

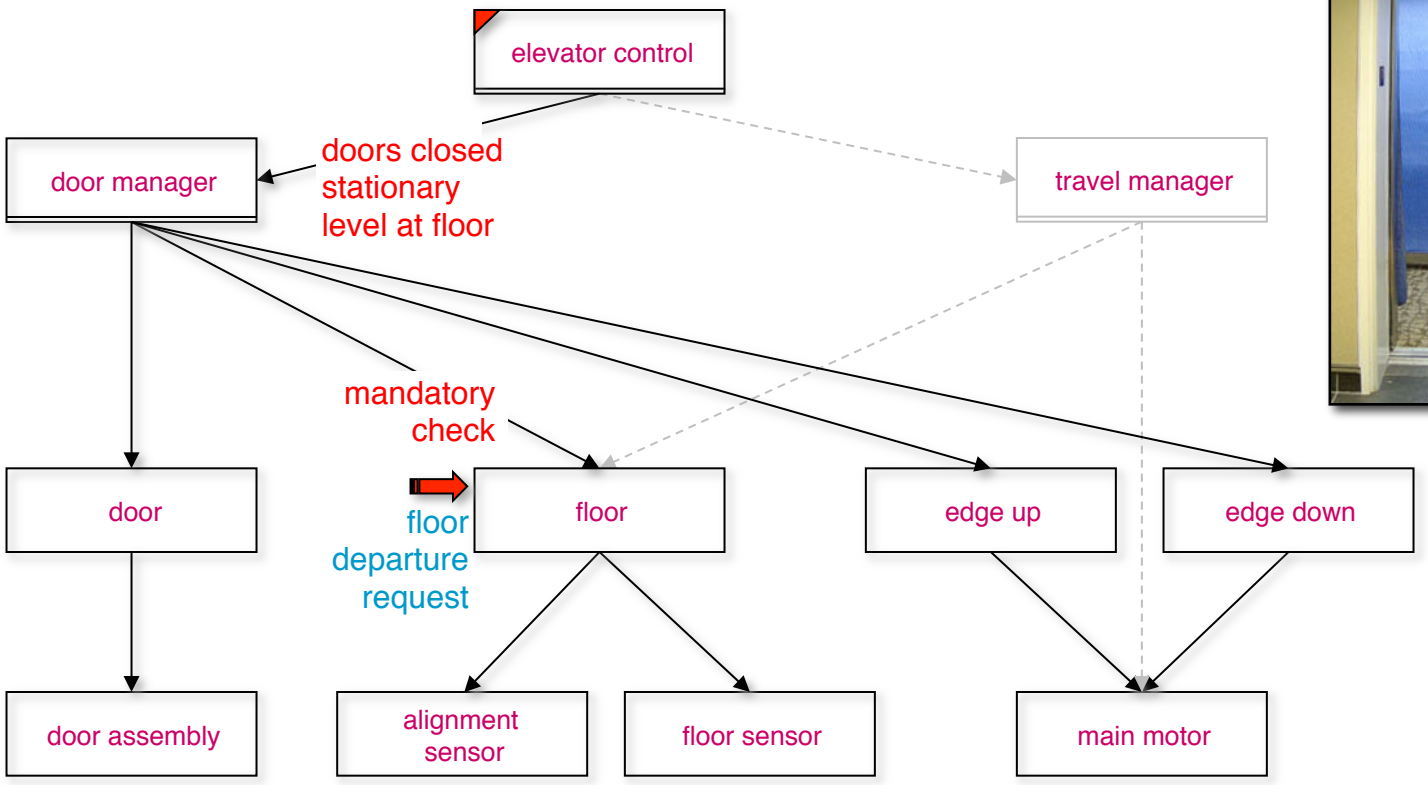
Continuous alignment while at a floor



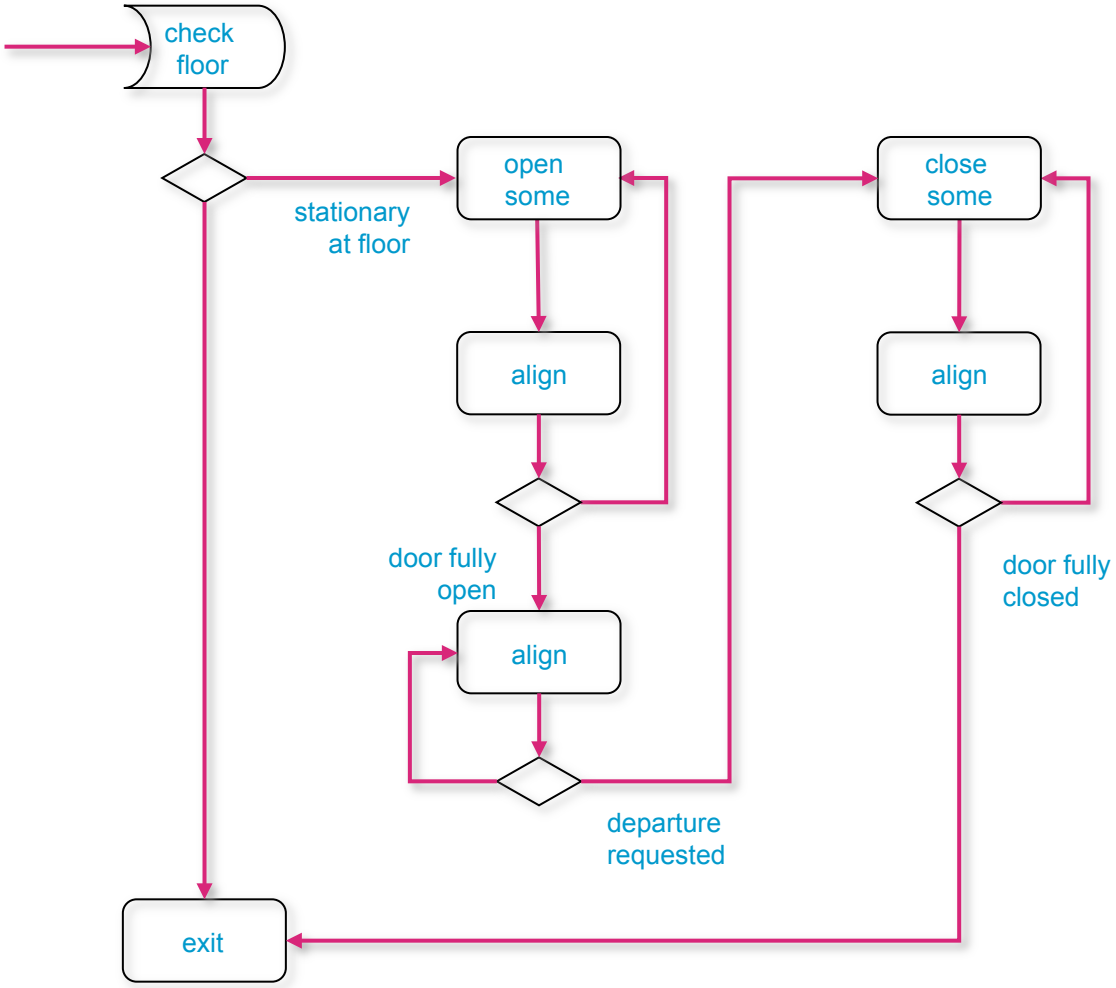


Safe Design

Doors are closed while moving or not at a floor



Behavior Analysis



Formal Analysis

Pre-condition (rechecked)

- doors closed
- level at floor

Post-condition (guarantees)

- doors closed
- level at floor

Invariant (enforced)

- level at floor

Formal Assumptions

Sub-problems

- dependable floor check
  - correct report
- dependable alignment
  - finite bounded motion
  - correct direction
- dependable opening
  - eventual completion
  - accurate report
- dependable closure
  - eventual completion
  - accurate report

Other assumptions

- motor safety guarantees
  - lack of motion implies the motor is off
  - motor needs a control signal to start
- no concurrency
  - no external motor activation
  - no unexpected state changes
- floor departure request limited to single boolean setting

3.3 Architectural Styles

- Architectural styles are abstractions for classes of organizational patterns encountered in software engineering practice
  - structural patterns
    - composition
  - behavioral patterns
    - communication
    - coordination
- Idioms relate to high level organizational patterns
- Patterns relate to design solutions for frequently encountered problems
- Reference architectures relate to classical and influential solutions proven in a specific domain

More about Styles

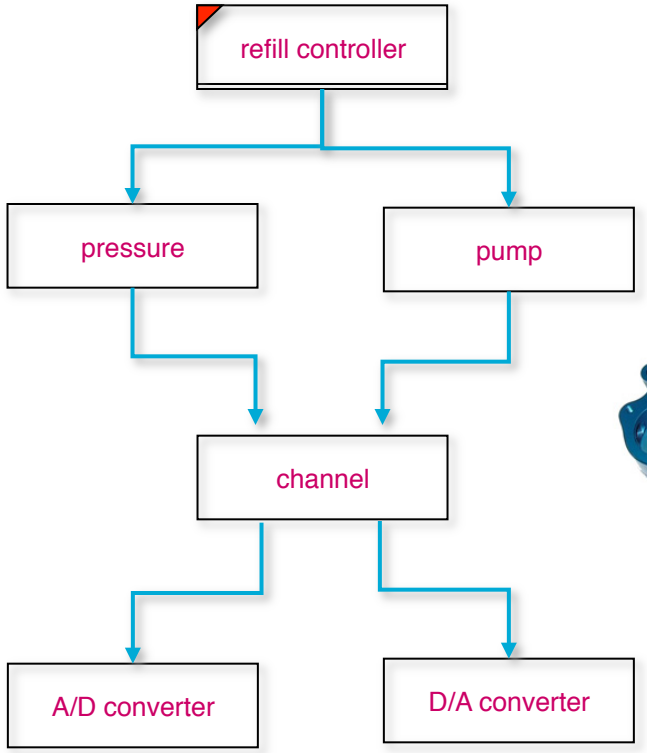
- Characterization
  - vocabulary
  - configuration rules
  - semantics
- Benefits
  - design reuse
  - predictability
  - corporate knowledge
- Taxonomic principle
  - the dominant strategy for achieving separation of concerns
    - composition
    - communication
    - coordination

A Taxonomy of Styles

- Composition
  - static
    - layered
    - controller
    - client/server
    - pipe and filter
    - dataflow
    - peer to peer
    - proxy
  - dynamic
    - object orientation
    - software bus
    - service provision
- Communication
  - explicit
    - message passing
    - event propagation
- Coordination
  - implicit
    - blackboard
    - shared tuple space
    - database
- Emerging styles
  - new software domains
    - mobile computing
    - sensor networks

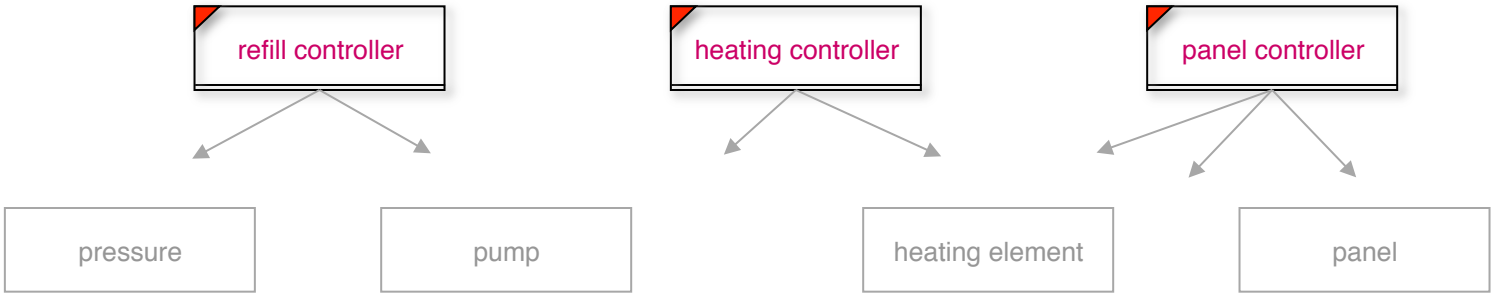
Layered

- Components at one layer share access to resources provided by the layer below
- It is critical to limit or structure sharing
- Hierarchical structuring facilitates
  - clean separation of constraints
  - limited impact of changes



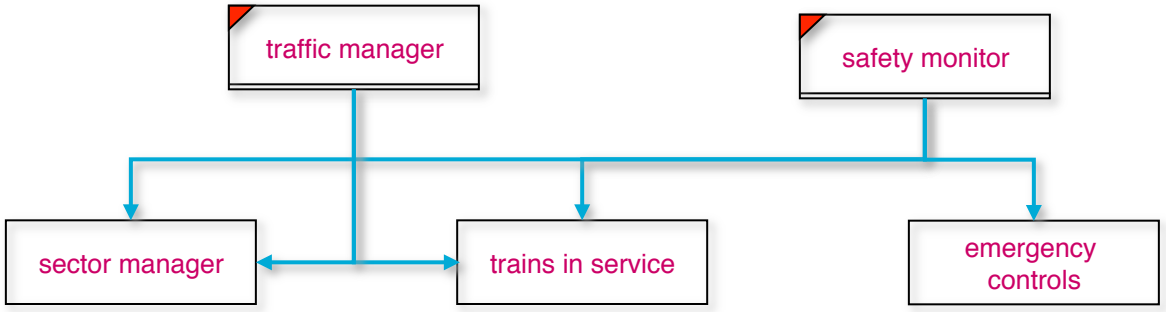
Controller

- System behavior
  - abstracted as a set of finite state machines
  - distributed among a small number of decoupled controllers
  - limited information sharing
  - synchronization limited to situations that demand consistency among decisions
- Complexity control
  - logical and structural separation into multiple modes of operation
    - simple and uniform transitions rules
  - object encapsulation of non-finite state aspects
  - partitioning of the control logic
  - state reduction



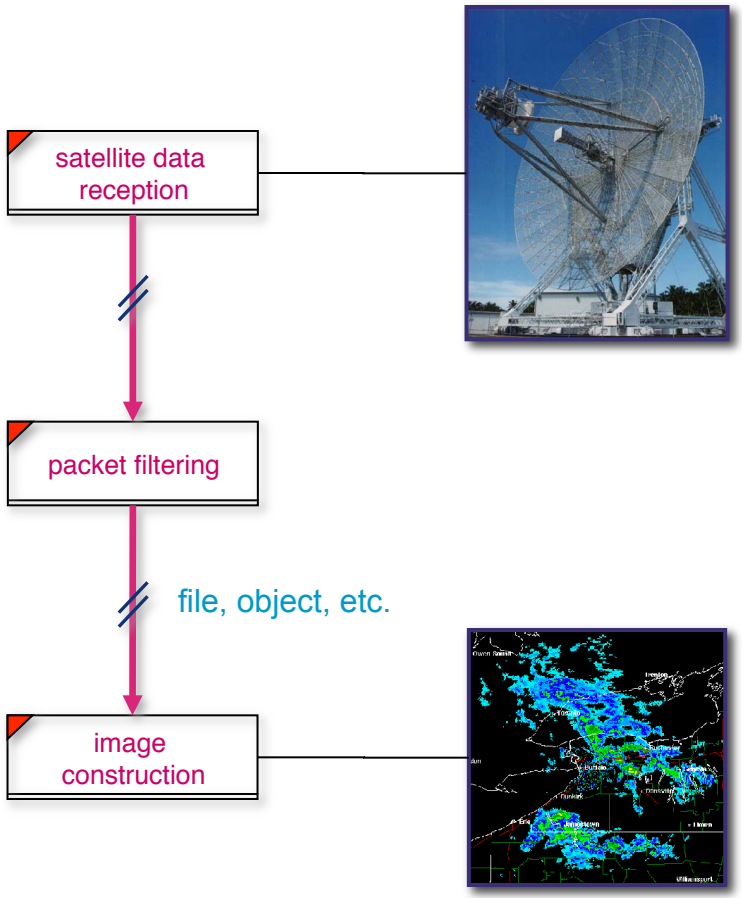
Client Server

- Natural extension of procedure call to distributed computing
- Components are segregated into clients and servers
  - Interactions are via remote procedure calls
  - Servers do not need to know client identities
  - Clients must know the identity of the servers



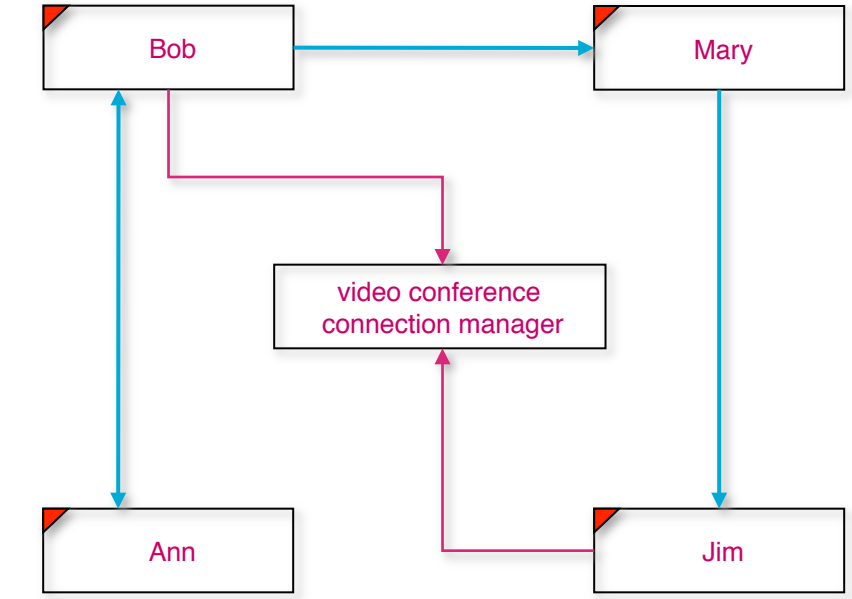
Pipe and Filter

- Independent components process data in an incremental fashion
- Connectors are mechanisms for supporting data streaming between pairs of components
- Standardized and compatible interfaces are the key to composition
- Signal processing, UNIX shells, and image processing are common instances of this style
- ***Dataflow***
  - a generalization to arbitrary directed acyclic graphs
  - image processing, workflow, etc.



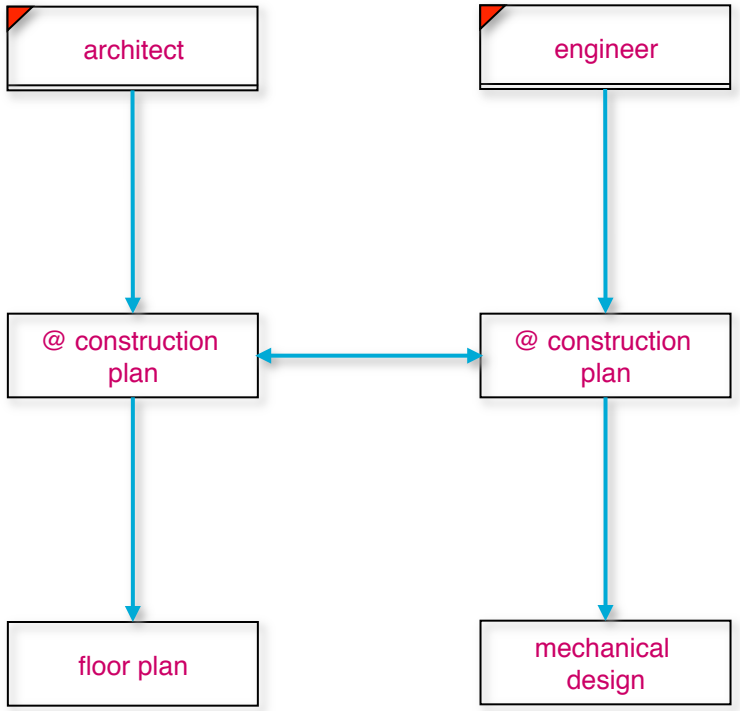
Peer to Peer

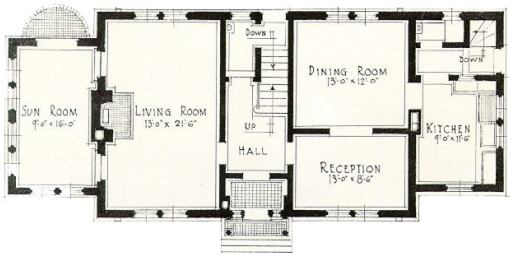
- Components are active objects that know (or learn about) each other
- Interactions are direct
- Server support is often required for discovery and some coordination
- Sample applications
  - music sharing
  - ad hoc network interactions



Proxy

- Interface components that decouple the access to a resource from the resource itself
  - resource updates do not affect the clients
  - different clients may have different views
  - communication protocols are hidden inside the proxy
  - proxy may be remote
  - proxy may be mobile code



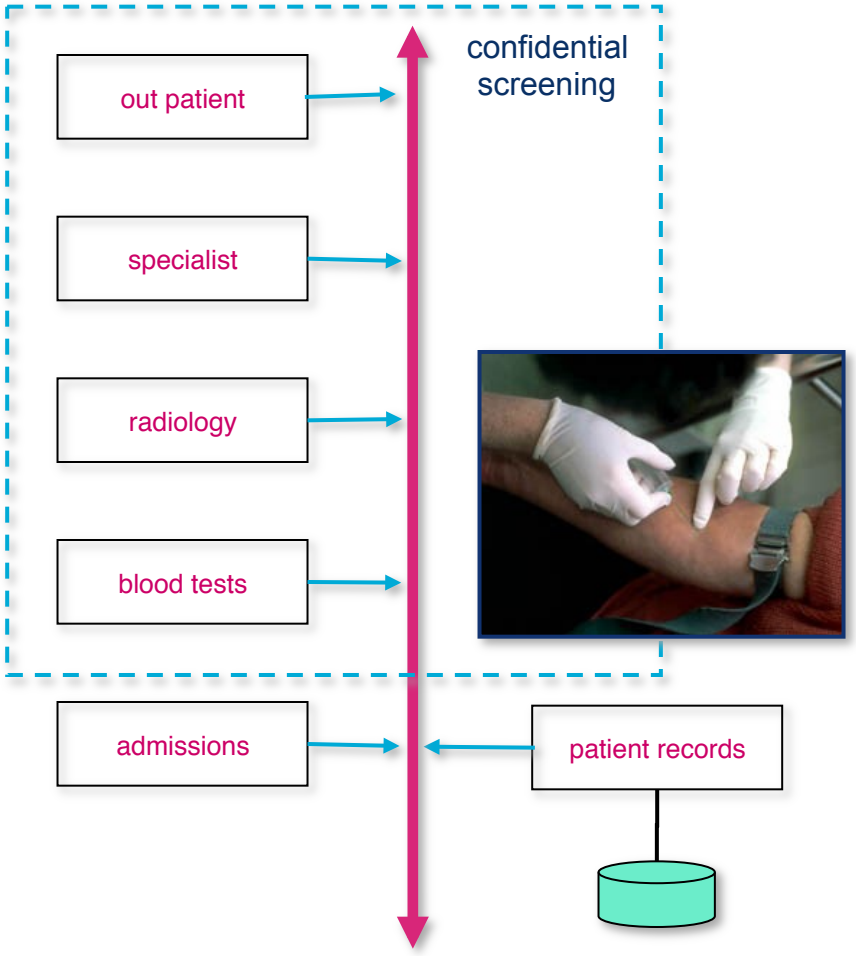


Object Orientation

- Components are objects and connectors are method invocations
- All architectural styles can assume an object oriented flavor
- The stereotypical object oriented style entails
  - web of cooperating objects
  - reference dissemination is often excessive

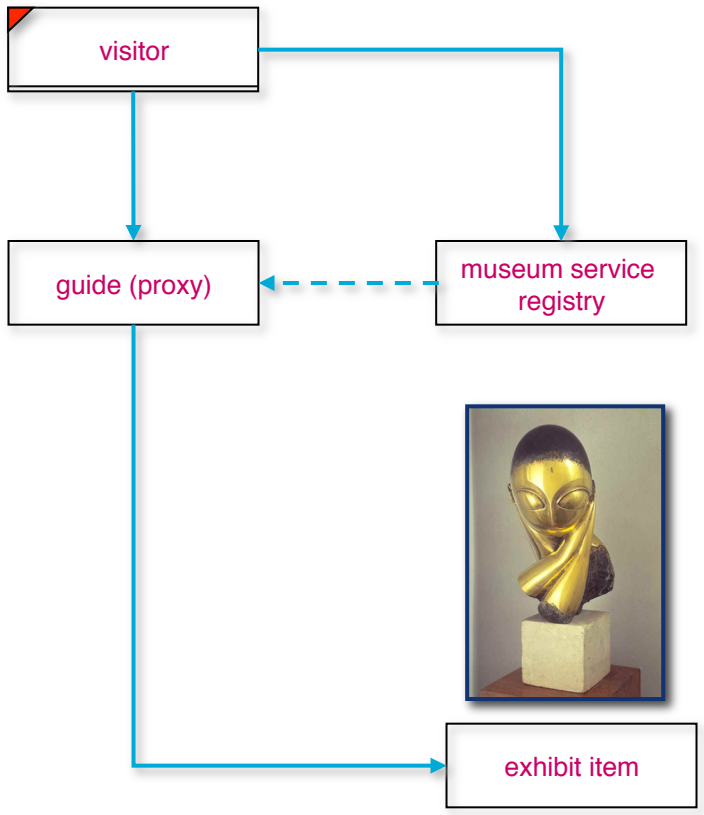
Software Bus

- Mechanism for decoupled access to
  - distributed data
  - services
- Static and dynamic object communities are supported
- Specialized services facilitate object registration, discovery, creation, and invocation
- CORBA is one standard that facilitates software bus construction



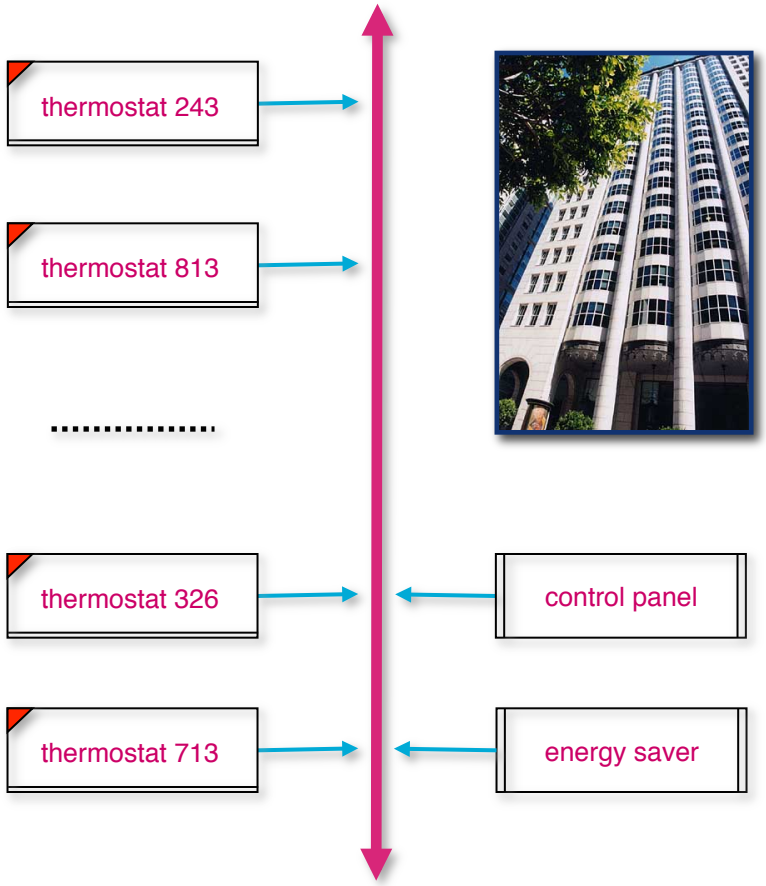
Service Provision

- Service providers register with service registries
  - specialized ontologies facilitate understanding
- Clients
  - services are discovered by querying service registries
  - interactions with services are standardized
    - proxy
    - service access protocols (e.g., SOAP)



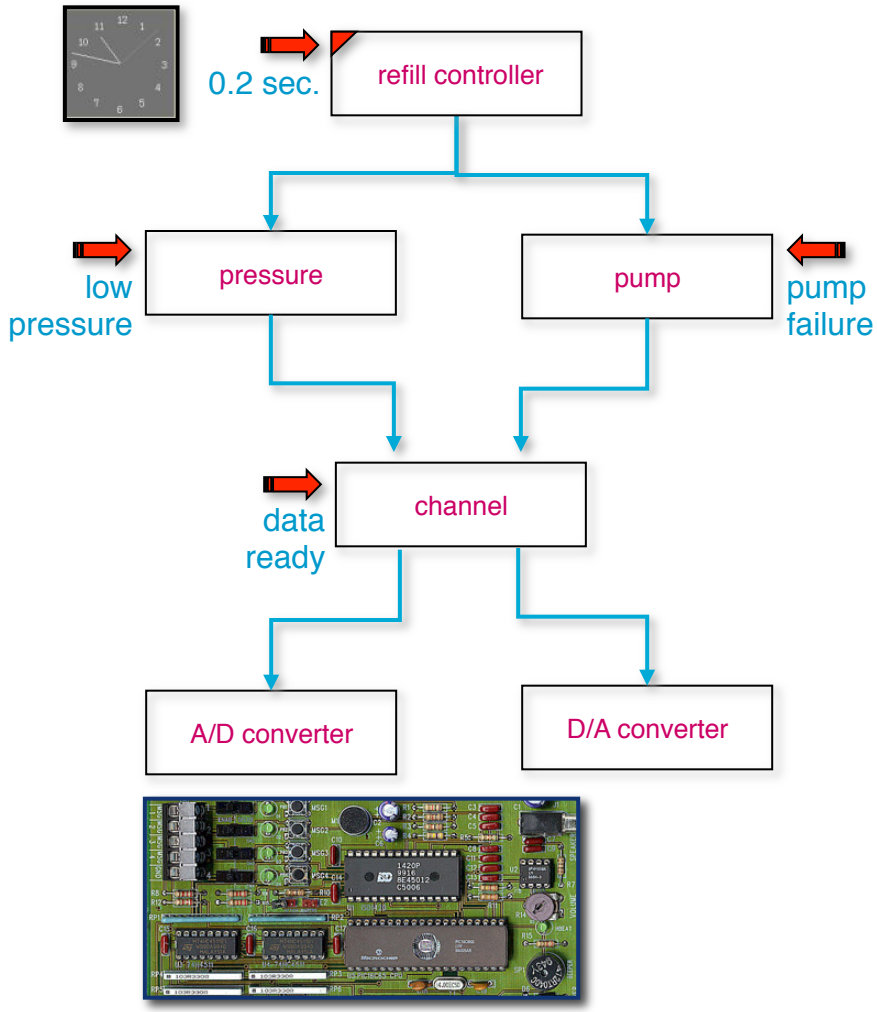
Message Passing

- Connectors are message passing capabilities
  - port (many to one)
  - channel (one to one)
  - mailbox (many to many)
- Component or connector identity is required to enable communication
- Bootstrapping requires some initial acquaintance
- Shared message interpretation and format
- High degree of decoupling and modularity is achievable
- Analysis relies on sequence and interaction diagrams



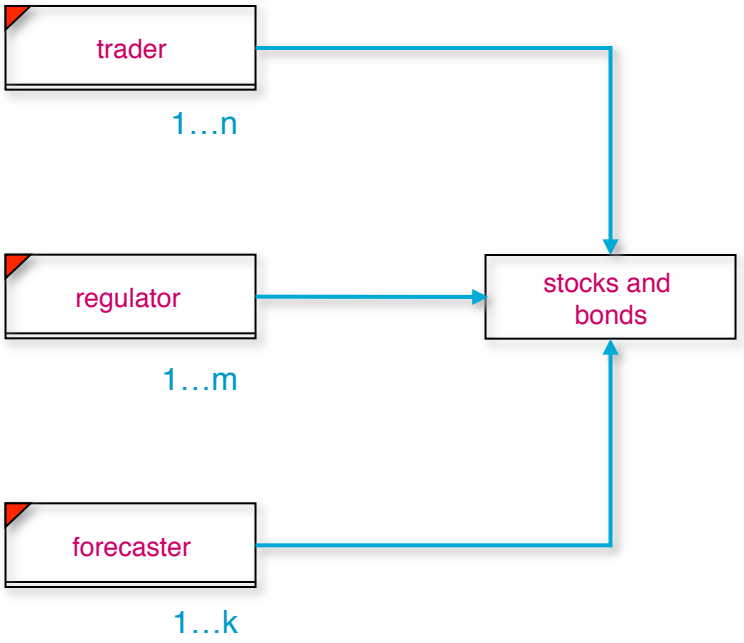
Event Propagation

- Publish/subscribe models
  - are meant to function in distributed settings
  - achieve a high level of decoupling
  - client subscribes to specific events
  - notifications are delivered automatically
- Explicit event propagation simplifies the design of reactive systems
  - low level events are mapped to abstract events
  - language support is available



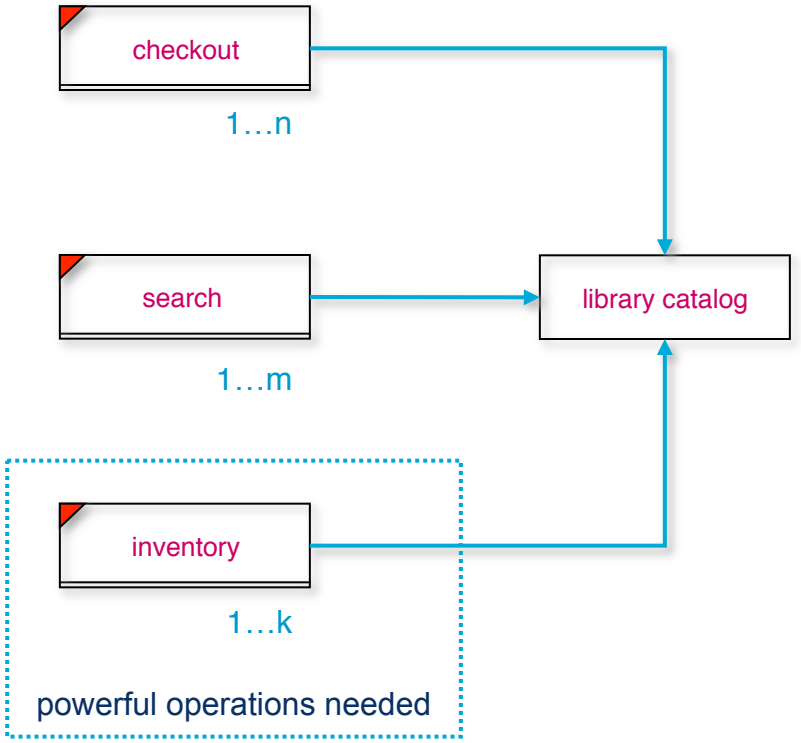
Blackboard

- A shared data repository
  - stores all key system data
  - controls system activities through its own state
- Other components
  - communicate with each other via the blackboard
  - respond to changes in the state of the blackboard
- Sample systems
  - planners
  - rule-based decision
  - compilers

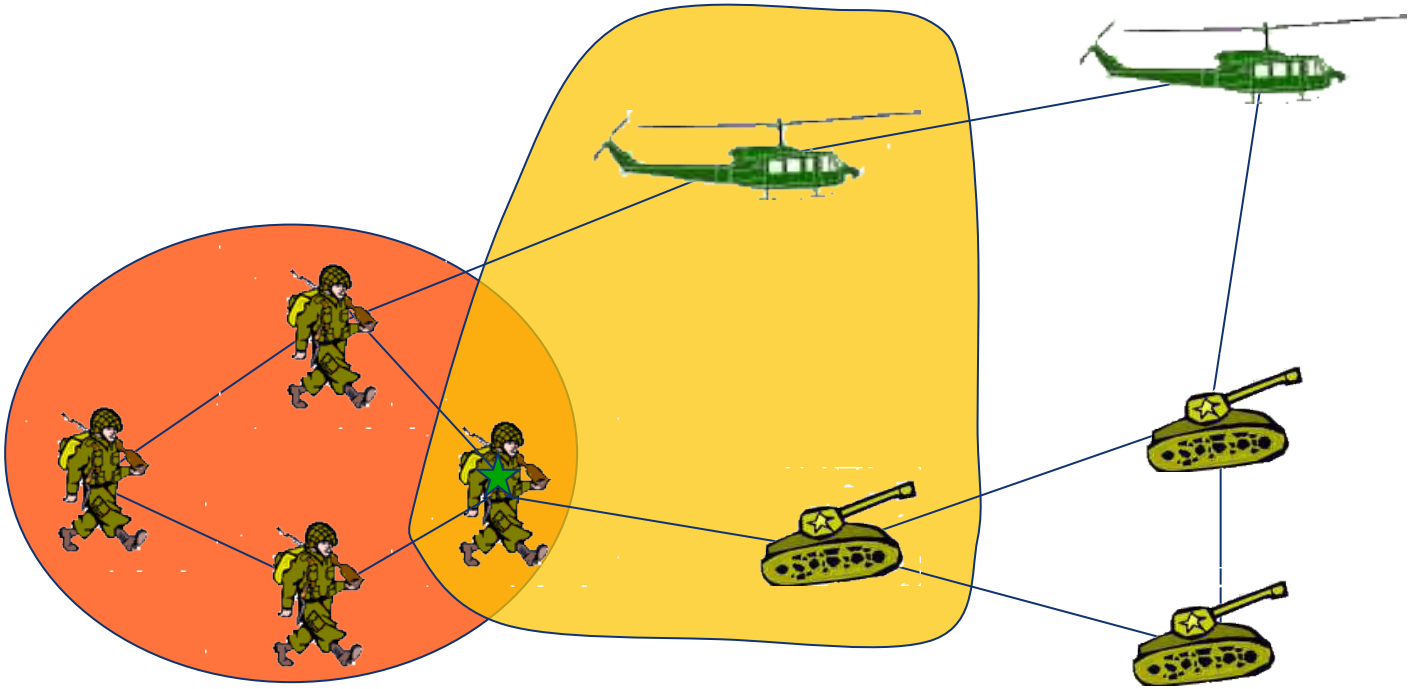


Shared Tuple Space

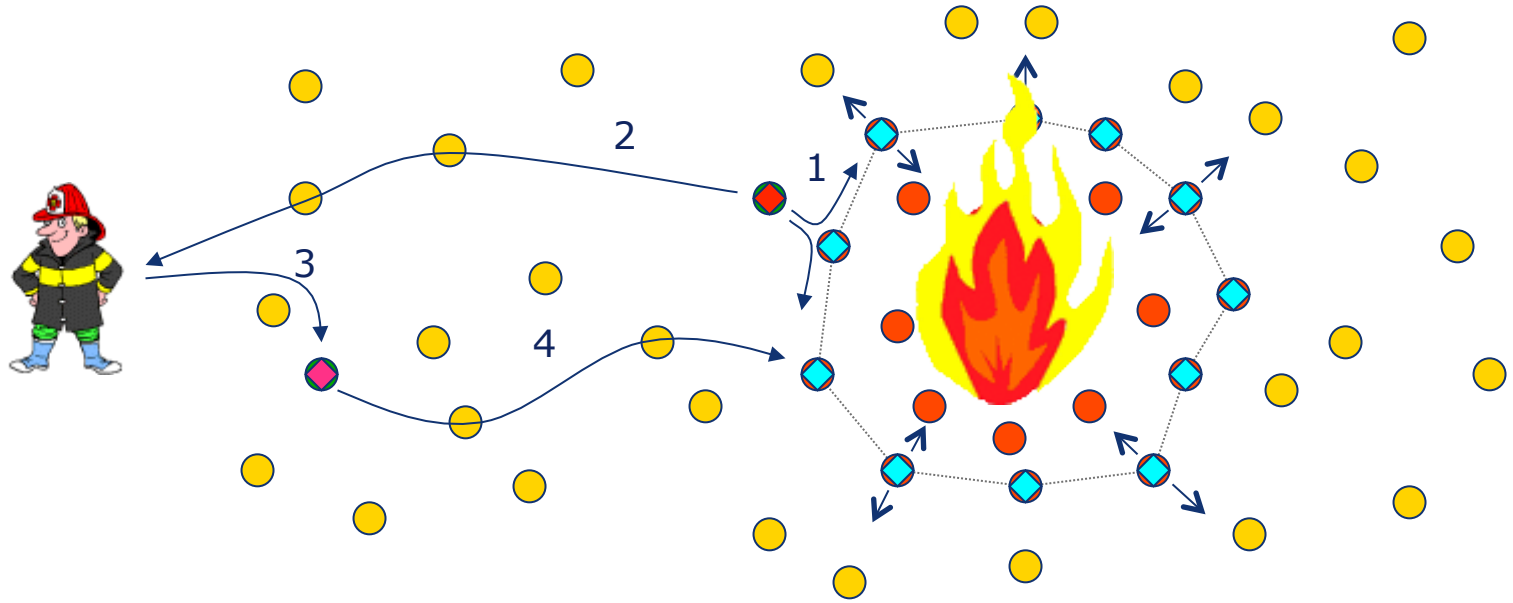
- The tuple space serves as a central data repository
  - out inserts a tuple
- Retrieval is contents-based (pattern matching)
  - read obtains a tuple copy
  - in removes a tuple
- High level of decoupling
- Minimal interface
- ***Databases***
  - large scale
  - complex queries
  - optimized processing
  - reuse



Mobile Computing



Sensor Networks



Observations

- The architectural style impacts the properties of the system up to a certain level
  - distinct architectures may share a common style
  - similar architectures may exhibit distinct properties
- Design rules may enforce the extent to which resulting architectures acquire specific properties
- Architectural styles may be associated with specific application domains
- Management and study of architectural styles
  - is mostly an art
  - has strong engineering implications

3.4 Design Patterns

- Role of architecture
  - relies on fundamental modeling assumptions
  - captures essential aspects of the system behavior
  - provides a basis for behavior analysis
- Designer goal
  - an architecture that ensures specific product qualities
- Mode of operation
  - reuse design patterns as a starting point
  - exploit know-how embodied in design patterns

Fundamental Principles

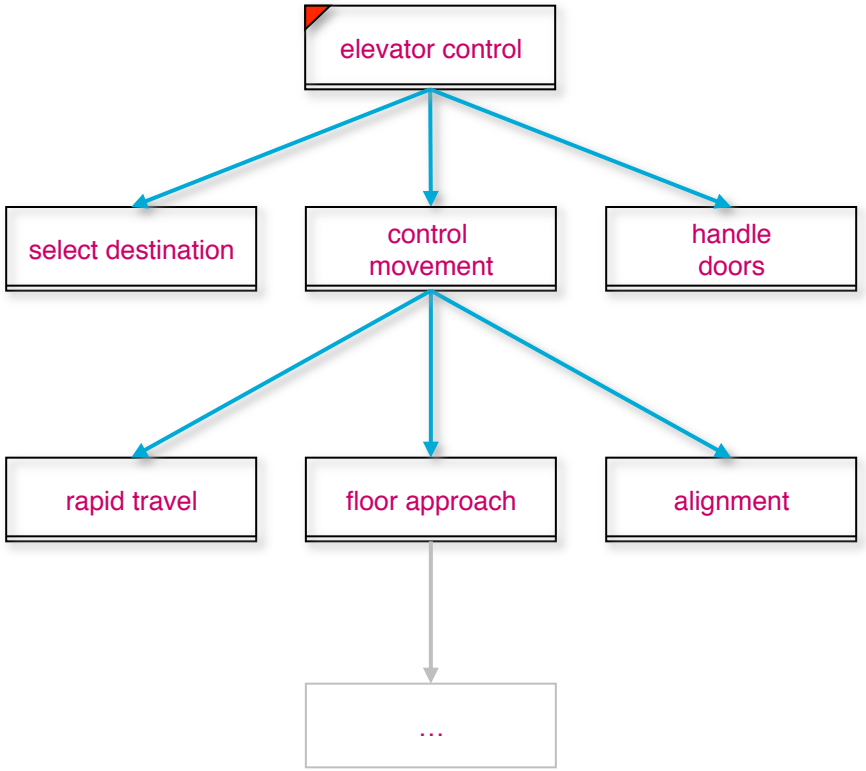
- Understand product goals and critical concerns
- Reuse design and analysis strategies
- Design for analyzability
  - with respect to critical concerns
- Practice design and analysis integration

Pattern Description Strategy

- Critical concern
  - systems are highly complex
  - multiple critical considerations may be relevant
  - patterns focus on single dimensions
- Solution sketch
  - initial designs are only a starting point
  - creativity and experience guide the refinement process
- Analytical method
  - reliance on specification techniques generally used in architectural design

3.4.1 Predictable Behavior

- Functional composition ensures predictability
  - the behavior of one function is always the same for the same input (system state)
  - functions are independent of each other
  - there are no side effects
  - there is no concurrency
  - computations are deterministic



Analysis

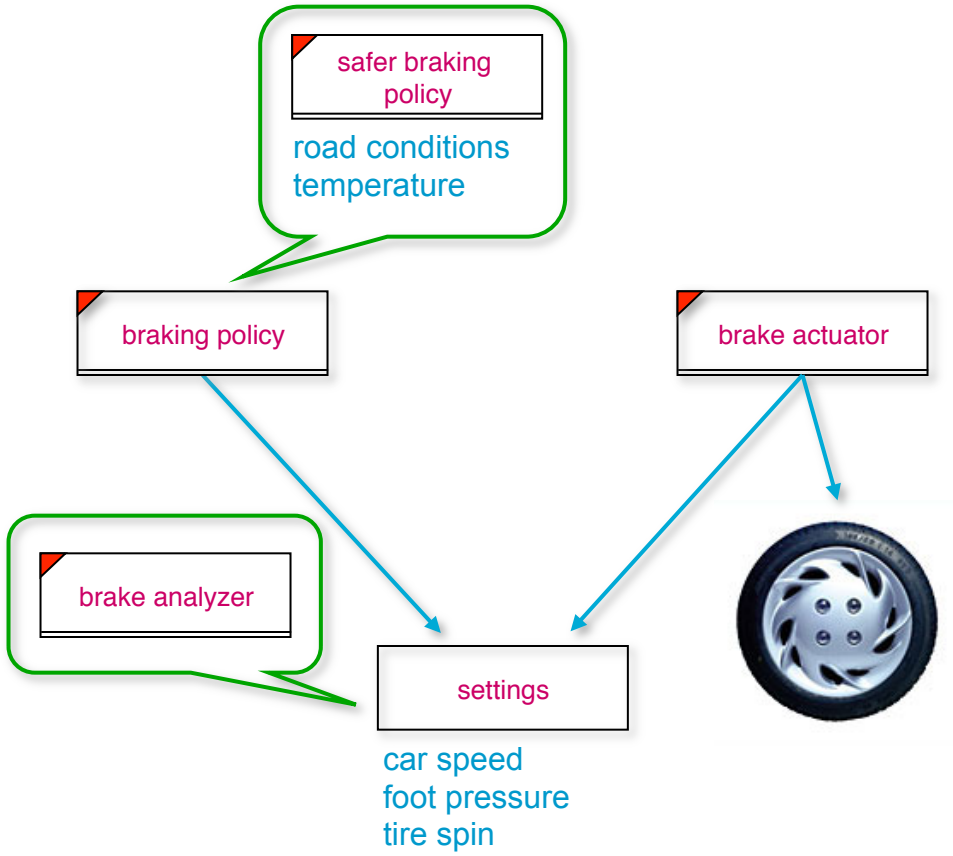
- Composition mechanics
  - localized and readily identifiable for analysis
  - components of interest
    - *elevator control*
    - *control movement*
  - strict sequential behavior: (a (x y z) b)*
    - deterministic
    - repeatable
- No concurrency anywhere in the system
- Interface consistency
  - parameter syntax and interpretation
    - e.g., null vs. empty message
  - implicit parameters (e.g., shared objects)

Analysis

- The specification method impacts analysis
- Use of pre and post assertions is critical
  - defines behavior independently of implementation
  - provides the basis for independent testing
  - promotes interface consistency
  - makes static analysis possible
  - makes run-time checking feasible
  - enables run-time self-healing
- Example: alignment
  - pre: stopped at some floor; ± 20 cm of horizontal; confirmed
  - post: stopped at some floor; ± 1 cm of horizontal; confirmed

3.4.2 Expandability

- Decoupled computing facilitates reconfiguration
  - adding/removing of components and data
- Blackboard architecture
  - access by name
- Tuple space coordination
  - access by contents
- Decoupling by design
  - encoding a traditional design is not a good solution



Analysis

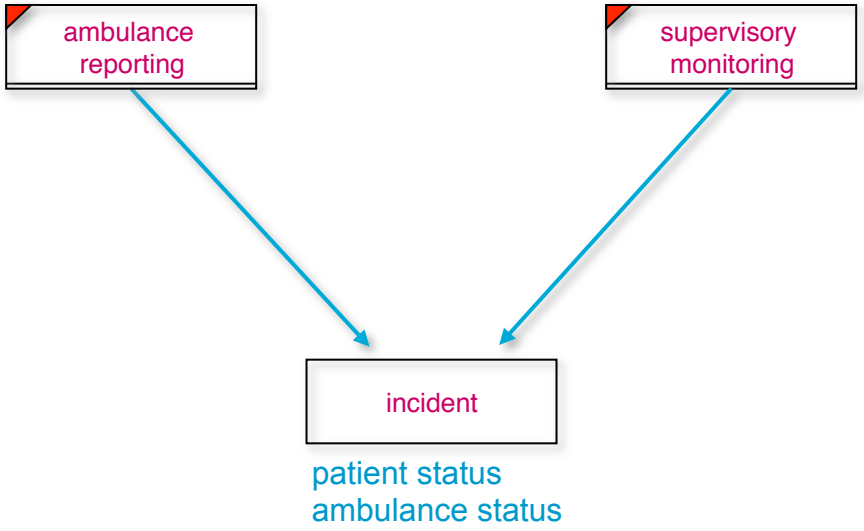
- Invariant property
  - characterization of the system state
  - holds at initialization
  - holds after every operation
    - precise definition of an “operation” is required
    - procedure invocation, block, statement, etc.
- Static analysis offers maximal benefits
  - isolated evaluation of each individual operation
- Run-time enforcement and verification is feasible
  - testing during development
  - protection during field operation

Analysis

- Structural invariants for dynamic systems
  - one and only one brake actuator
- Data representation
  - all measurements are in the metric system
  - <“speed”, 12, m/s, “at”, 1236, ms>
- Data consistency
  - brake application and cruise control cannot coexist
- Operating assumptions
  - freshness and accuracy

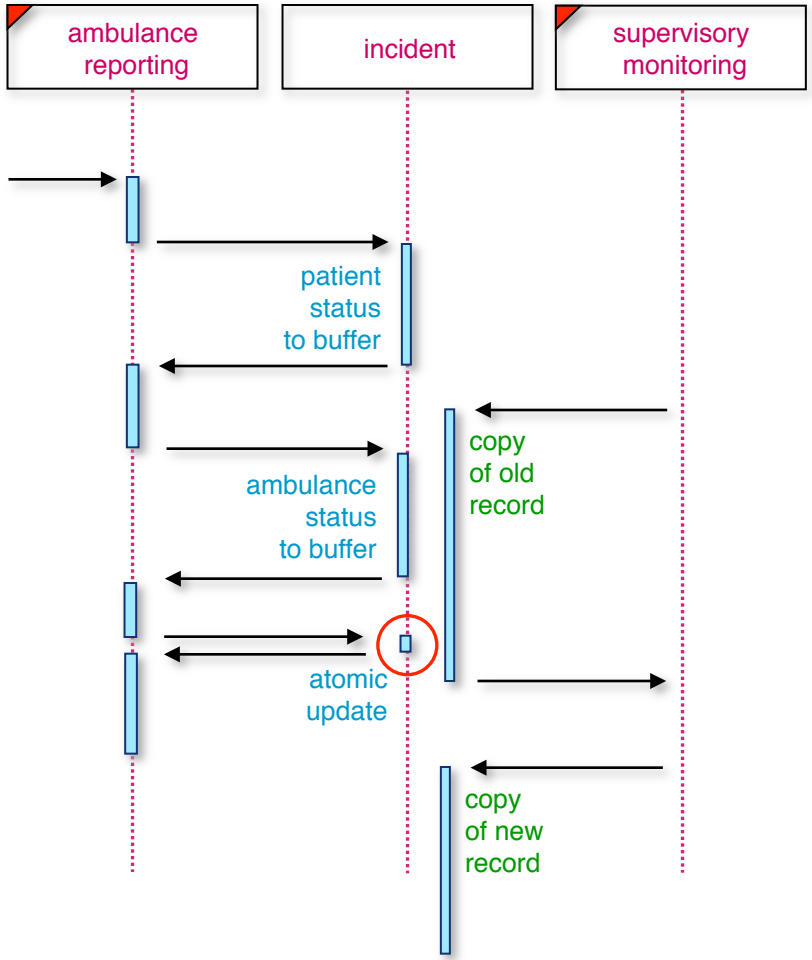
3.4.3 Non-Interference

- Concurrent processes can interfere with each other
- Data inconsistencies lead to incorrect conclusions
  - patient loaded
  - ambulance empty
- Wide range of patterns specialized to varying situations



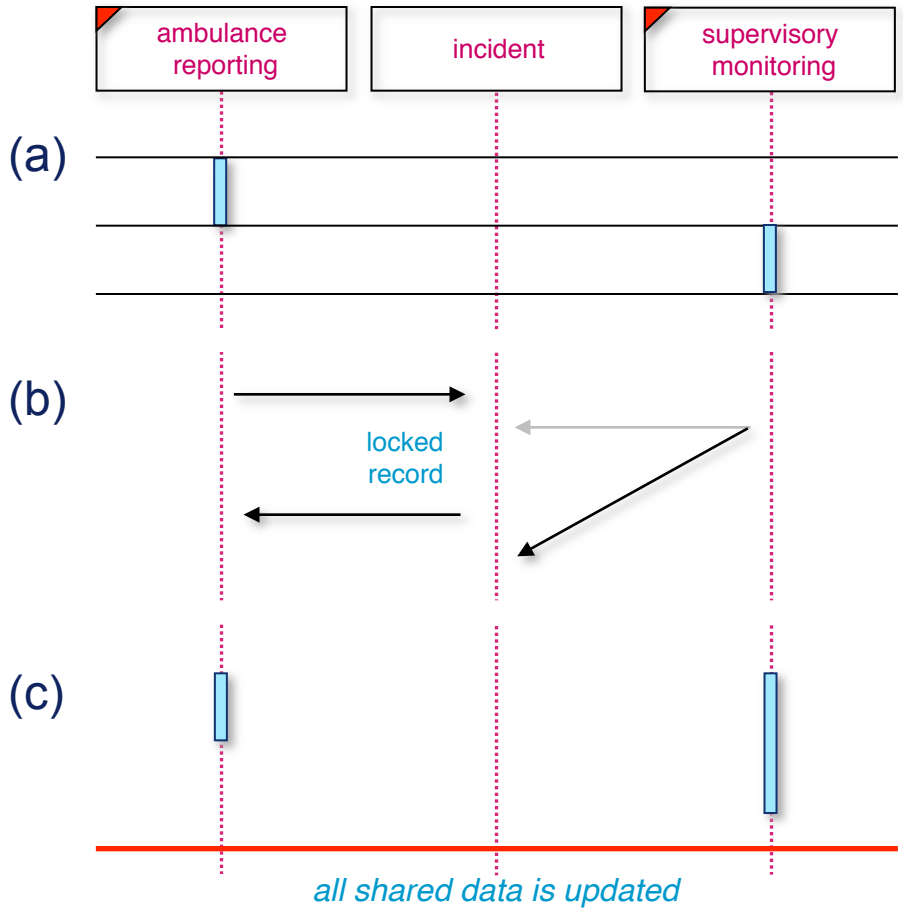
Analysis

- Assessing non-interference
  - syntactic methods: no shared data
  - semantic methods: no concurrent access to shared data (by analysis)
- Design: double buffer
  - atomic copy on read
  - atomic swap on write
- Analysis:
  - no interference possible



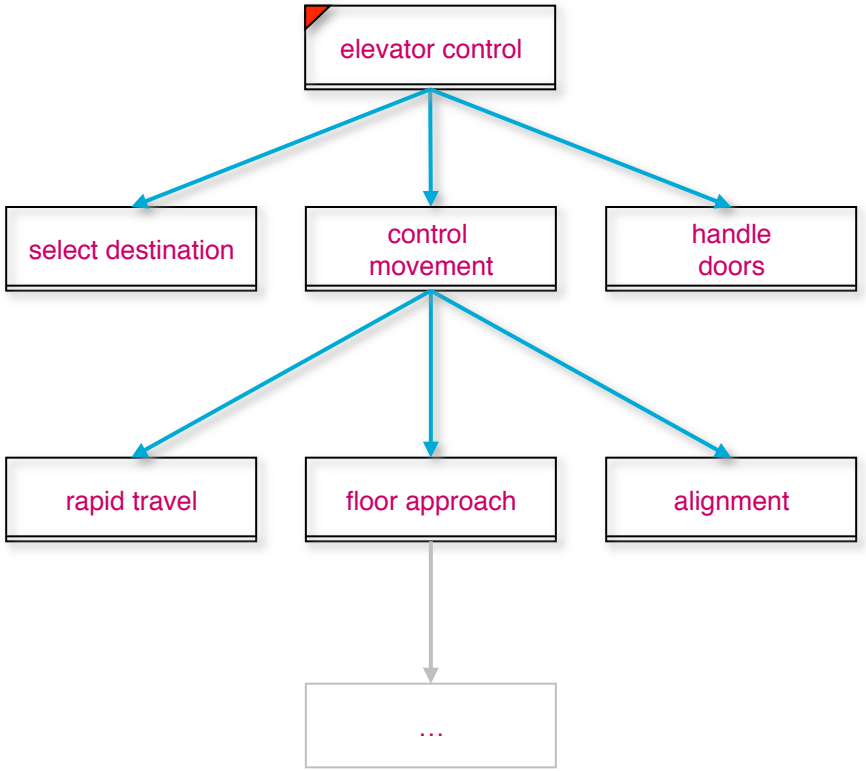
Analysis

- Controlling interference
  - explicit scheduling
  - record locking
  - barrier synchronization



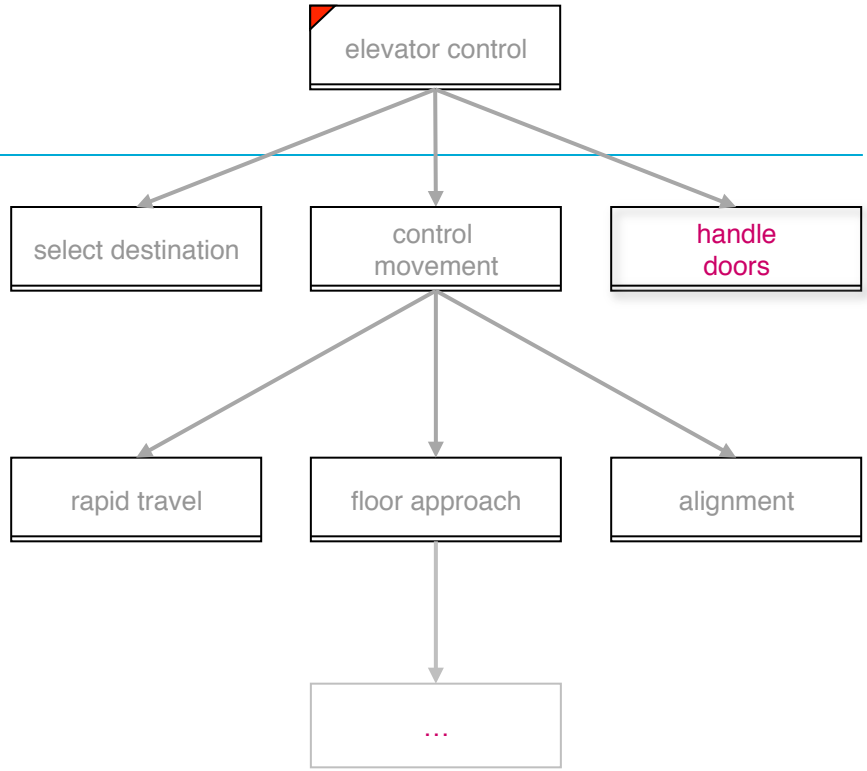
3.4.4 Strong Guarantees

- Precise specification of relevant requirements
- Designed partitioned for
  - analyzability
  - separation of concerns
- Design rules limit scope and cost of analysis
- Partition enhance ability to carry complete analysis



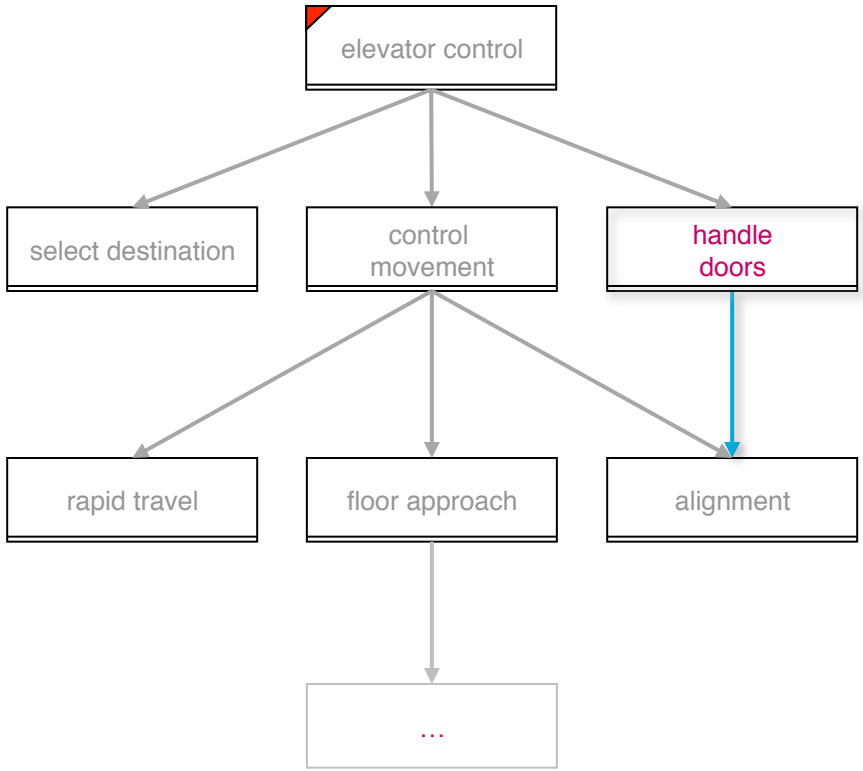
Analysis

- Requirement
  - when doors are open elevator must be stopped at some floor
- Architectural implications
  - handle doors:
    pre: stopped at floor
    post: doors closed
  - invariant:
    inside handle doors elevator is stopped at floor
    [no access to elevator travel motor]
    outside handle doors doors are closed
    [no access to door activation motor]



Analysis

- Requirements
  - when doors are open
    - elevator must be stopped at some floor
    - alignment must be within tolerance
- New design is needed to ensure analyzability
  - elevator motor is in use while doors are open
- New alignment guarantee
  - invariant: inside alignment if stopped at a floor the elevator remains at that floor

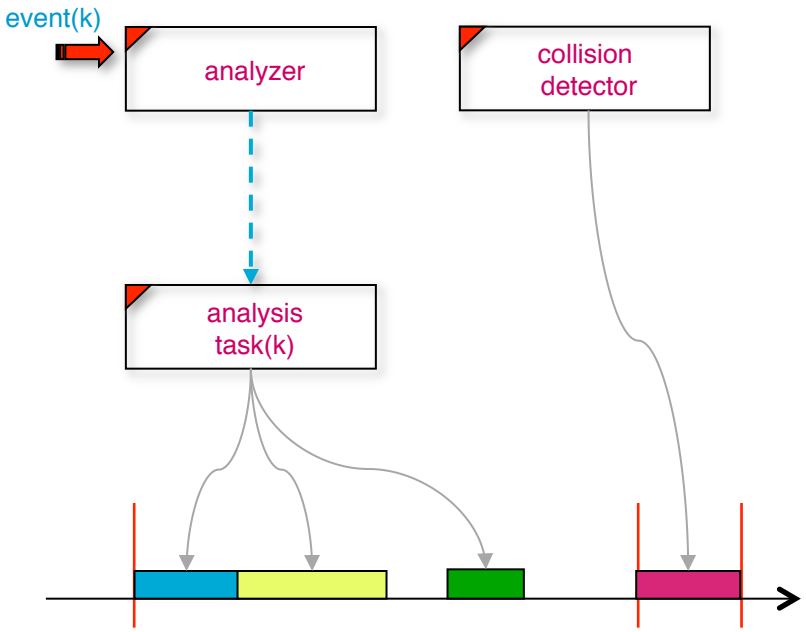


3.4.5 Real Time Guarantees

- Strong real time guarantees demand strict design rules
- Predictable scheduling
  - deterministic order
  - analyzable
- Predictable component behavior
  - restricted constructs
    - bounded loops
    - no recursion
  - computable worst case behavior

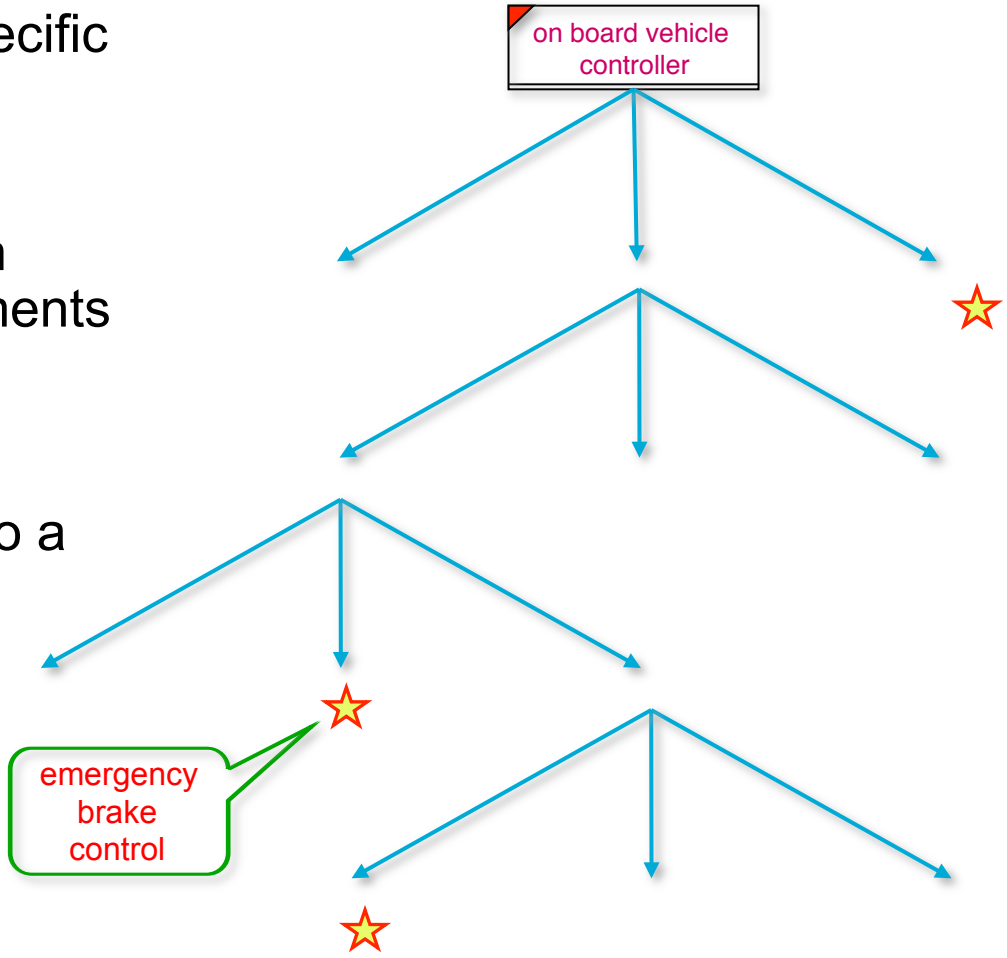
Analysis

- Static analyzer
  - T sec. allocation
  - worst case bound t_k
  - $\sum t_k \leq T$
- Dynamic analyzer
  - priority based preemption
  - admission control
- Collision detection
  - most critical task
  - long duration
  - must run in the background
  - reserved fraction of time



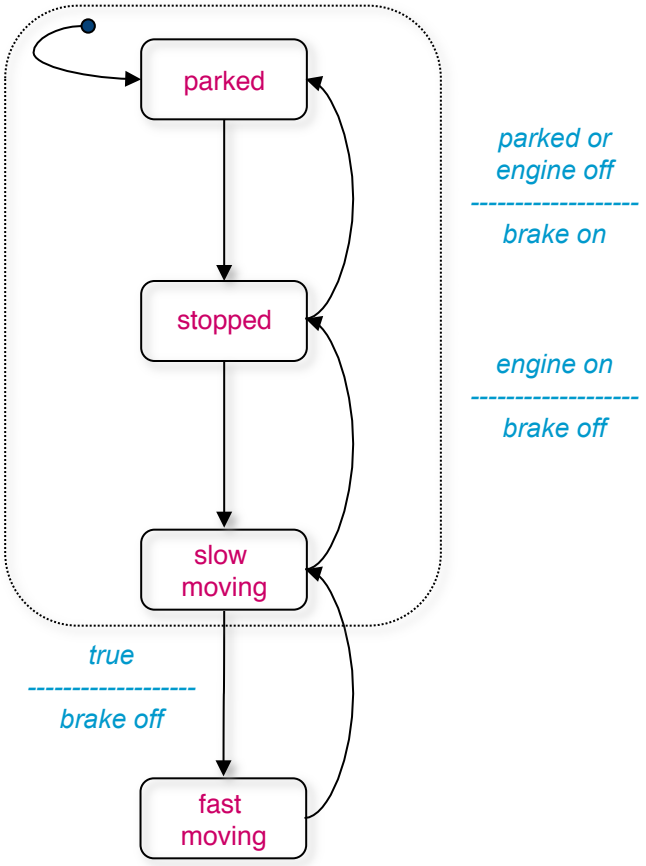
3.4.6 Safety Guarantees

- Safety is a requirement driven concern under design specific faults
- Precise specification
- Easy traceability between architecture and requirements
  - isolation
  - visibility
- Analyzable with respect to a specific set of faults
  - failure analysis
  - fault model



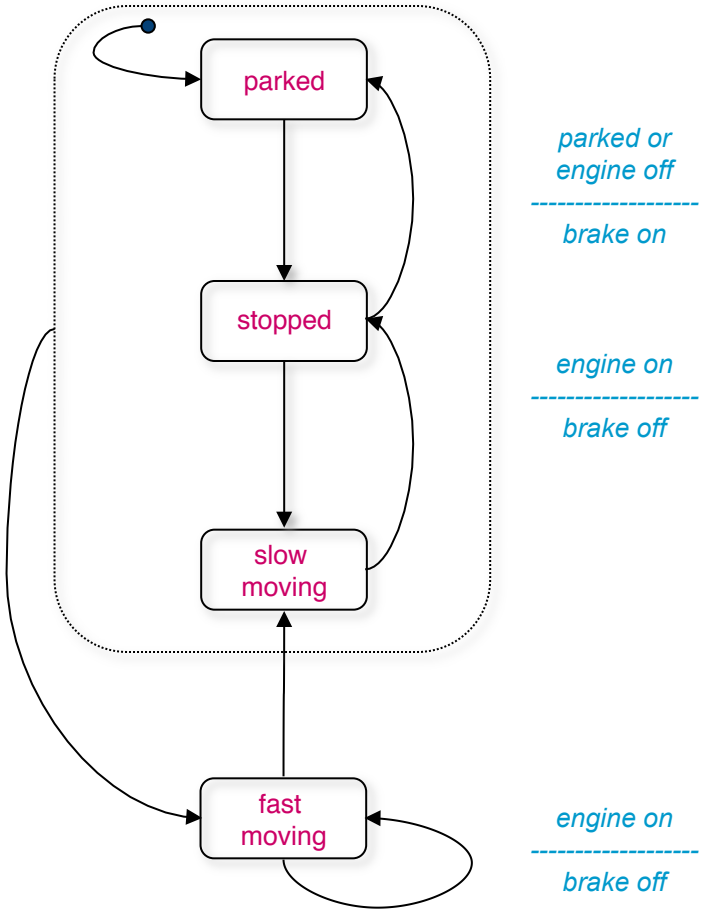
Analysis

- Drive-by-wire emergency brake safety specifications
  - brake can be engaged only when the car is parked
    - stalled on a ramp?
  - brake can be engaged when the car is parked or the engine is off
    - even at high speed?
  - brake can be engaged when the car is parked or the engine is off at a low speed



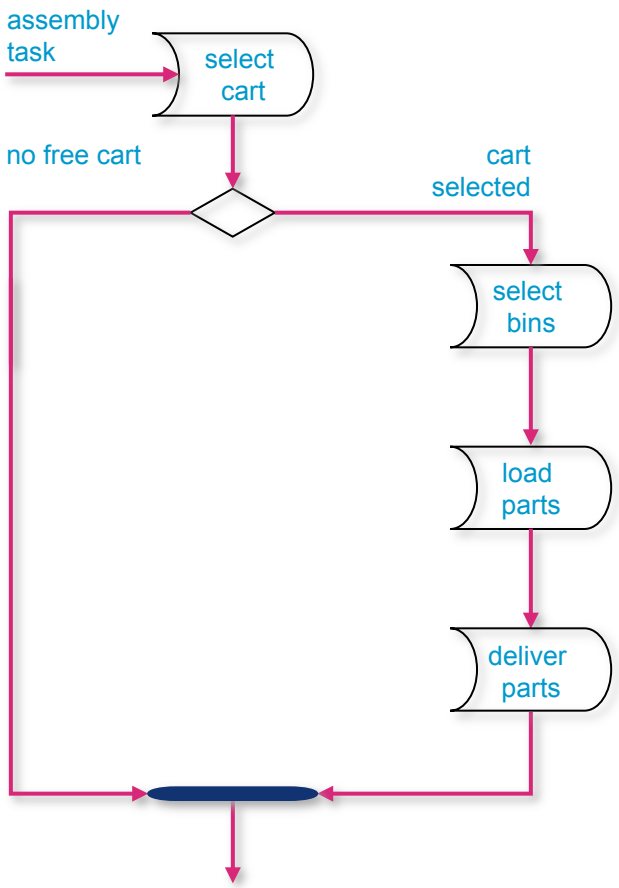
Analysis

- Drive-by-wire emergency brake hazard analysis
  - car may accelerate fast if hit from behind
  - missing transitions must be added and implications on the specification must be clarified
  - visibility make both analysis and modification feasible



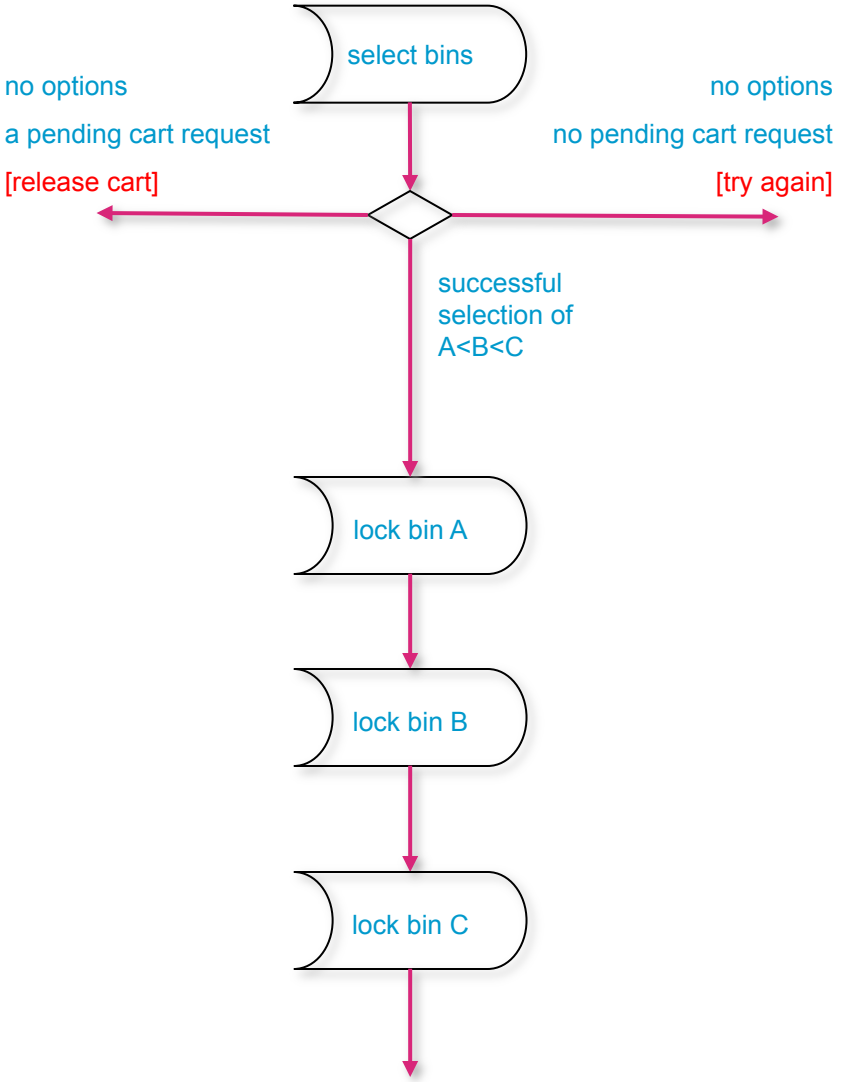
3.4.7 Anomaly Prevention

- Certain behavior anomalies are well understood and offer standard remedies
- Encapsulating relevant behaviors as part of controllers simplifies analysis
- Deadlock prevention
  - impose ordering on processes of resources
- Livelock avoidance
  - introduce asymmetry in the decision process



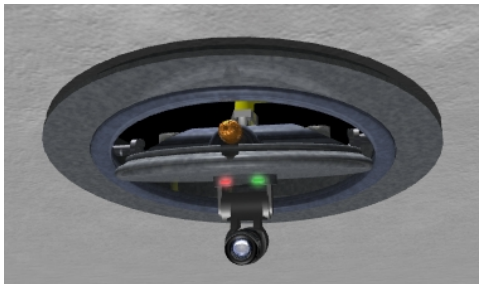
Analysis

- Livelock may happen when a cart unable to find the required parts is reselected repeatedly
  - reselection must be inhibited if other requests are pending
- Deadlock may occur when groups of parts are needed together, e.g., (A, B, C) or (X, Y)
  - bins must be reserved in the same order by all carts



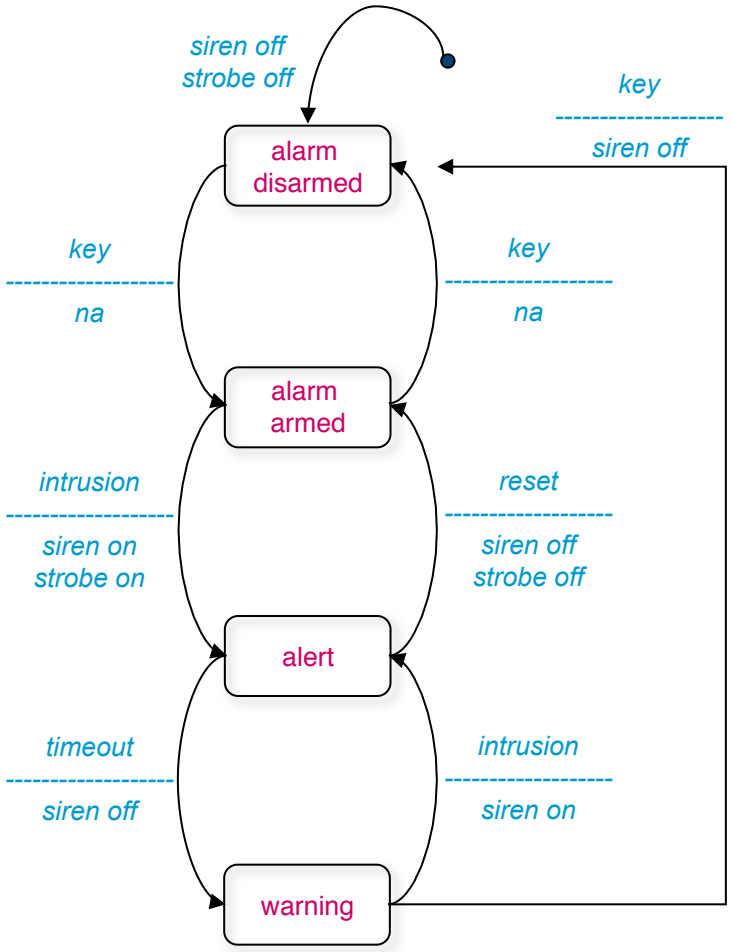
3.4.8 Behavioral Integrity

- Object integrity
  - prescribed patterns of usage are often required
  - must be enforced for proper functioning
  - objects may be unable to provide proper enforcement
  - exception handling may be undesirable
- Global consistency
  - objects are not used alone
  - local states may need to be consistent across multiple objects
- Behavior encapsulation and analysis facilitate enforcement



Analysis

- Home security alarm
  - audible alarm activation must be of limited duration due to local ordinances
  - visual alarm must be on any time the audible alarm is on due to disabled protection legislation
  - easy traceability to requirements is required by contractual obligation



Conclusions

- Software architecture design is central to
  - bridging requirements and implementation
  - ensuring and assessing the satisfiability of both functional and non-functional requirements
  - effective planning of the development effort
- Design patterns provide a reasonable starting point for formulating the architectural features
- Analysis must be an integral part of the design process
- Designing for analyzability is critical to ensuring credibility in the design, reducing analysis costs, and achieving desired level of quality