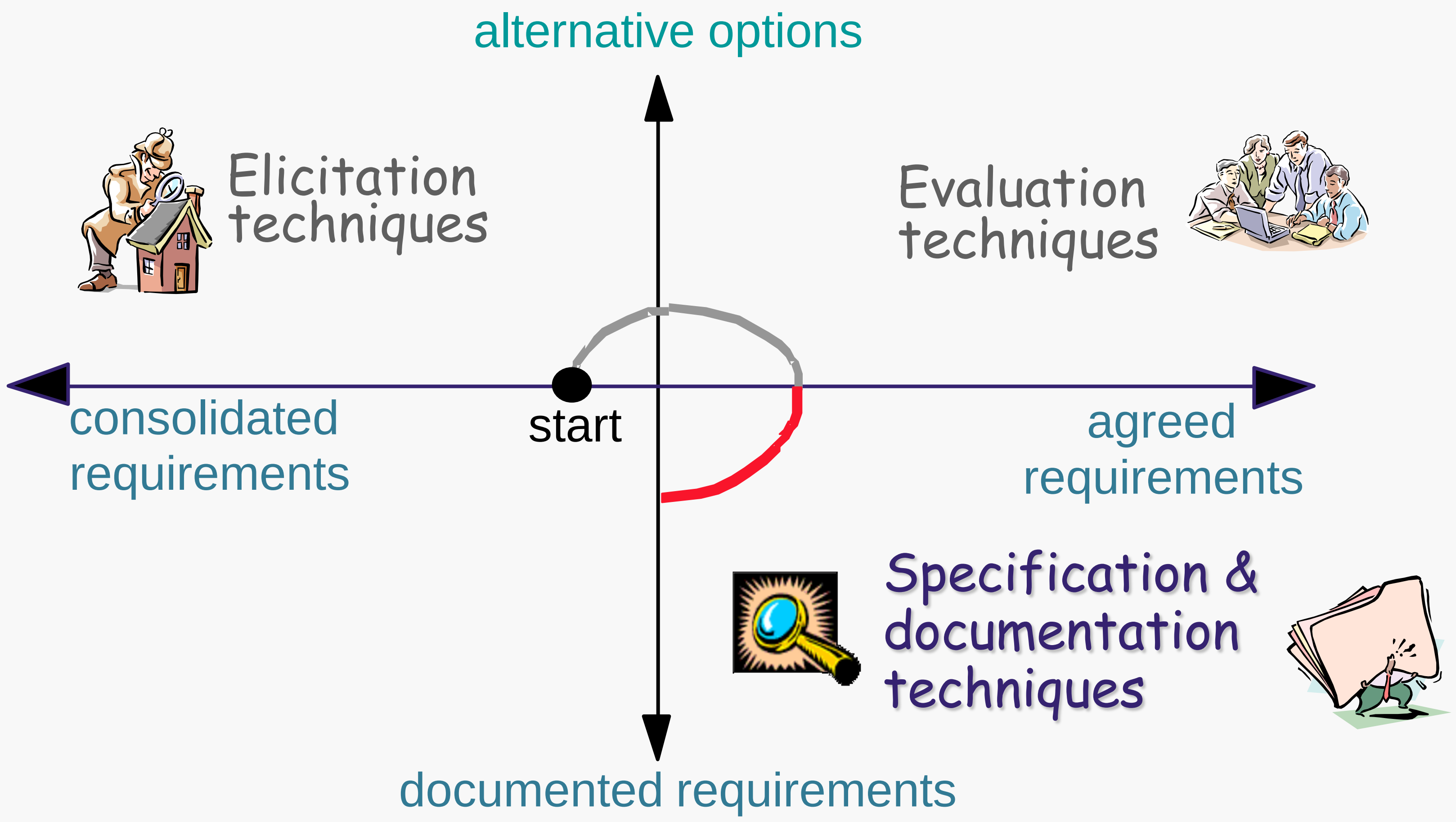
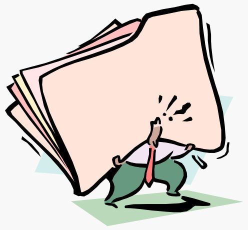
Requirements Engineering

Lecture 06:

Requirements Specification & Documentation


alternative options
Elicitation techniques
Evaluation techniques
consolidated requirements
start
agreed requirements
Specification & documentation techniques
documented requirements



Specification & documentation - Review

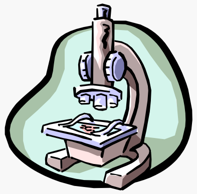
- Precise definition of all features of the agreed system
  - Objectives, concepts, relevant domain properties, system/software requirements, assumptions, responsibilities
  - Rationale for options taken, satisfaction arguments
  - Likely system evolutions & variants
- Organization of these in a coherent structure
- Documentation in a form understandable by all parties
  - Often in annex: costs, workplan, delivery schedules

Resulting product: **Requirements Document** (RD)

Requirements specification & documentation: outline

- Free documentation in unrestricted natural language
- Disciplined documentation in structured natural language
  - Local rules on writing statements
  - Global rules on organizing the Requirements Document
- Use of diagrammatic notations
  - System scope: context, problem, frame diagrams
  - Conceptual structures: entity-relationship diagrams
  - Activities and data: SADT diagrams
  - Information flows: dataflow diagrams
  - System operations: use case diagrams
  - Interaction scenarios: event trace diagrams
  - System behaviors: state machine diagrams
  - Stimuli and responses: R-net diagrams
  - Integrating multiple system views, multi-view spec in UML

Requirements specification & documentation: outline (2)



- Formal specification
  - Logic as a basis for formalizing statements
  - History-based specification
  - State-based specification
  - Event-based specification
  - Algebraic specification

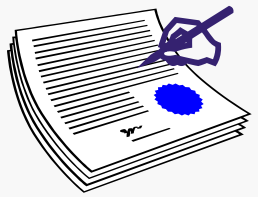
Free documentation in unrestricted natural language

- Unconstrained prose writing in natural language (NL) ...
  - ☺ Unlimited expressiveness, communicability, no training needed
  - ☹ Prone to many of the spec errors & flaws
- In particular, ambiguities are inherent to NL; can be harmful

  "Full braking shall be activated by any train that receives an outdated acceleration command or that enters a station block at speed higher than X m.p.h. and for which the preceding train is closer than Y yards."

- Frequent confusions among logical connectives in NL
  - e.g. case analysis:

```
         If Case1 then <Statement1>
      or if Case2 then <Statement2>          (amounts to true!)
vs.      If Case1 then <Statement1>
     and if Case2 then <Statement2>
```



Disciplined documentation in structured NL: local rules on writing statements

- Use stylistic rules for good NL spec, e.g.
  - Identify who will read this; write accordingly
  - Say what you are going to do before doing it
  - Motivate first, summarize after
  - Make sure every concept is defined before use
  - Keep asking yourself: "Is this comprehensible? Is this enough? Is this relevant?"
  - Never more than one req, assumption, or dom prop in a single sentence. Keep sentences short.
  - Use "shall" for mandatory, "should" for desirable prescriptions
  - Avoid unnecessary jargon & acronyms
  - Use suggestive examples to clarify abstract statements
  - Supply diagrams for complex relationships among items

Disciplined documentation in structured NL: local rules on writing statements (2)

- Use decision tables for complex combinations of conditions

input if-conditions

binary filling with truth values

Train receives outdated acceleration command	T	T	T	T	F	F	F	F
Train enters station block at speed ≥ *X* mph	T	T	F	F	T	T	F	F
Preceding train is closer than *Y* yards	T	F	T	F	T	F	T	F
Full braking activated	X		X		X			
Alarm generated to station computer	X	X	X	X				

output then-conditions

one case = AND-combination

- Systematic, simple, additional benefits ...
  - Completeness check: 2^N columns required for full table
  - Table reduction: drop impossible cases in view of dom props; merge 2 columns differing only by single "T", "F" => "-"
  - Test cases for free (cause-effect coverage)

Disciplined documentation in structured NL: local rules on writing statements (3)

- Use standardized **statement templates**

Identifier --suggestive; hierarchical if compound statement

Category --functional or quality req, assumption, domain property, definition, scenario example, ...

Specification --statement formulation according to stylistic rules

Fit criterion --for measurability (see next slide)

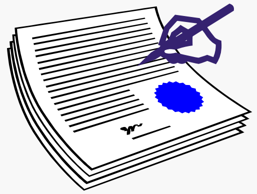
Source --for traceability to elicitation sources

Rationale --for better understanding & traceability

Interaction --contribution to, conflict with other statements

Priority level --for comparison & prioritization

Stability, Commonality levels --for change management



Fit criteria make statements measurable

- Complement statements by quantifying the extent to which they must be satisfied [Robertson, 1999]
- Especially important for measurability of NFRs

Spec: The scheduled meeting dates shall be convenient to participants

Fit criterion: Scheduled dates should fit the diary constraints of at least 90% of invited participants in at least 80% of cases

Spec: Info displays inside trains shall be informative & understandable

Fit criterion: A survey after 3 months of use should reveal that at least 75% of travelers found in-train info displays helpful for finding their connection

Disciplined documentation in structured NL: global rules on organizing the RD

- Grouping rules: Put in same section all items related to common factor ...
  - system objective
  - system component
  - task
  - conceptual object
  - software feature
  - ...
- Global templates for standardizing the RD structure
  - domain-specific, organization-specific, company-specific

Elevator System: Specifications vs Fit Criteria

- **Specication 1**: The doors shall open and close gradually to allow people to easily enter and exit the car.
  Fit Criterion: 90% of time the time (when the lift car is at ground floor) the door shall not take more than 2 seconds to completely open. Otherwise on any other floor door can take 2-3 seconds to open
- **Specication 2:** If the system fails due to heavy load that is more than 1587 kgs, the users should not be put in any danger.
  Fit Criterion: The user using the lift should be alerted and then feedback sound should be given to the user indicating the amount of users to be dismounted from the elevator.
- **Specication 3**: The system should implement an algorithm to minimize average wait times.
  Fit Criterion: The system should implement an algorithm to minimize wait times in such a way that average wait time for an elevator will not exceed 60 seconds.

IEEE Std-830 template for organizing the RD

1. Introduction
   - 1.1 RD purpose
   - 1.2 Product scope
   - 1.3 Definitions, acronyms, abbreviations
   - 1.4 References
   - 1.5 Overview
2. General Description
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 General constraints
   - 2.5 Assumptions & Dependencies
   - 2.6 Apportioning of requirements
3. Specific Requirements

domain, scope, purpose of system-to-be

glossary of terms

elicitation sources

sw-environment boundary: interfaces with users, devices, other sw

functionalities of software-to-be

assumptions about users

development constraints (hw limitations, implem platform, ...)

environment assumptions (subject to change)

optional, deferable reqs

IEEE Std-830 template for organizing the RD (2)

3. Specific Requirements ····· alternative templates for specific types of system

- 3.1 Functional requirements
- 3.2 External interface reqs ····· NFRs: interoperability
- 3.3 Performance reqs ····· NFRs: time/space performance
- 3.4 Design constraints ····· NFRs: development reqs
- 3.5 Software quality attributes ····· NFRs: quality reqs
- 3.6 Other requirements ····· NFRs: security, reliability, maintainability

Appendices

Index

- Variant: VOLERE template [Robertson, 1999]
  - explicit sections for domain properties, costs, risks, development workplan, ...

Example (Elevator in IEEE Std-830: Samples)

1. **Introduction**

   1.1 Purpose

   The purpose of this software requirement specification document is to provide a complete description of the requirements in order to design controller software of an elevator system for a hotel. The intended audience of this document is all of the stakeholders for a project involving the development of elevator controller software. This includes the software development team and the hotel owner, the hotel's architect, the hotel's restaurant's manager, the front desk, the hotel staff, and the hotel's elevator technician.

Example (Elevator in IEEE Std-830: Samples)

1. **Introduction**

   1.2 Scope

   This elevator system is meant to be deployed in the hotel. The controlling software that is specified within this document is responsible for controlling the elevator's movement between floors. This document entails the specification of the software only. The software is not responsible for any of the elevator's physical mechanisms and therefore any requirements relating to said mechanisms are not included in this document. The goal of creating this software is to make an elevator system that is fully operational for use in the hotel. This would make traveling from different floors much faster and much more convenient.

Example (Elevator in IEEE Std-830: Samples)

1. **Introduction**

   1.3 Definition, Acronyms, and Abbreviations

   These definitions will be used throughout the requirements and the rest of the document.

   - **Call button:** This refers to either the "up" or "down" buttons a person presses to call the elevator to a floor.
   - **Stop button:** This refers to the set of buttons located inside the elevator that are pressed to indicate that a person would like the elevator to stop and open its doors on a floor.
   - **Requested floor:** Refers to a floor for which the elevator has been requested to stop at.
   - **Up request:** Refers to when a request is made to move to a floor higher than the elevator currently is (This could be via call button or stop button)
   - **Down request:** Refers to when a request is made to move to a floor lower than the elevator currently is (This could be via call button or stop button)
   - **Floor stop:** Refers to the elevator stopping and opening its doors once when it arrives at a requested floor.

Example (Elevator in IEEE Std-830: Samples)

2. Introduction

2.1 Product Perspective

The controller software specified in this Software Requirements Spec is a part of a larger system - the elevator system. The software therefore must be able to interface with the elevator system's physical mechanisms such as the buttons, the site manager's console, the floor displays, and the elevator display.

2.2 Product Function

This subsections contains a description of the main functions the controller software must have.

a) Control Doors: The controller software decides where and when the elevator closes and opens its doors.

b) Control Movement: The controller software is able to determine how the elevator moves. The software must be able to monitor and influence the speed, acceleration, and direction of movement. The controller is able to determine where the elevator's destination in response to request and/or floor buttons being pressed.

3. Specific Requirements

3.1 Product Perspective

The controller software specified in this Software Requirements Spec is a part of a larger system - the elevator system. The software therefore must be able to interface with the elevator system's physical mechanisms such as the buttons, the site manager's console, the floor displays, and the elevator display.

3.2 Functional Requirements

This subsections contains a description of the main functions the controller software must have.

1. The elevator will make a floor stop if a call button is pressed on said floor. The elevator will make this floor stop at the requested floor in 17 floor stops or less. The elevator will make a floor stop if a stop button is pressed for a certain floor in 17 stops or less provided that the passenger pressed a stop button that causes the elevator to travel in the direction they indicated when they pressed the call button, that is, if they made an up request with the call button they select a higher floor when they press the stop button.

   ....

3. If the elevator door is open and the close door button is pressed, the elevator will make an attempt to close its doors after 3 seconds.

Volere Template

Project Drivers

1. The Purpose of the Project
2. The Client, the Customer, and Other Stackholders
3. Users of the Product

Project Constraints

4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

Functional Requirements

7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

Volere Template

Nonfunctional Requirements

10. Look and Feel Requirements
11. Usability and Humanity Requirements
12. Performance Requirements
13. Operational and Environmental Requirements
14. Maintainability and Support Requirements
15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

Volere Template

Project Issues

18. Open Issues
19. Off-the-Shelf Solutions
20. New Problems
21. Tasks
22. Migration to the New Product
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions

Example (Elevator in Volere Pattern: Samples)

Project Drivers

1. The Purpose of the Project
   a. The User Business or Background of the Project

      Skyhi Lifts manufactures elevators. The company requires a software-based controller to control these elevators.

   b. Goals of the Project

      The goal of the project is to develop a software based controller.
2. The Client, the Customer, and Other Stakeholders
   a. The Client: Skyhi Lifts
   b. The Customer: Real state developers
   c. Other Stakeholders: (1) Business Analysts; (2) Usability Experts; (3) Legal Experts; (4) Developers and Testers; (5) Hardware Vendors; (6) Others

Example (Elevator in Volere Pattern: Samples)

3. Users of the Product

a. The Hands-on Users of the Product
   - Through sensors and buttons general public will be using the product. As the product is embedded software and certain rule is programmed to run, there is no direct interaction with human. Here the users are the sensors, motor and buttons.
   - Maintenance and Service Technicians – needs to change the setup through some interface which is not specified.

b. Priorities assigned to users

   Key Users: Sensors, monitor, and buttons

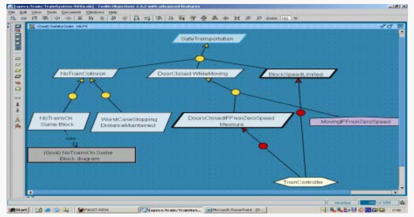
   Secondary Users: Maintenance and Service Technicians

c. User Participation: Not applicable.

d. Maintenance User and Service Technicians

   In order to service elevators the maintenance users and service technicians requires the product to have some features in this regard.

Use of diagrammatic notations

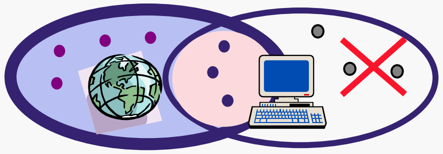


- To complement or replace NL prose
- Dedicated to specific aspects of the system (as-is or to-be)
- Graphical: to ease communication, provide overview
- Semi-formal ...
  - Declaration of items in formal language (syntax, semantics)

    => surface checks on RD items, machine-processable
  - Informal spec of item properties in NL
- This lecture: typical sample of frequently used diagrams, showing complementarities

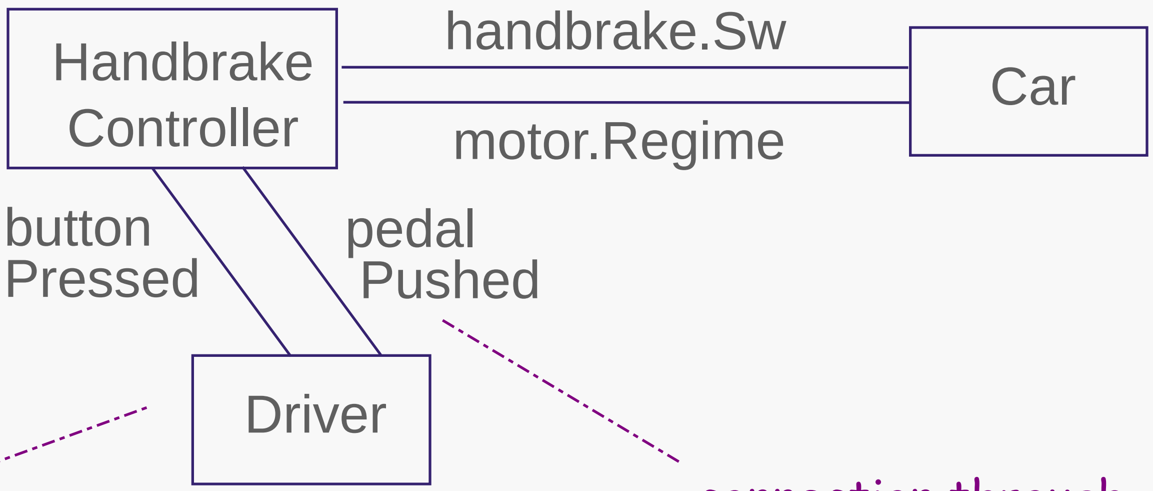
Requirements specification & documentation: outline

- Free documentation in unrestricted natural language
- Disciplined documentation in structured natural language
  - Local rules on writing statements
  - Global rules on organizing the Requirements Document
- Use of diagrammatic notations
  - System scope: context, problem, frame diagrams
  - Conceptual structures: entity-relationship diagrams
  - Activities and data: SADT diagrams
  - Information flows: dataflow diagrams
  - System operations: use case diagrams
  - Interaction scenarios: event trace diagrams
  - System behaviors: state machine diagrams
  - Stimuli and responses: R-net diagrams
  - Integrating multiple system views, multi-view spec in UML



System scope: context diagrams

- Declare system components & their interfaces [DeMarco '78]
  => system structure
  what is in system, what is not
  environment of each component: neighbors, interfaces

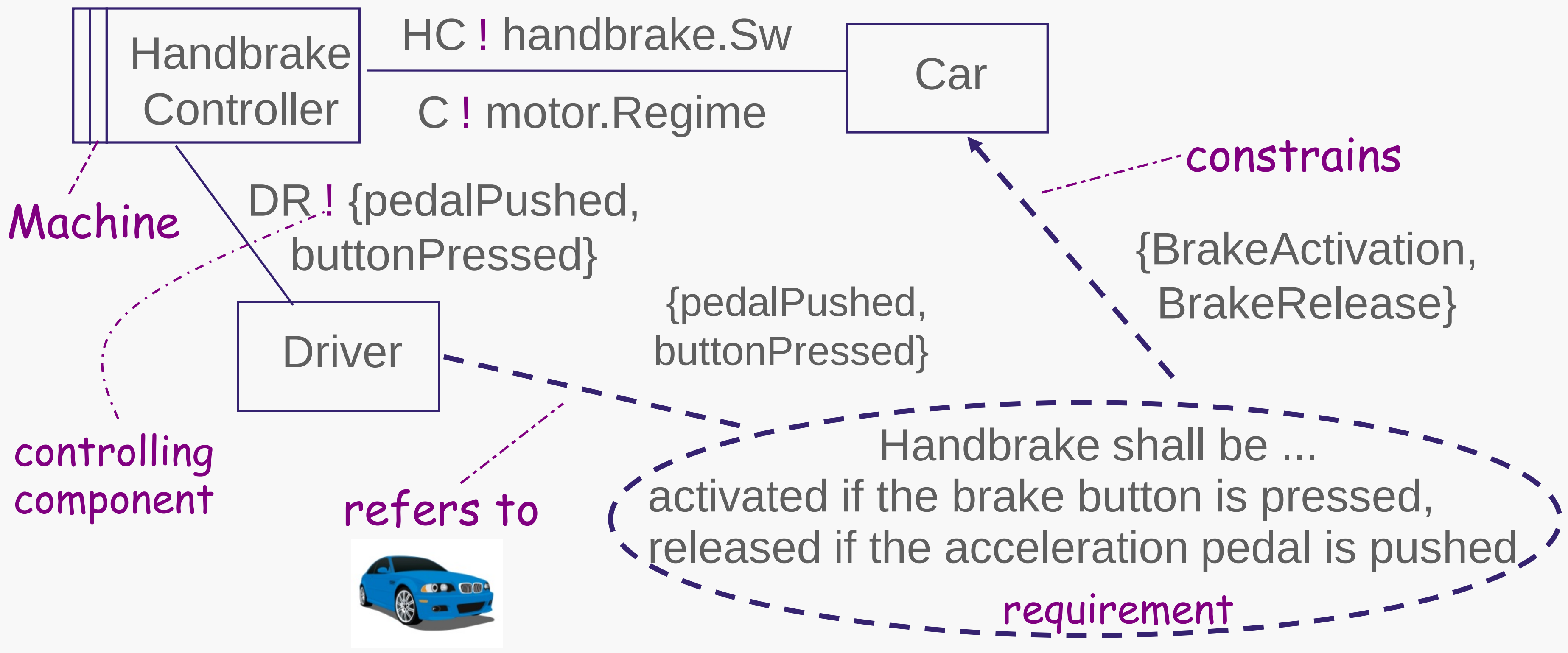


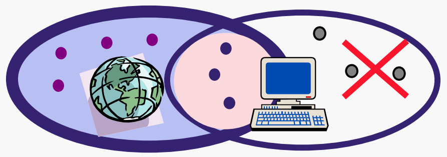
system component

connection through shared phenomenon (data, event)

System scope: problem diagrams

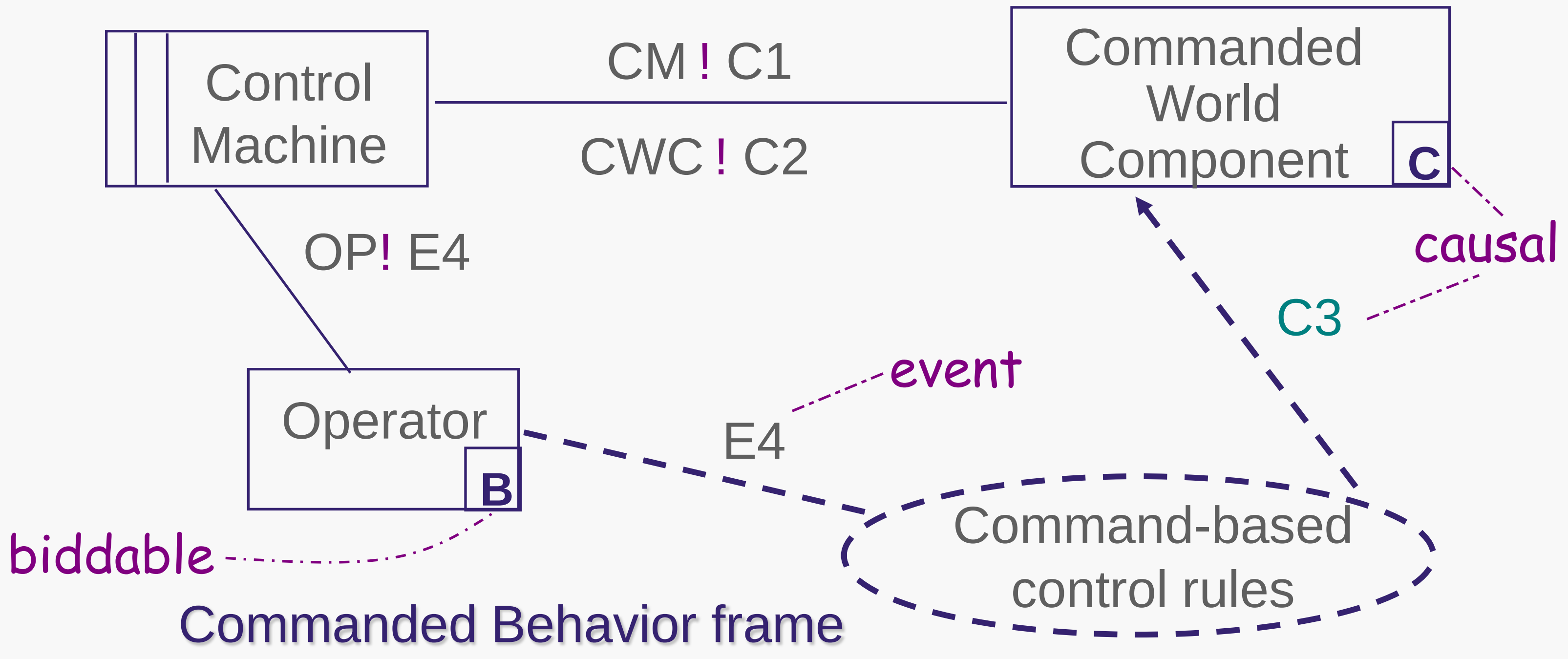
- More detailed form of context diagram: highlights...
  - the Machine among system components
  - for shared phenomenon: who controls it, who monitors it
  - requirements, components affected by them

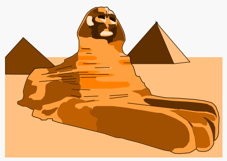

Handbrake Controller
HC ! handbrake.Sw
C ! motor.Regime
Car
Machine
constrains
DR ! {pedalPushed, buttonPressed}
{BrakeActivation, BrakeRelease}
Driver
{pedalPushed, buttonPressed}
controlling component
refers to
Handbrake shall be ...
activated if the brake button is pressed,
released if the acceleration pedal is pushed
requirement



System scope: frame diagrams

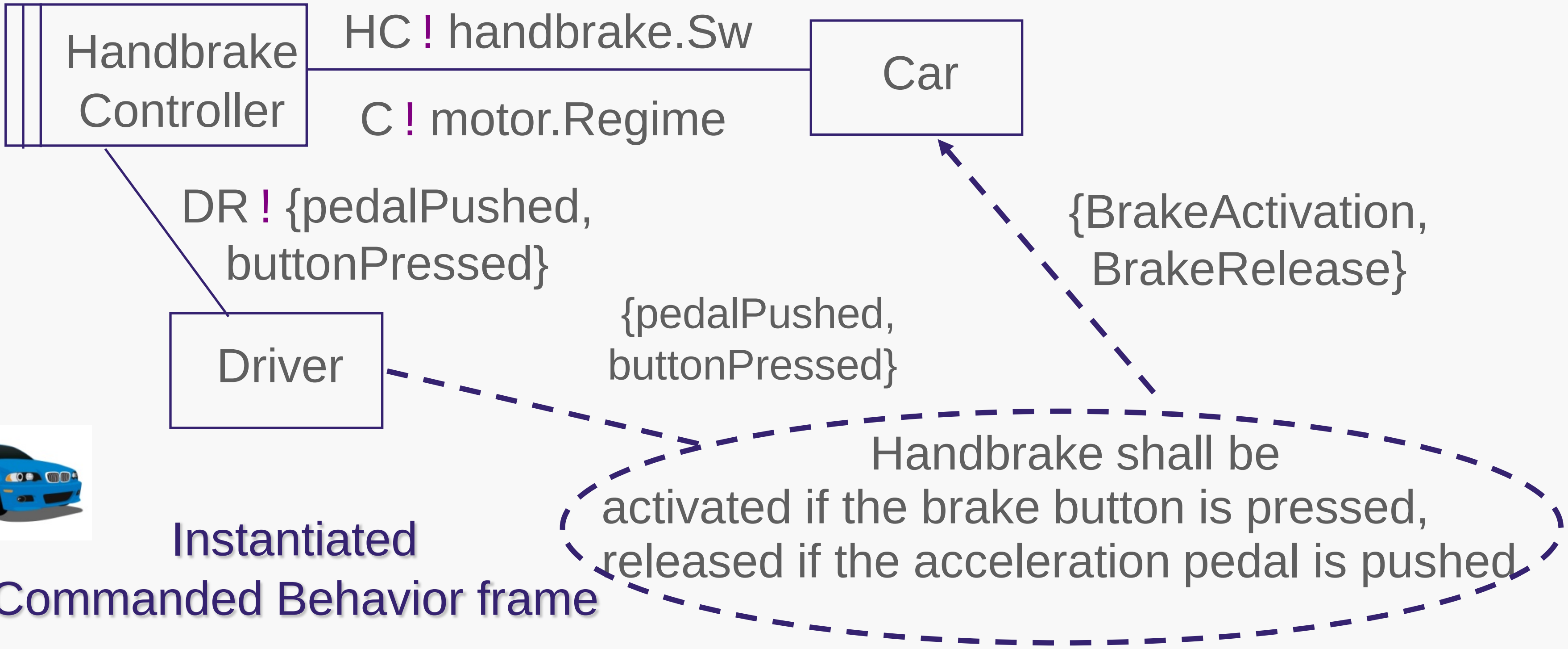
- Capture frequent problem patterns
  - typed phenomena (C: causal, E: event, Y: symbolic)
  - typed components (C: causal, B: biddable, X: lexical)
- E.g. Simple Workpieces, Information Display, Commanded Behavior (see book [RE])





Reusing problem frames

- Candidate system-specific problem diagram can be obtained by instantiation, in matching situations
  - under typing constraints
  - mutiple frames reusable for same problem world


Handbrake Controller
HC ! handbrake.Sw
C ! motor.Regime
Car
DR ! {pedalPushed, buttonPressed}
Driver
{pedalPushed, buttonPressed}
{BrakeActivation, BrakeRelease}
Handbrake shall be activated if the brake button is pressed, released if the acceleration pedal is pushed

Instantiated
Commanded Behavior frame

Conceptual structures: entity-relationship diagrams

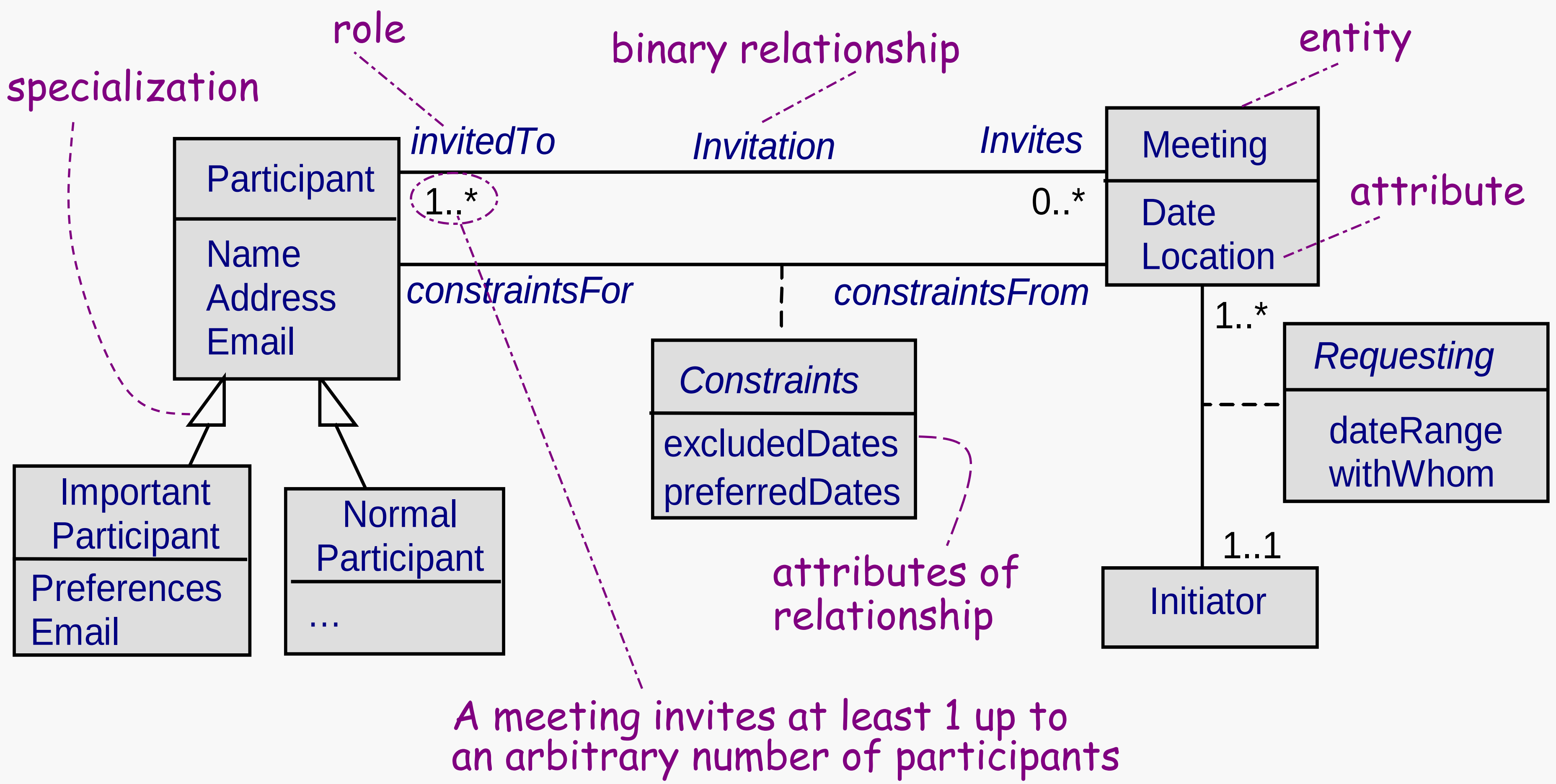
- Declare conceptual items, structure them
- **Entity**: class of concept instances ...
  - having distinct identities
  - sharing common features (attributes, relationships)

  e.g. Meeting, Participant
- N-ary **relationship**: feature conceptually linking N entities, each playing a distinctive role (N ≥ 2)
  - **Multiplicity**, one one side: min & max number of entity instances, on this side, linkable at same time to single tuple of entity instances on the other sides

  e.g. Invitation linking Participant and Meeting
- **Attribute**: feature intrinsic to an entity or a relationship
  - has range of values

  e.g. Date of Meeting

Entity-relationship diagram: example



Multiplicities may capture requirements or domain properties

☹ No distinction between prescriptive & descriptive

Entity-relationship diagrams (2)

- Entity specialization: subclass of concept instances, further characterized by specific features (attributes, relationships)
  - by default, inherits attributes & relationships from superclass
  - rich structuring mechanism for factoring out structural commonalities in superclasses

  e.g. ImportantParticipant, with specific attribute Preferences

  Inherits relationships Invitation, Constraints, attribute Address

  (Email of ImportantParticipant inhibits default inheritance)

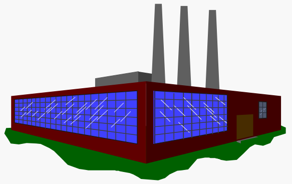
- Diagram annotations: to define elements precisely
  - essential for avoiding spec errors & flaws

  e.g. annotation for Participant:

  "Person expected to attend the meeting, at least partially, under some specific role. Appears in the system when the meeting is initiated and disappears when the meeting is no longer relevant to the system"

Requirements specification & documentation: outline

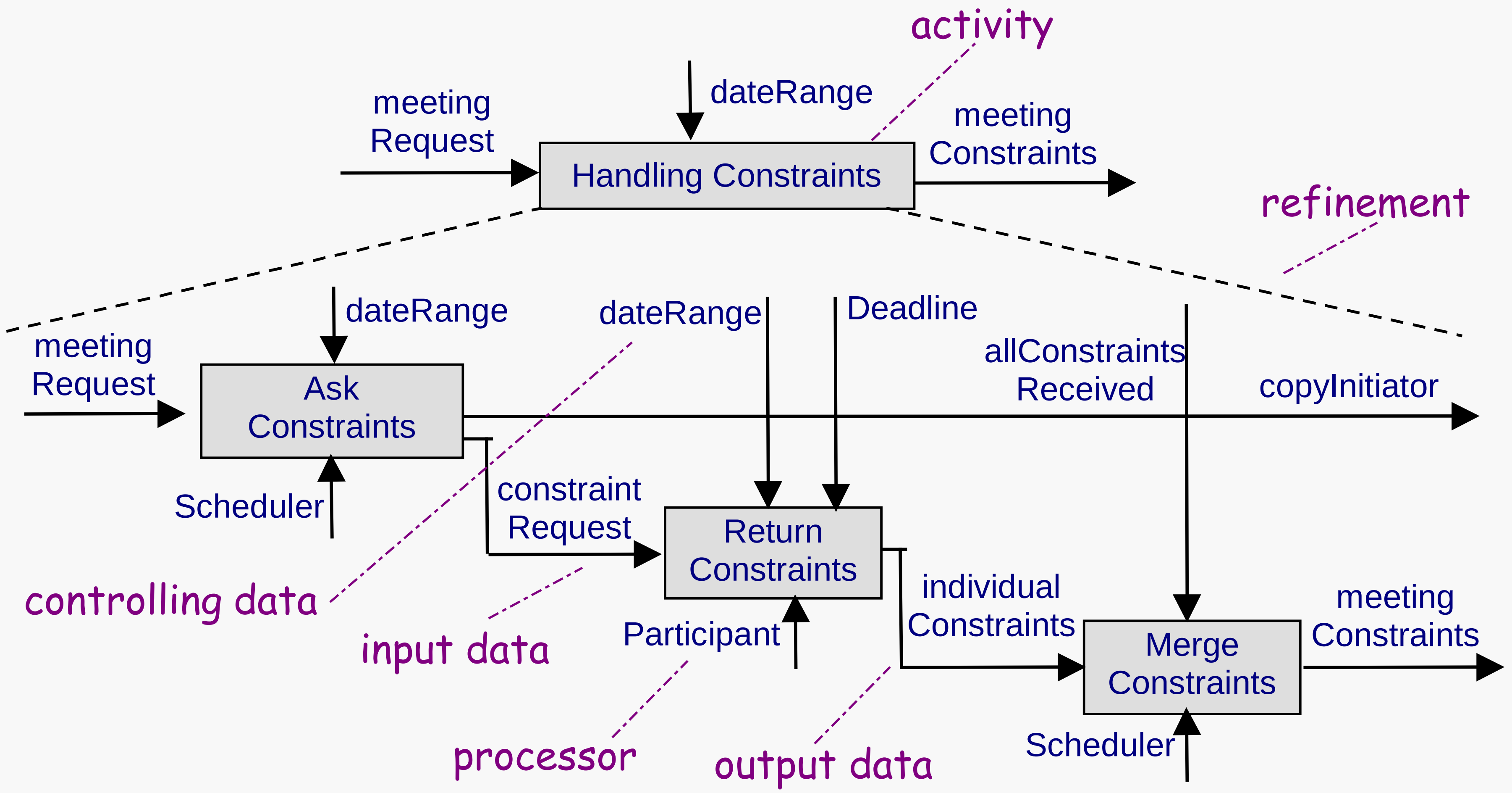
- Free documentation in unrestricted natural language
- Disciplined documentation in structured natural language
  - Local rules on writing statements
  - Global rules on organizing the Requirements Document
- Use of diagrammatic notations
  - System scope: context, problem, frame diagrams
  - Conceptual structures: entity-relationship diagrams
  - Activities and data: SADT diagrams
  - Information flows: dataflow diagrams
  - System operations: use case diagrams
  - Interaction scenarios: event trace diagrams
  - System behaviors: state machine diagrams
  - Stimuli and responses: R-net diagrams
  - Integrating multiple system views, multi-view spec in UML



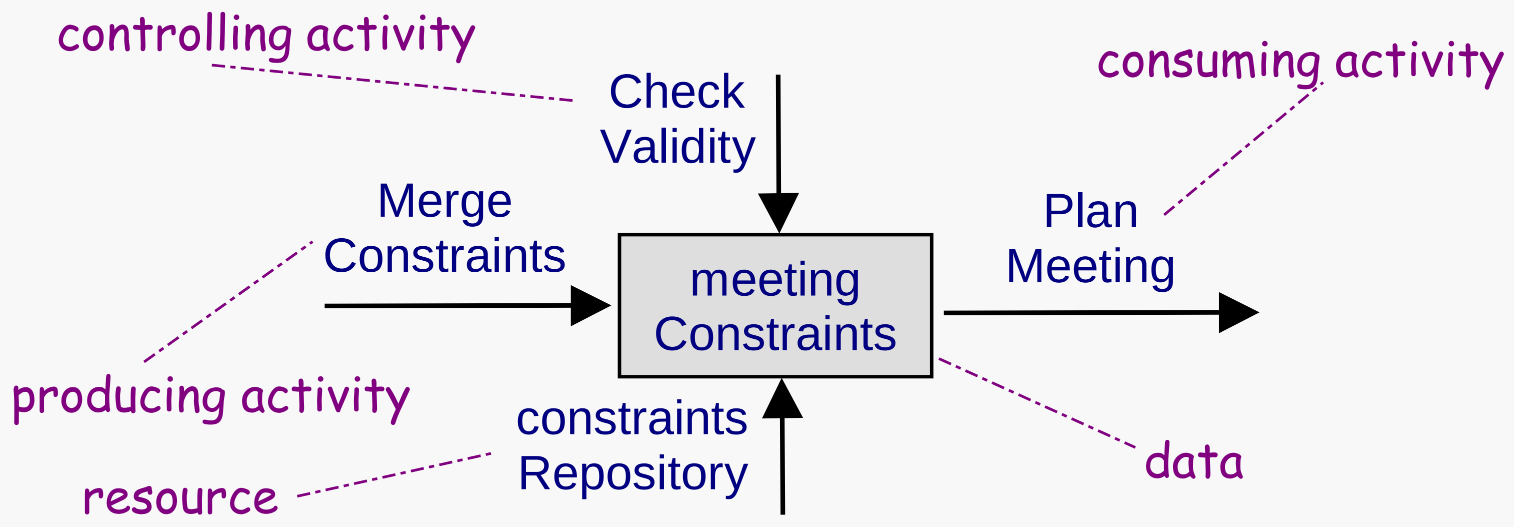
Activities and data: SADT diagrams

- Capture activities & data in the system (as-is or to-be)
- Actigram: relates activities through data dependency links
  - East $\rightarrow$: input data; West $\rightarrow$: output data
  - North $\rightarrow$: controlling data/event; South $\rightarrow$: processor
  - Activities refinable into sub-activities
- Datagram: relates data through control dependency links
  - East $\rightarrow$: producing activity; West $\rightarrow$: consuming activity
  - North $\rightarrow$: validation activity; South $\rightarrow$: needed resources
  - Data refinable into sub-data
- Data-activity duality:
  - data in actigram must appear in datagram
  - activities in datagram must appear in actigram

SADT diagrams: actigram example



SADT diagrams: datagram example

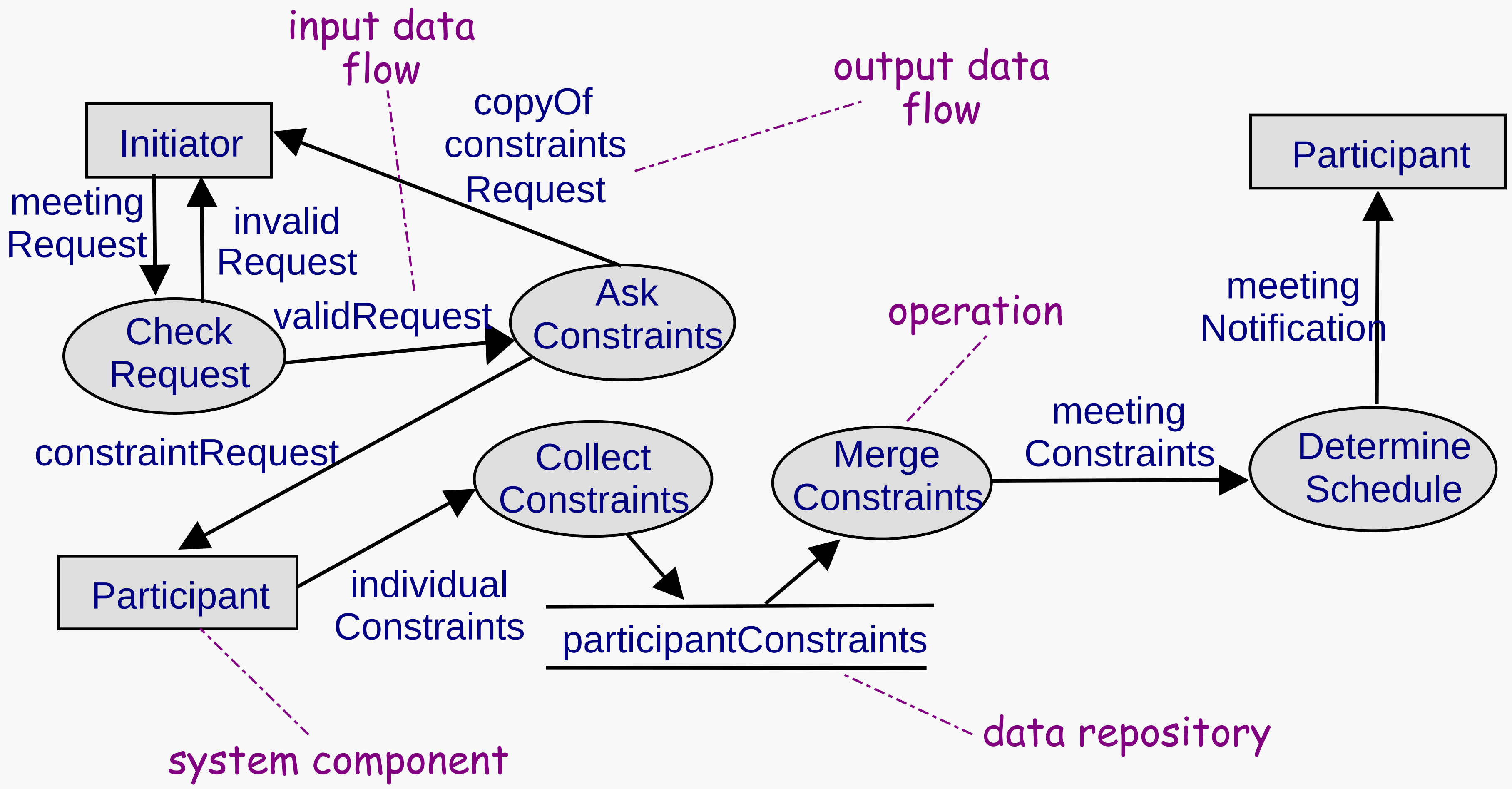


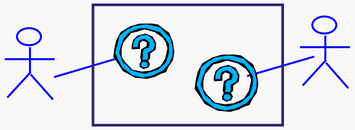
- Consistency/completeness rules checkable by tools
  - Every activity must have an input and an output
  - All data must have a producer and a consumer
  - I/O data of an activity must appear as I/O data of subactivities
  - Every activity in a datagram must be defined in an actigram, ...

Information flows: dataflow diagrams

- Capture system operations linked by data dependencies
  - simpler but less expressive than actigrams
- Operation = data transformation activity
- Input, output links = data flows
  - operation needs data flowing in to produce data flowing out (≠ control flow !)
- Data transformation rule to be specified ...
  - in annotation (structured NL)
  - or in another DFD (operation refinement, cf. SADT)
- System components, data repositories = origins, ends of flow
- Consistency/completeness rules checkable by tools, cf. SADT

Dataflow diagram: example

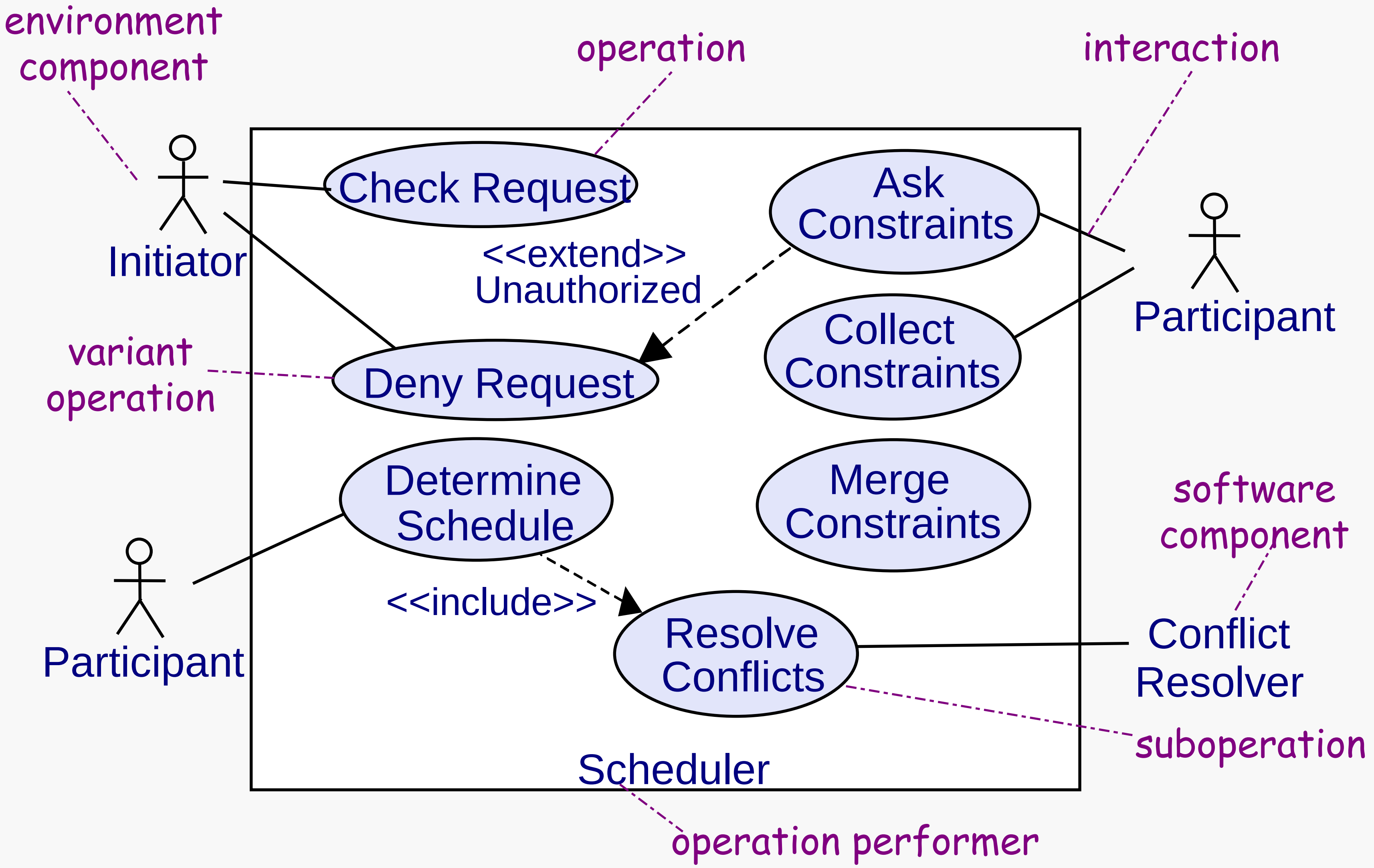




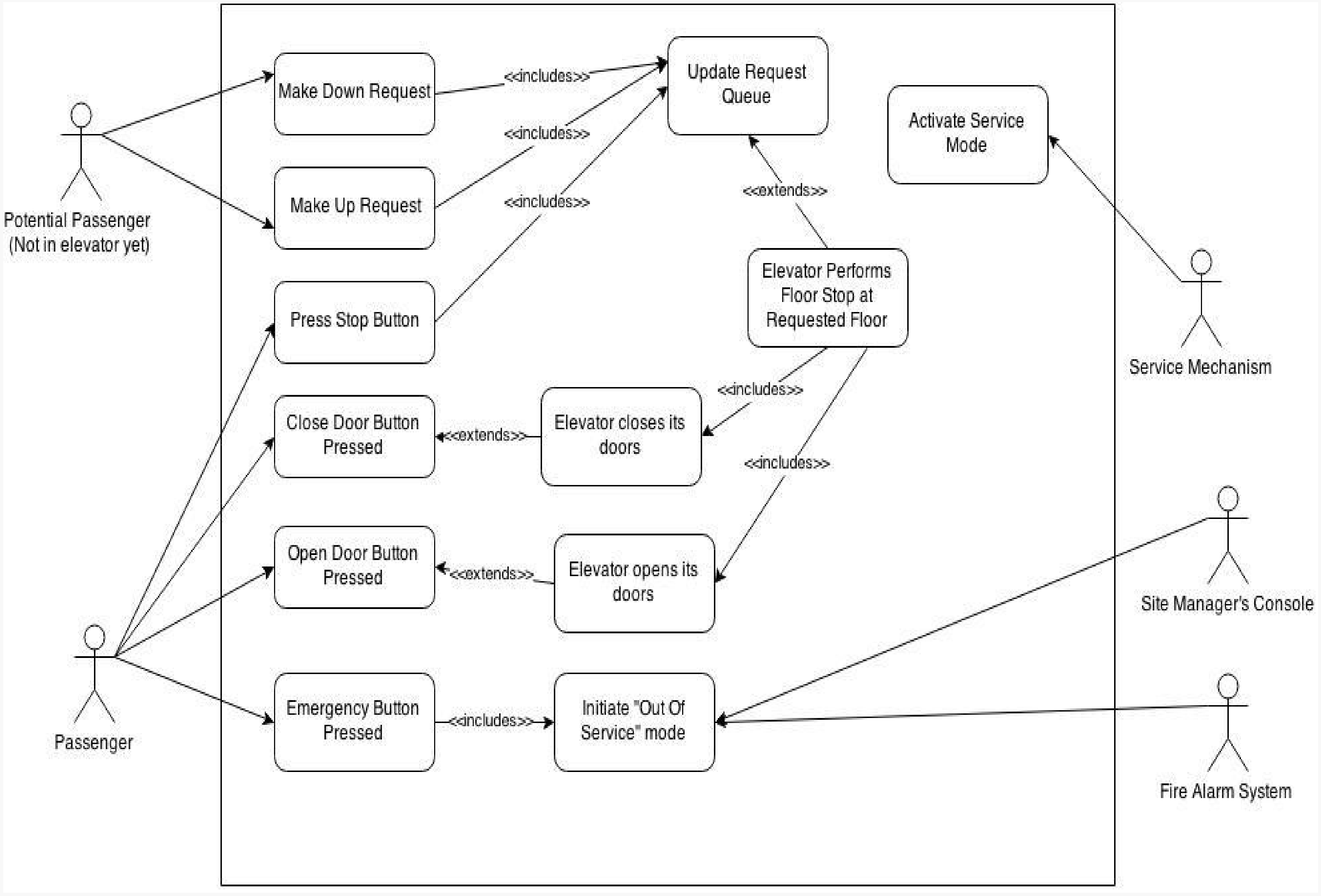
System operations: use case diagrams

- Capture operations to be performed by a system component & interactions with other components
  - yet simpler, outline view ... but vague
  - to be made precise by annotations, interaction scenarios, ...
  - introduced in UML to replace DFDs
- Structuring mechanisms ...
  - <<include>>: to specify "suboperation"
  - <<extend>> + precondition: to specify "variant" operation in exception case

Use case diagram: example



Example Use case diagram: Elevator System



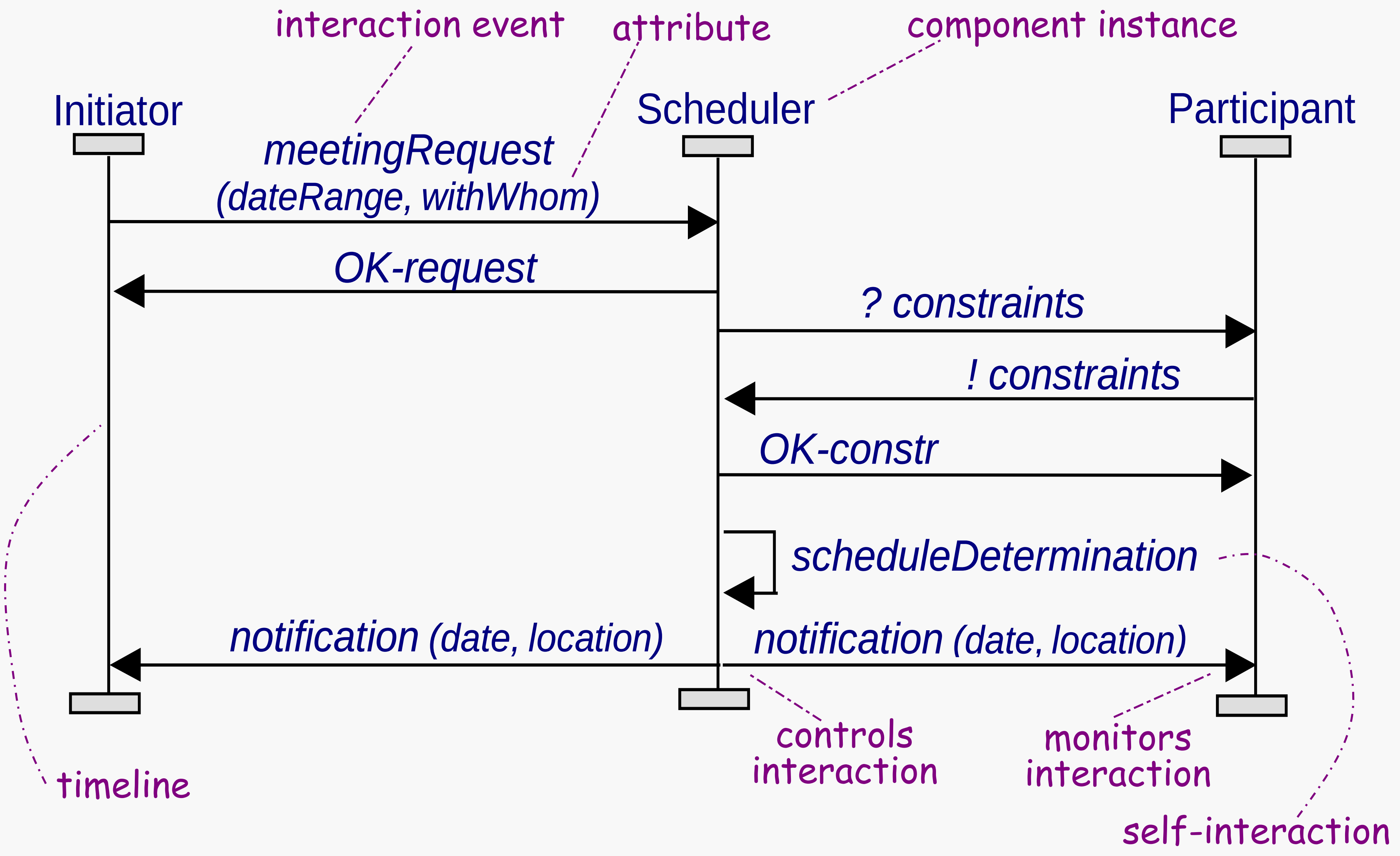
Requirements specification & documentation: outline

- Free documentation in unrestricted natural language
- Disciplined documentation in structured natural language
  - Local rules on writing statements
  - Global rules on organizing the Requirements Document
- Use of diagrammatic notations
  - System scope: context, problem, frame diagrams
  - Conceptual structures: entity-relationship diagrams
  - Activities and data: SADT diagrams
  - Information flows: dataflow diagrams
  - System operations: use case diagrams
  - Interaction scenarios: event trace diagrams
  - System behaviors: state machine diagrams
  - Stimuli and responses: R-net diagrams
  - Integrating multiple system views, multi-view spec in UML

Interaction scenarios: event trace diagrams

- Capture positive scenarios by sequences of interactions among instances of system components
  - variants: MSC (ITU), UML sequence diagrams
- Parallel composition of timelines
  - one per component instance
- Pairwise directed interactions down timelines
  - information transmission through event attributes
- Interaction event synchronously controlled by source instance & monitored by target instance
  - total order on events along timeline (event precedence)
  - partial order on all diagram events

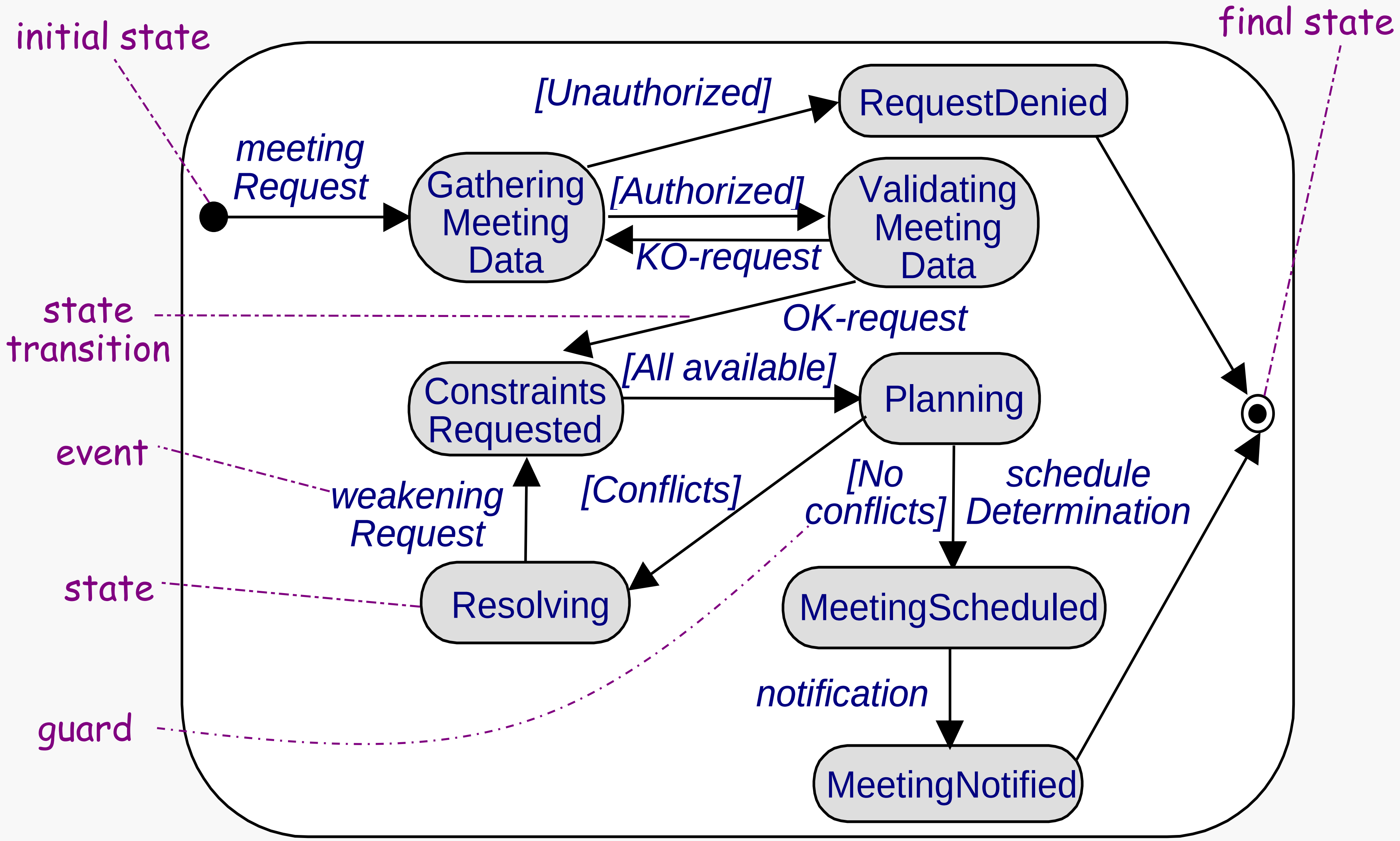
Event trace diagram: example

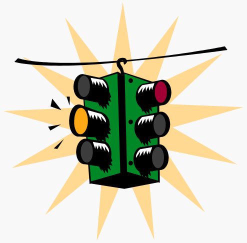


System behaviors: state machine diagrams

- Capture the admissible behaviors of system components
- **Behavior** of component instance =
  sequence of state transitions for the items it controls
- SM **state** = set of situations where a variable characterizing
  a controlled item has always the same value
  - e.g. state MeetingScheduled: always same value for Date, Location
    (while other variable WithWhom on Meeting may change value)
  - **Initial, final** states = states where item appears, disappears
  - States may have some duration
- SM **state transition**: caused by associated event
  - **if** item in source state and event ev occurs
    then it gets to target state
  - Events are instantaneous phenomena

Example of state machine diagram:
meeting controlled by a meeting scheduler





State machine diagrams: transitions and guards

- Event occurrence is a sufficient condition for transition firing
  - Event can be external stimulus (e.g. meetingRequest) or application of internal operation (e.g. determineSchedule)
- Guard = necessary condition for transition firing
  - Item gets to target state ...

    if item is in source state and event ev occurs

    and only if guard condition is true
  - Guarded transition with no event label:

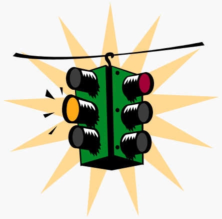
    fires as soon as guard gets true (= trigger condition)
- Non-deterministic behavior: multiple outgoing transitions with same event and no or overlapping guards
  - often to be avoided for safety, security reasons

Scenarios and state machines

- SM trace = sequence of successive SM states up to some point
  - e.g. < GatheringMeetingData, RequestDenied >
  - always finite, but SM diagram may have infinitely many traces
- A SM diagram generalizes ET diagram scenarios:
  - from specific instances to any component instance
  - trace coverage: SM traces include ET traces, and (many) more

    e.g. scenario/SM trace from previous slides:

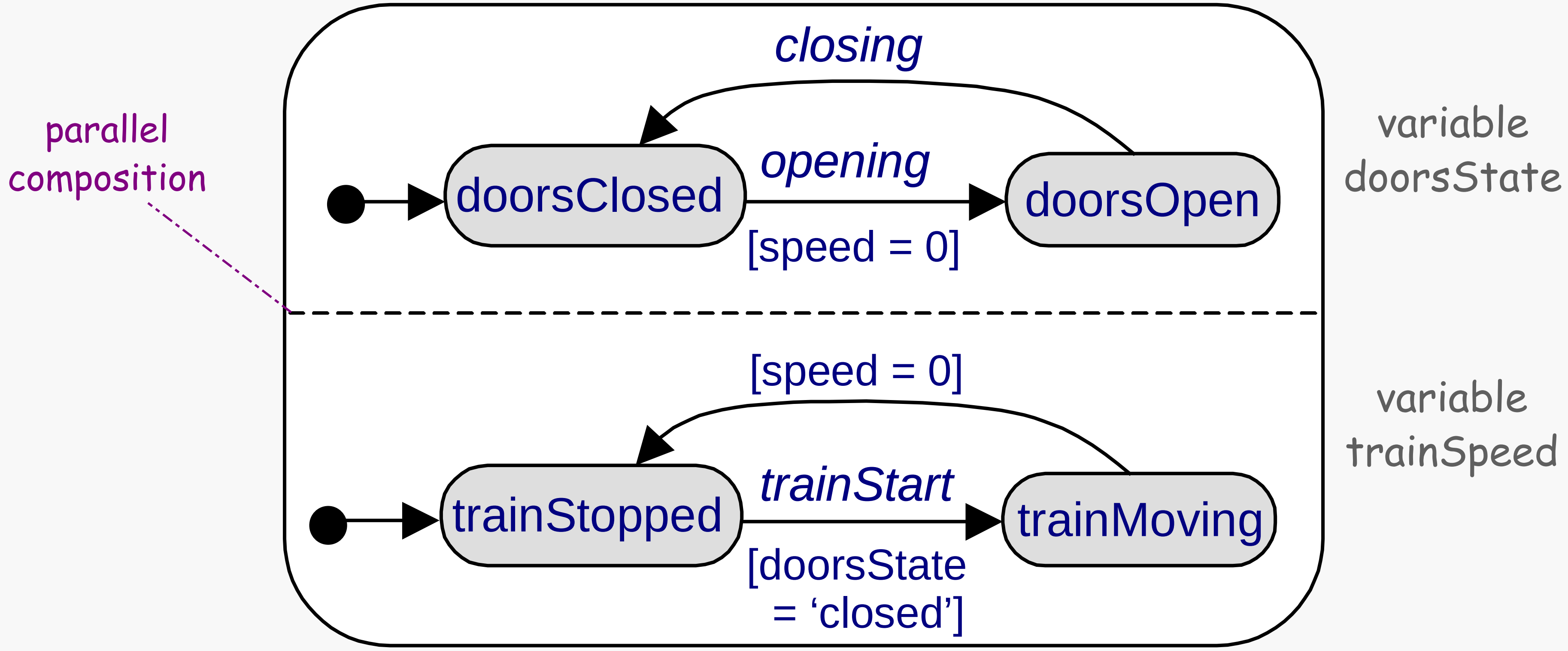
    < ValidatingMeetingData; ConstraintsRequested; Planning;
    MeetingScheduled; MeetingNotified >



Concurrent behaviors and statecharts

- Components often control multiple items in parallel
- Problems with flat SM diagram …
  - N item variables each with M values => M^N states !
  - same SM state mixing up different variables
- Statechart = parallel composition of SM diagrams [Harel, 1987]
  - one per variable evolving in parallel
  - statechart state = aggregation of concurrent substates
  - from M^N explicit SM states to $M \times N$ statechart states !
- Statechart trace = sequence of successive aggregated SM states up to some point
- Interleaving semantics: for 2 transitions firing in same state, one is taken after the other (non-deterministic choice)

Statechart example

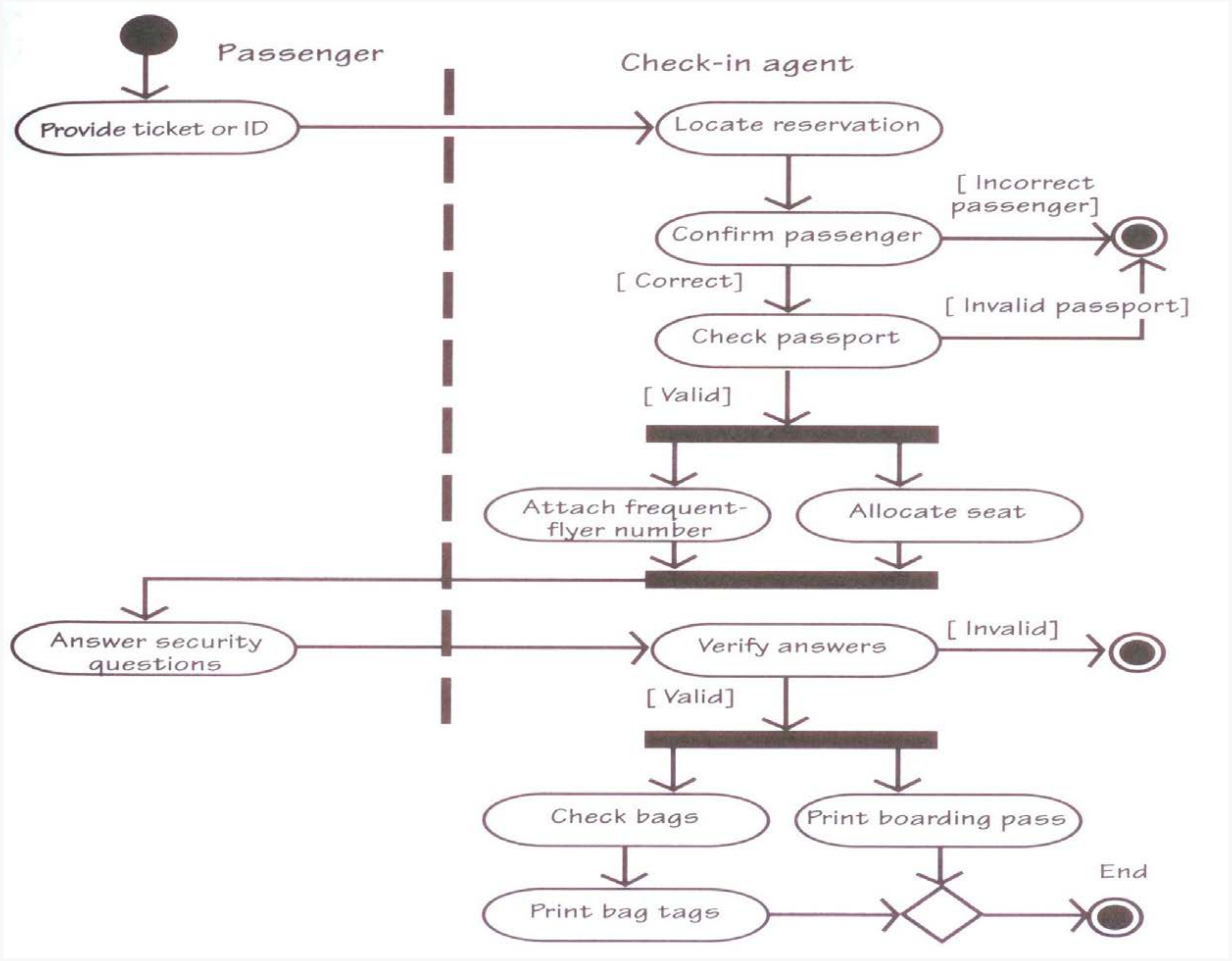


- Trace example:
  < (doorsClosed, trainStopped); (doorsClosed, trainMoving);
  (doorsClosed, trainStopped); (doorsOpen, trainStopped) >
- Model-checking tools can generate counterexample traces leading to violation of desired property (cf. Chap. 5 in [RE])

UML Activity Diagrams

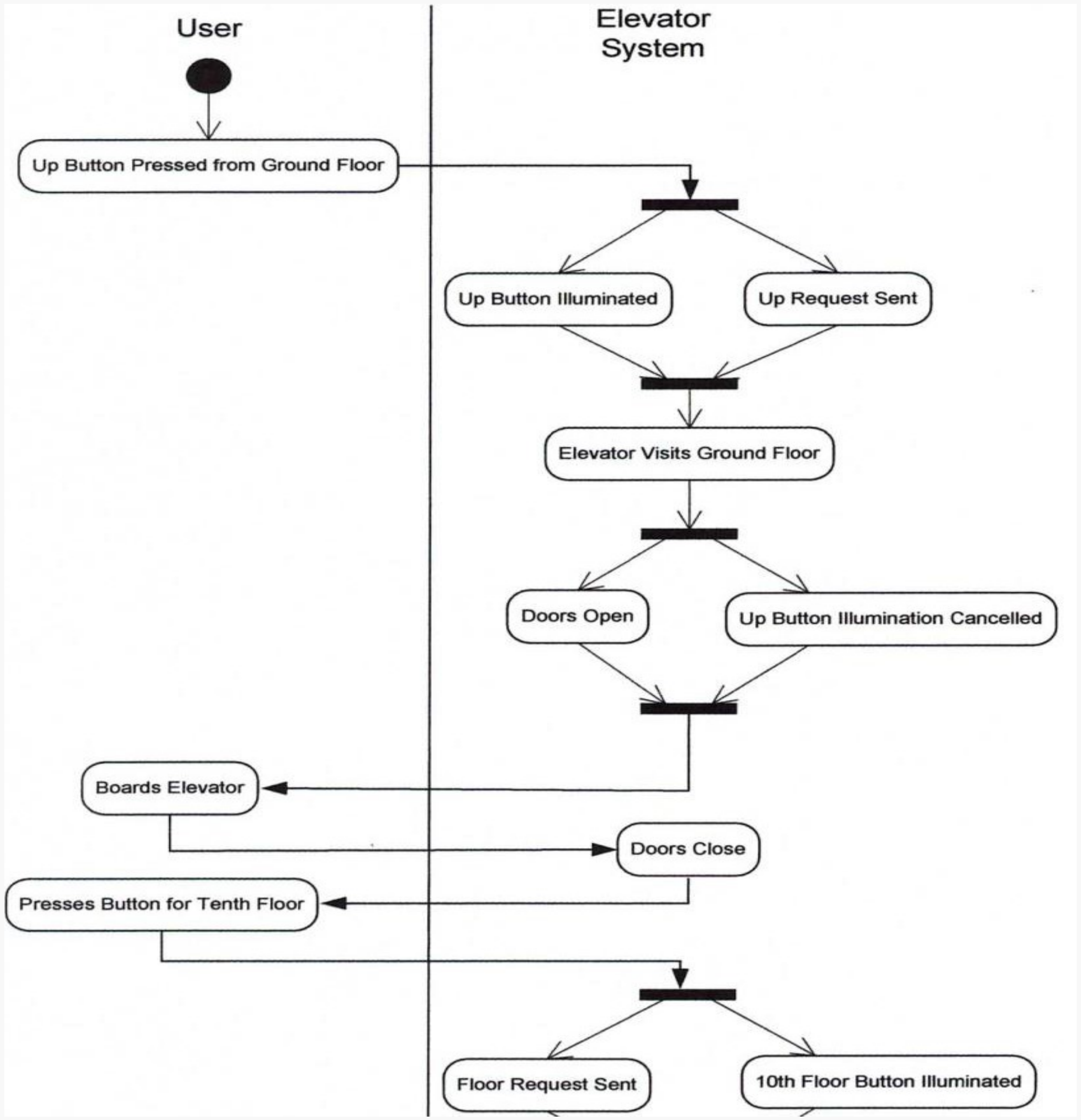
- Activity diagrams are specialized state machine diagrams and a modified version of classical flowcharts.
- Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity.
- An activity represents an operation on some class in the system that results in a change in the state of the system.
- Typically, activity diagrams are used to model *work flow* or *business processes* and *internal operation*.
- Basic elements in activity diagrams are ***activities***, ***branches*** (***conditions*** or ***selections***), ***transitions***, ***forks*** and ***joins*** (to model parallelism), and ***swim lines***.
- Swim lanes are useful when a designer wants to express how related activities can be grouped

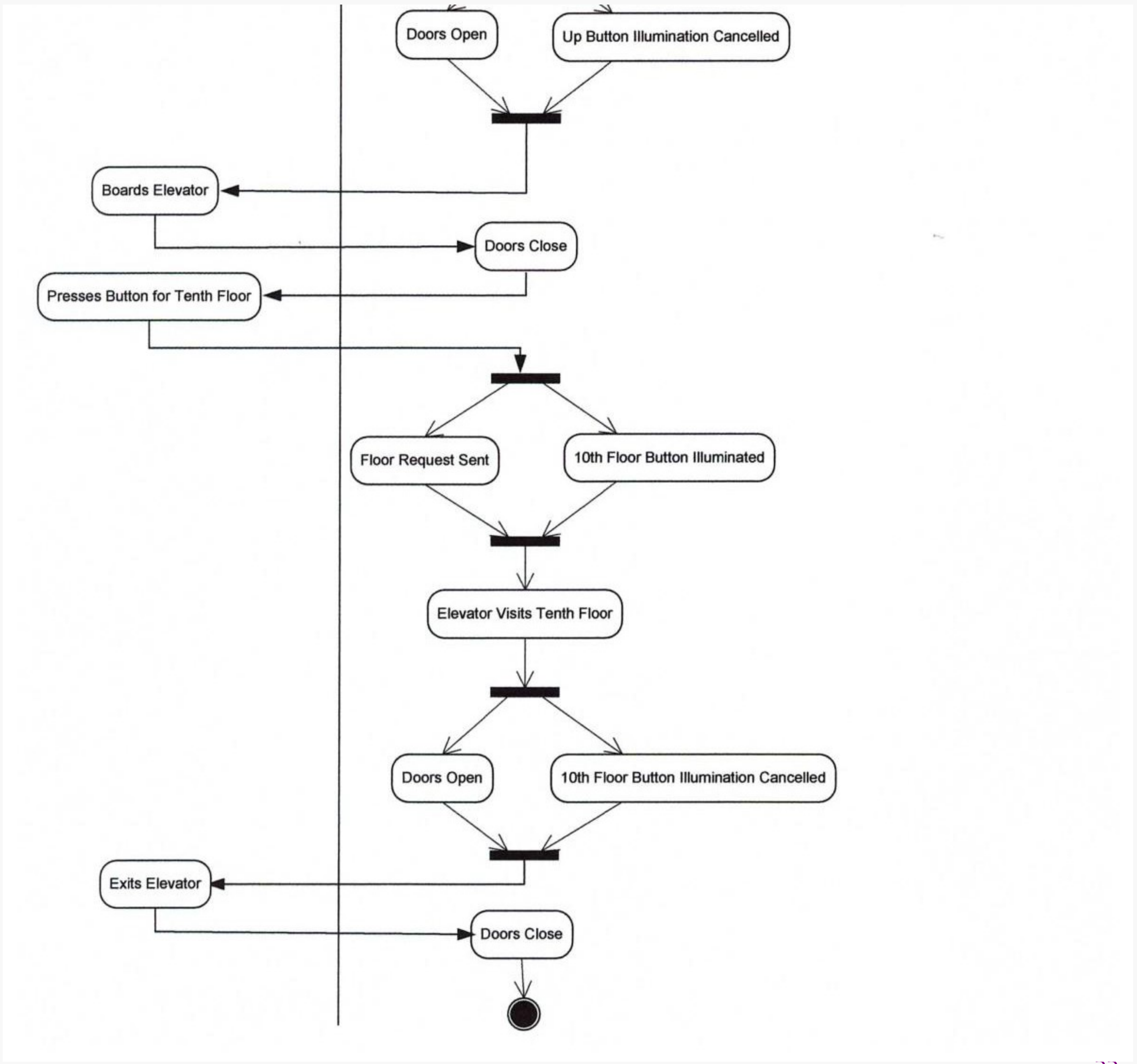
UML Activity Diagram: Checking for a flight



Modeling Scenarios using Activity Diagrams

Example (Elevator: Positive Scenario)

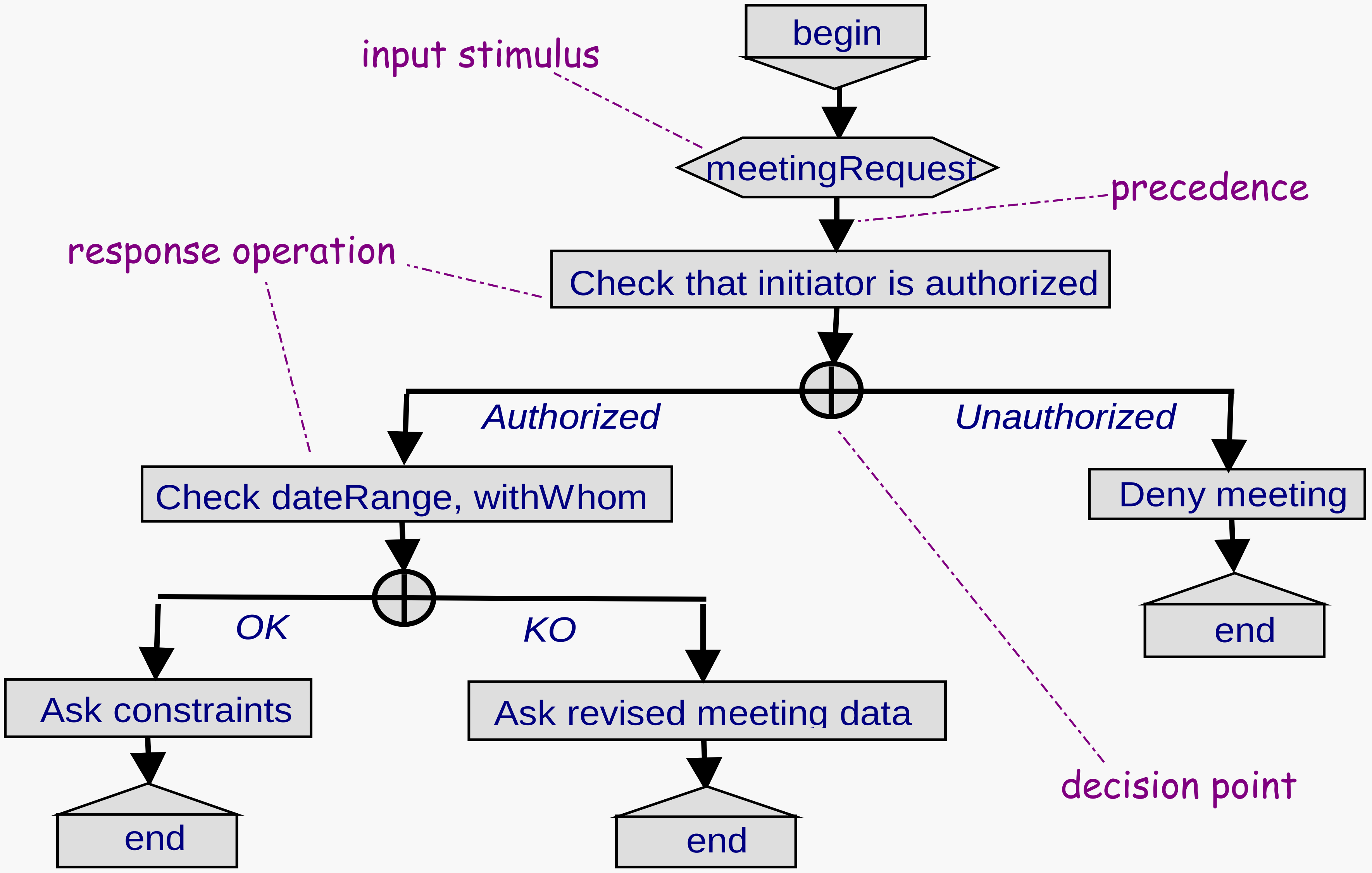

User
Elevator System
Up Button Pressed from Ground Floor
Up Button Illuminated
Up Request Sent
Elevator Visits Ground Floor
Doors Open
Up Button Illumination Cancelled
Boards Elevator
Doors Close
Presses Button for Tenth Floor
Floor Request Sent
10th Floor Button Illuminated

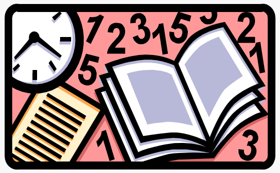

Doors Open
Up Button Illumination Cancelled
Boards Elevator
Doors Close
Presses Button for Tenth Floor
Floor Request Sent
10th Floor Button Illuminated
Elevator Visits Tenth Floor
Doors Open
10th Floor Button Illumination Cancelled
Exits Elevator
Doors Close

Stimuli and responses: R-net diagrams

- Capture all required responses to single stimulus [Alford, 1977]
  - chain of response operations to be performed by a system component
  - operation may generate stimuli for other R-nets
- Decision points, operation application under conditions
- Good for visualizing ...
  - answers to WHAT IF ? questions
  - required software reactions to environment events

R-net diagram: example





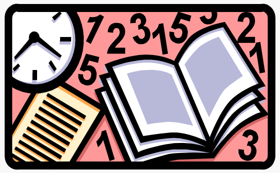
Integrating multiple system views

- Diagrams of different types cover different, complementary views of the system (as-is or to-be)
  - components & interfaces, conceptual structures, operations, flows, interaction scenarios, behaviors, ...
- Overlapping aspects => integration mechanism needed for ensuring compatibility & complementarity among diagrams
- Standard mechanism: **inter-view consistency rules** the specifier should meet
  - cf. static semantics rules enforced by compilers

    "every used variable must be declared"
    "every declared variable must be used", ...
  - can be used for inspection checklists
  - enforceable by tools
  - constrain diagram evolution

Inter-view consistency rules: examples

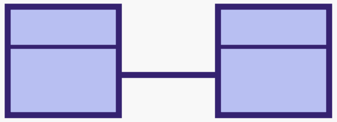
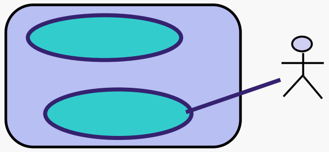
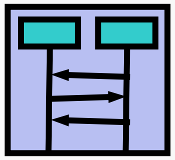
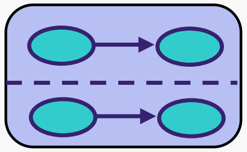
- Every component & interconnection in a problem diagram must be further specified in an ET diagram
- Every shared phenomenon in a problem diagram must appear as event in an ET diagram or as entity, attribute, or relationship in an ER diagram
- Every data in a flow or repository of a DFD diagram must be declared as entity, attribute, or relationship in an ER diagram
- Every state in a SM diagram must correspond to some value for some attribute or relationship in an ER diagram
- Every interaction event in an ET scenario must appear in a corresponding SM diagram



Multi-view specification in UML

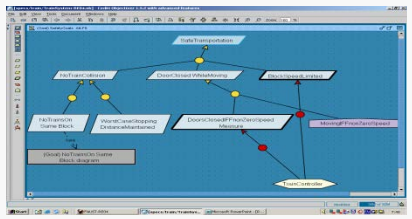
The Unified Modeling Language (UML) has standardized notations for diagrams relevant to RE

- Class diagrams: ER diagrams for structural view
- Use case diagrams: outline of operational view
- Sequence diagrams: ET diagrams for scenarios
- State diagrams: SM diagrams for behavioral view
- Activity diagrams: Scenarios, work flows, internal operations

Further studied in next lectures in a systematic method for building multi-view models

Diagrammatic notations: pros & cons

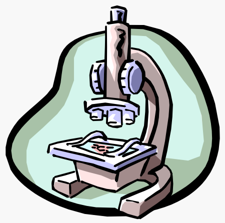


- ◆ Formal declaration of different system facets
  + informal annotations of properties for higher precision
- ◆ Graphical declaration =>
  - ☺ overview & structuring of important aspects
  - ☺ easy to understand, communicate
  - ☺ surface-level analysis, supported by tools (e.g. query engines)
- ◆ Semi-formal specification =>
  - ☹ language semantics may be vague (different interpretations)
  - ☹ only surface-level aspects formalized, not item properties
  - ☹ limited forms of analysis
  - ☹ functional and structural aspects only

=> formal specification needed for mission-critical aspects

Requirements specification & documentation (1): summary

- Free documentation in unrestricted NL is subject to errors & flaws
- Disciplined documentation in structured NL is always necessary
  - Local rules on statements: stylistic rules, decision tables, statement templates
  - Global rules on RD organization: grouping rules, structure templates
- Diagrams for graphical, semi-formal spec of complementary aspects
  - System scope: context, problem, frame diagrams
  - Conceptual structures: entity-relationship diagrams
  - Activities and data: SADT diagrams
  - Information flows: dataflow diagrams
  - System operations: use case diagrams
  - Interaction scenarios: event trace diagrams
  - System behaviors: state machine diagrams
  - Stimuli and responses: R-net diagrams
  - Integrating multiple views, multi-view spec in UML



Requirements specification & documentation (2): formal specification

- Logic as a basis for formalizing statements
- History-based specification
- State-based specification
- Event-based specification
- Algebraic specification