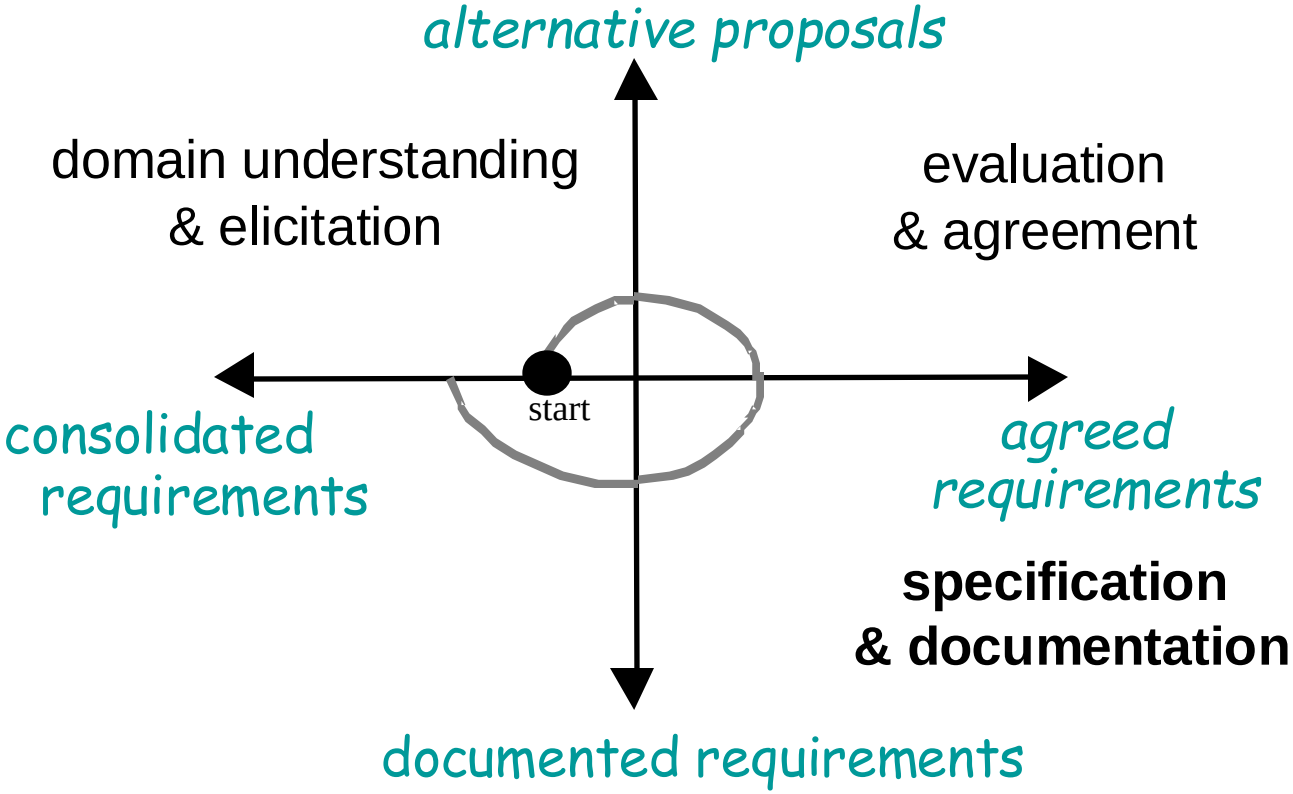
Requirements Specification & Documentation

Mariano Ceccato

mariano.ceccato@univr.it

The RE products and process



Specification & documentation

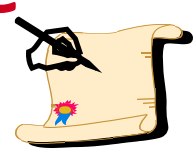
- Precise definition of all features of the agreed system
  - Objectives, concepts, relevant domain properties, system/software requirements, assumptions, responsibilities
  - Rationale for options taken, satisfaction arguments
  - Likely system evolutions & variants
- Organization of these in a coherent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

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

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
  - Integrating multiple system views, multi-view spec in UML

Free doc. 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  "Full braking shall be activated by any train that receives an outdated acceleration command **or** that enters a station block at speed higher than *X* km/h **and for which** the preceding train is closer than *Y* meters."
- Frequent confusions among logical connectives in NL
  - e.g. case analysis:

If Case1 **then** <Statement1>
or **if** Case2 **then** <Statement2>

(evaluates to **true**!)

vs.

If Case1 **then** <Statement1>
and **if** Case2 **then** <Statement2>

Disciplined doc. in structured NL: local rules

- 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

Local rules

- 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)

Local rules

- 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
- 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

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

3. Specific Requirements — *alternative templates for specific types of system*

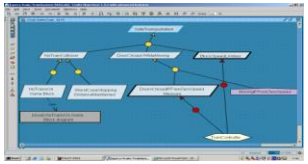
- 3.1 Functional requirements
- 3.2 External interface reqs — NFRs: interoperability
- 3.3 Performance reqs — NFRs: time/space performance
- 3.4 Design constraints — NFRs: development reqs
- 3.5 Software quality attributes — NFRs: quality reqs
- 3.6 Other requirements — NFRs: security, reliability, maintainability

Appendices

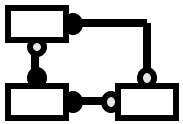
Index

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

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

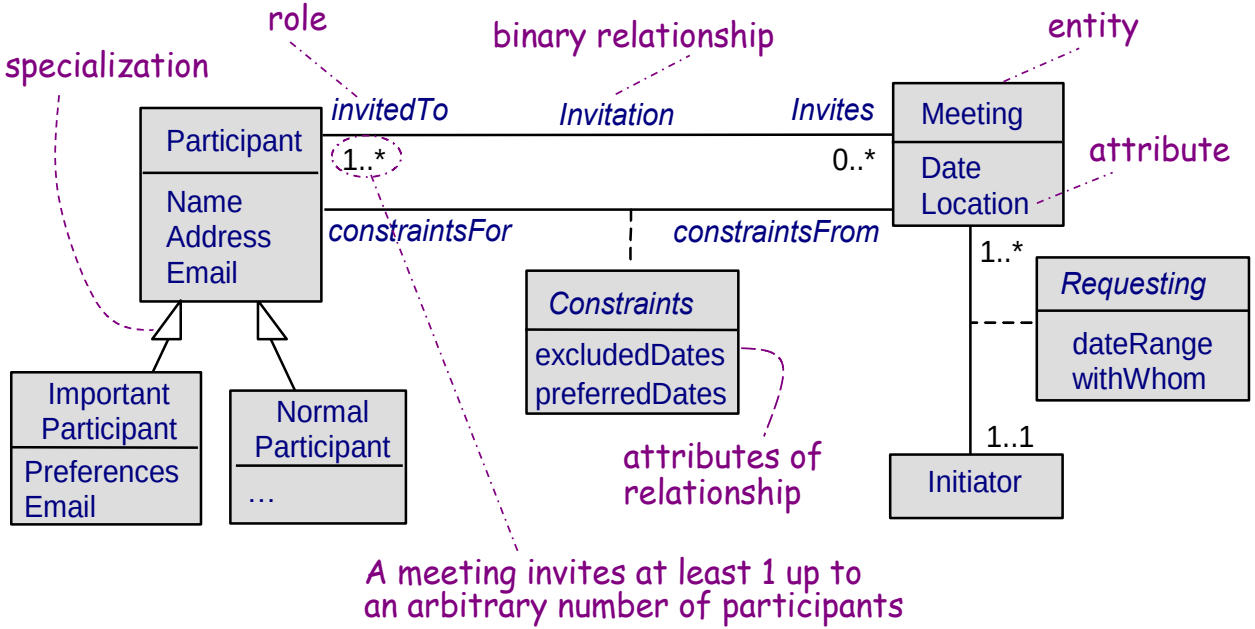
Outline

- 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
  - Integrating multiple system views, multi-view spec in UML

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 ($N \geq 2$)
  - **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

- 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

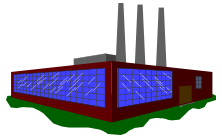
  e.g. annotation for Participant:

  "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"

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
  - Integrating multiple system views, multi-view spec in UML

Activities and data: SADT diagrams

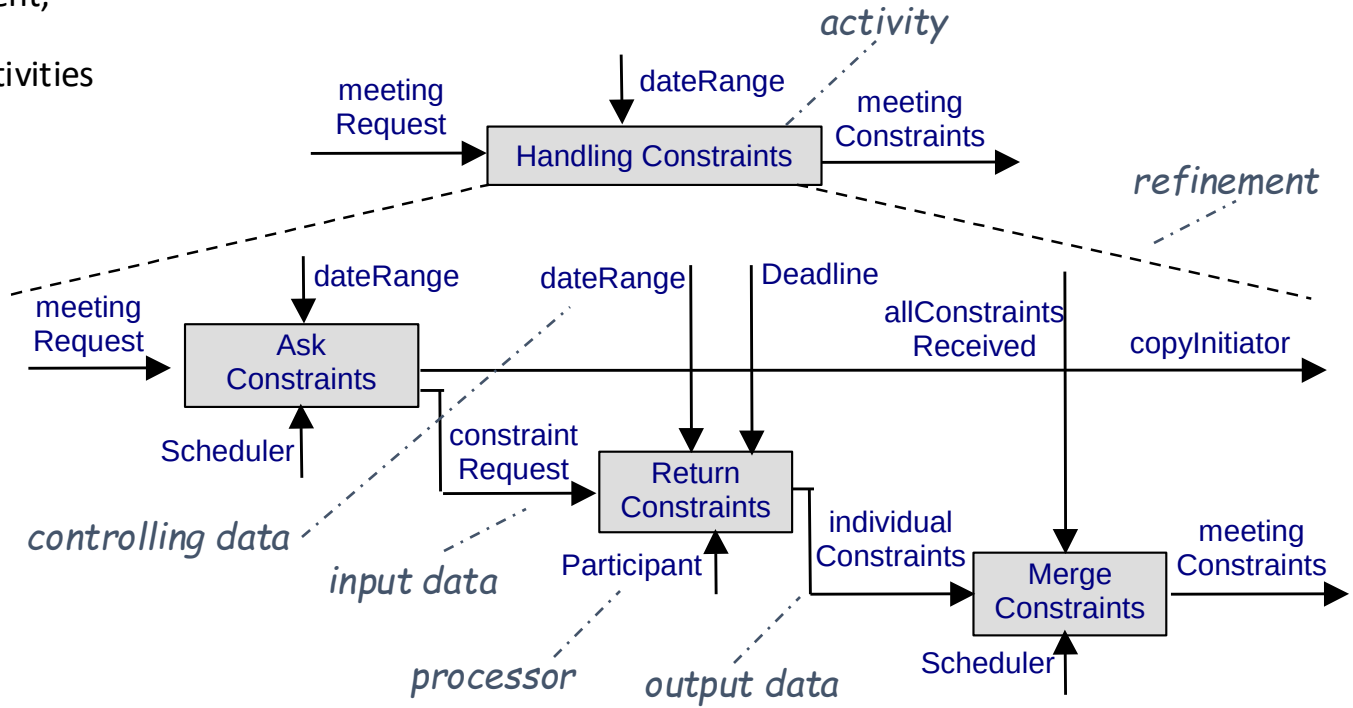


- SADT: Structured Analytics Design Technique
- Capture activities & data in the system (as-is or to-be)

- **Actigram**: relates activities through *data* dependency links (input/output)
- **Datagram**: relates data through *control* dependency links (producers/consumers)
  - 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

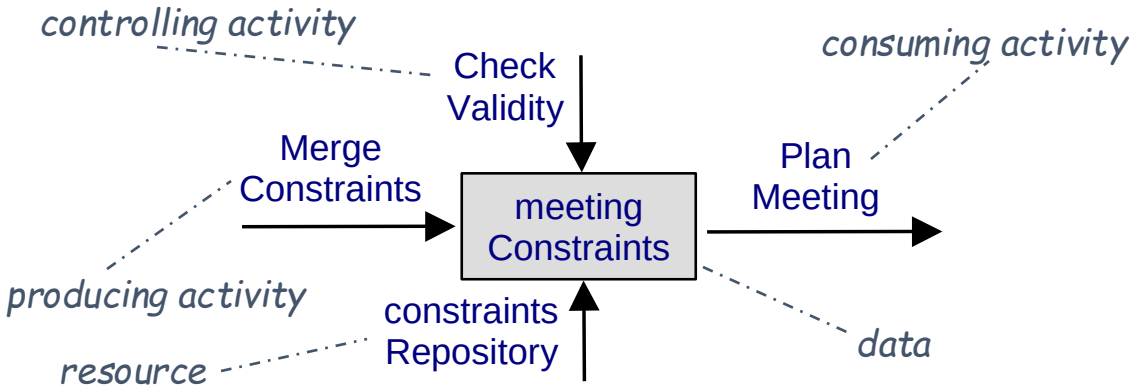
SADT diagrams: actigram example

East → : input data;
West → : output data
North → : controlling data/event;
South → : processor
Activities refinable into sub-activities



SADT diagrams: datagram example

East → : producing activity;
West → : consuming activity
North → : validation activity;
South → : needed resources
Data refinable into sub-data

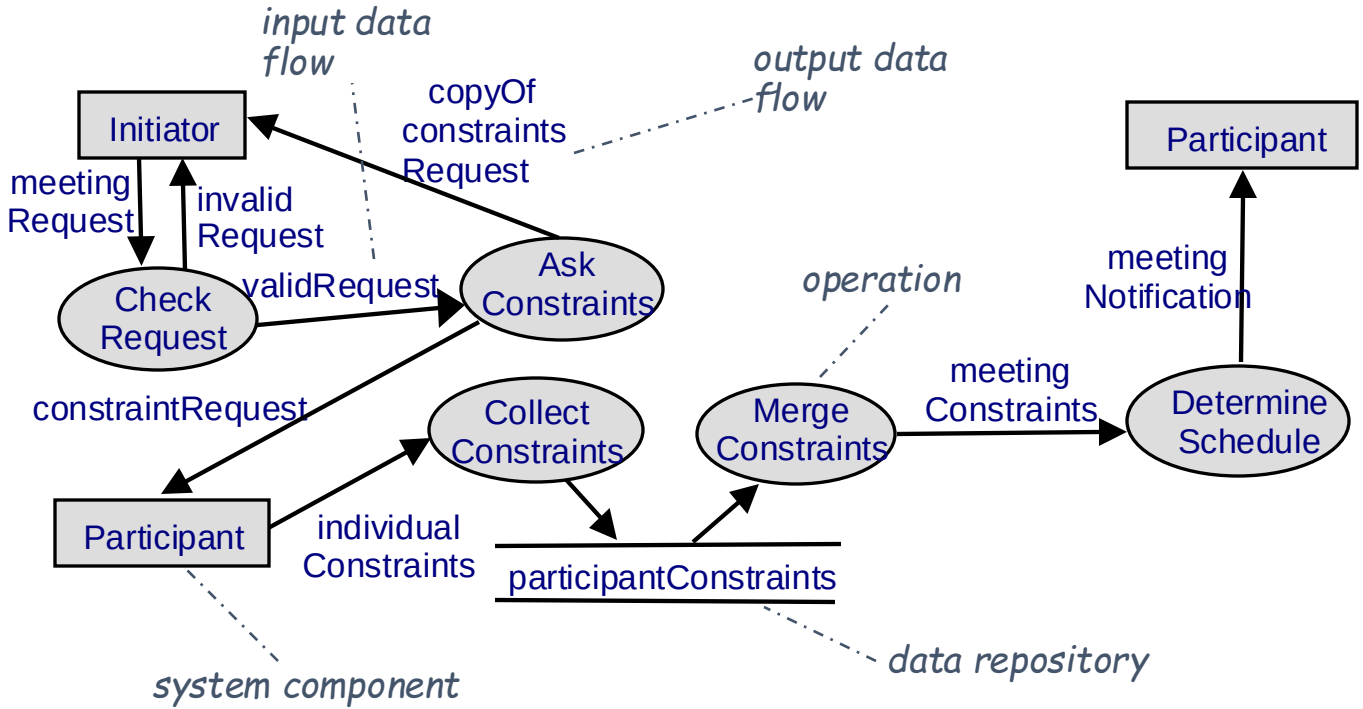


- Consistency/completeness rules checkable by tools
  - Every activity must have an input and an output
  - All data must have a producer and a consumer
  - I/O data of an activity must appear as I/O data of subactivities
  - Every activity in a datagram must be defined in an actigram, ...

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

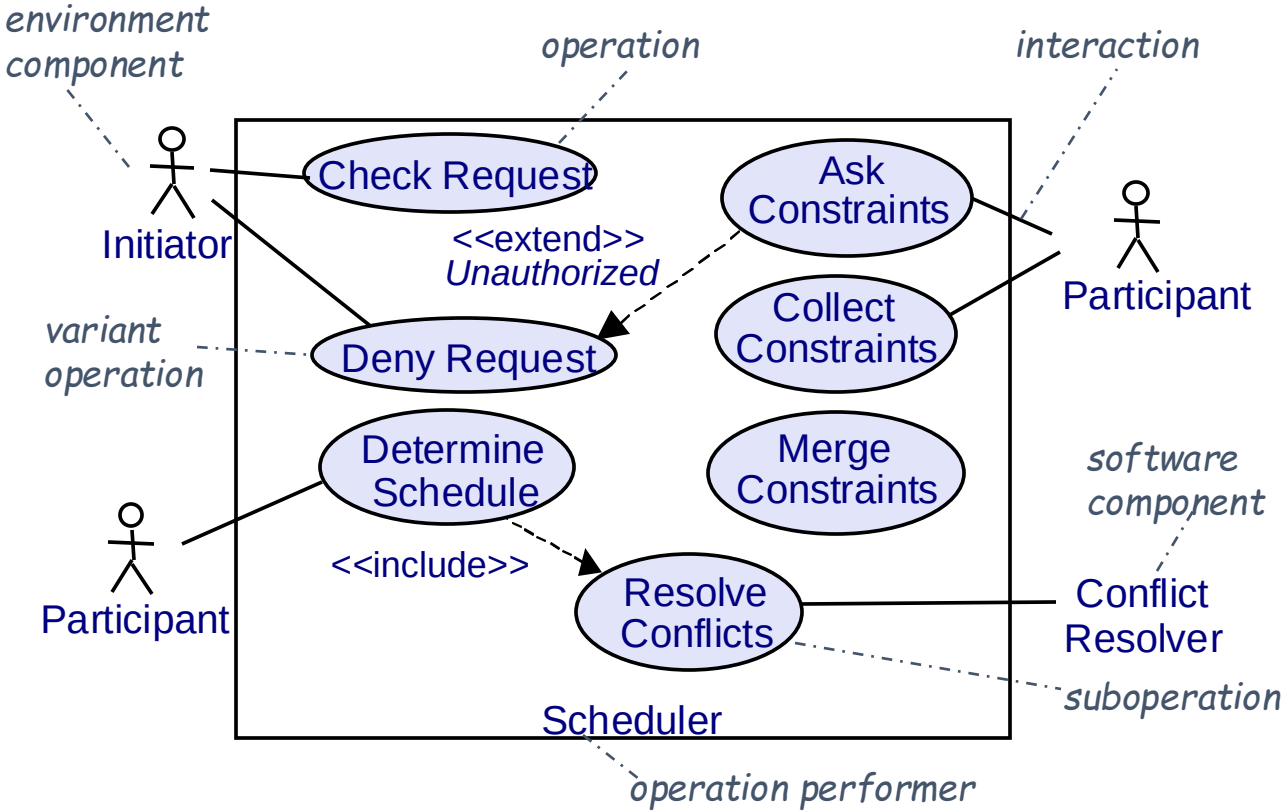
Dataflow diagram: example



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

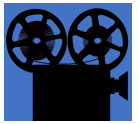
Use case diagram: example



Outline

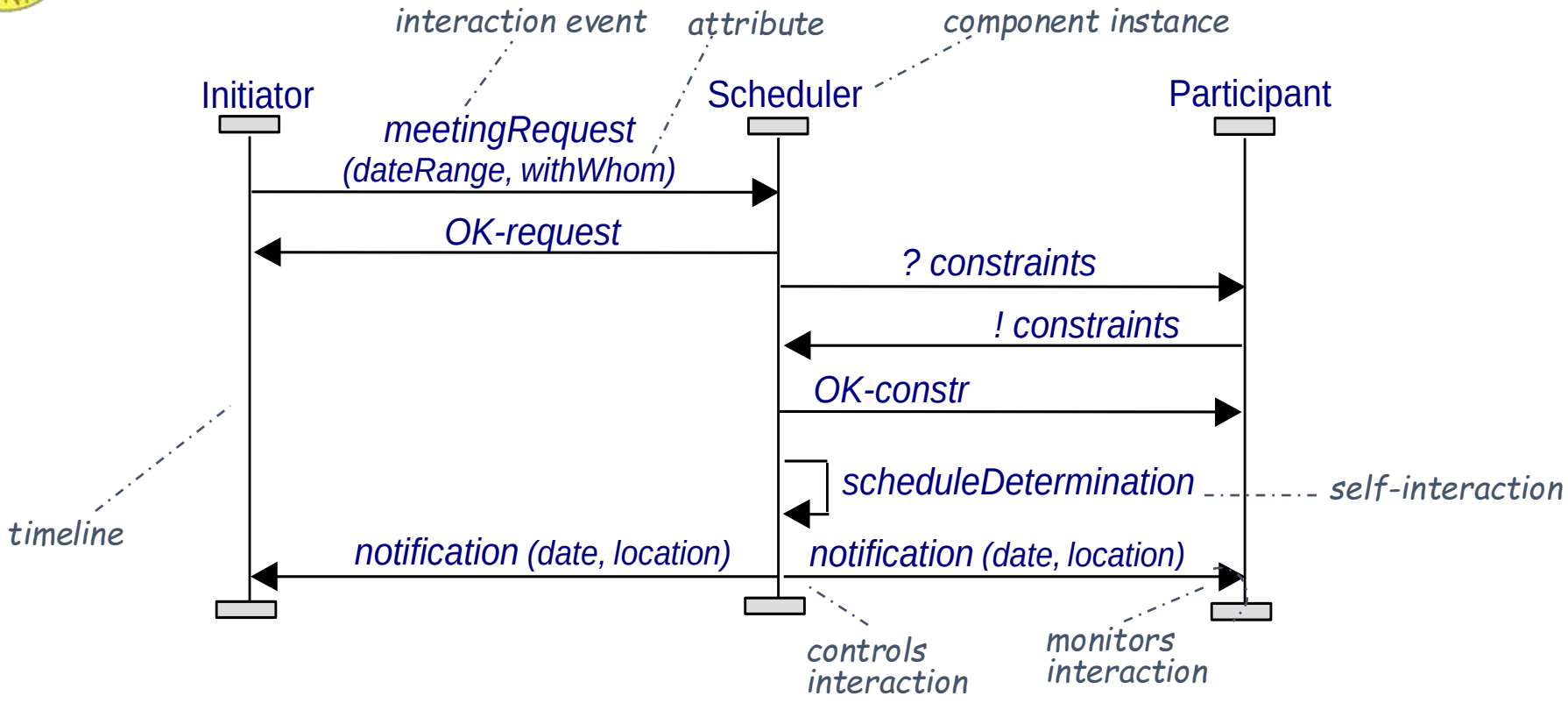
- Free documentation in unrestricted natural language
- Disciplined documentation in structured natural language
  - Local rules on writing statements
  - Global rules on organizing the Requirements Document
- Use of diagrammatic notations
  - System scope: context, problem, frame diagrams
  - Conceptual 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**
  - Integrating multiple system views, multi-view spec in UML

Interaction scenarios: event trace diagrams

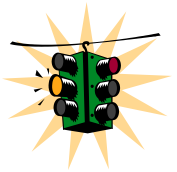


- Capture positive scenarios by sequences of interactions among instances of system components
  - variants: MSC (ITU), sequence diagrams (UML)
- Parallel composition of timelines
  - one per component instance
- Pairwise directed interactions down timelines
  - information transmission through event 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



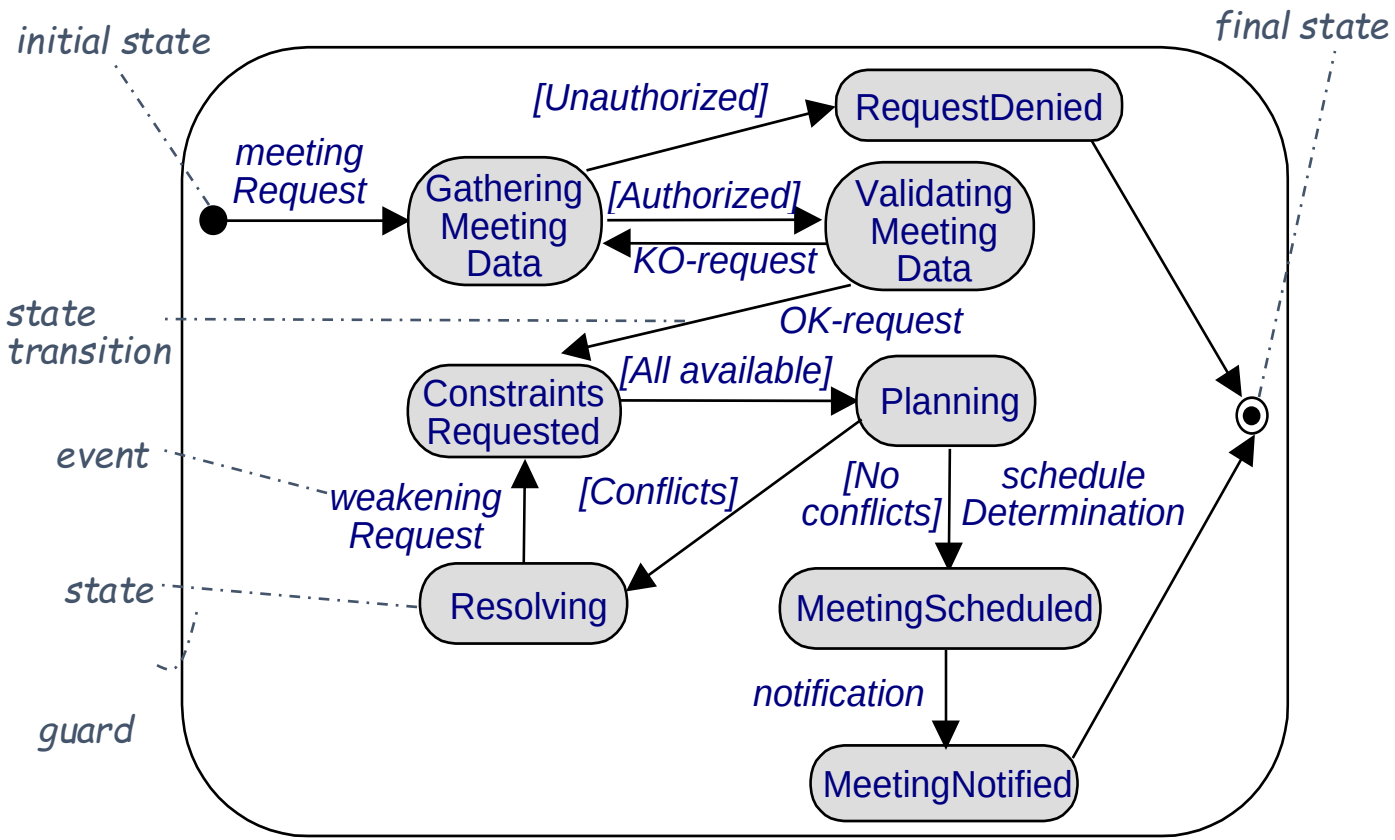
System behaviors: state machine diagrams



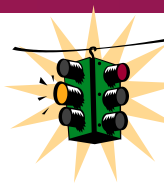
- Capture the admissible behaviors of system components
- **Behavior** of component instance =
  sequence of state transitions for the items it controls
- SM **state** = set of situations where a variable characterizing a controlled item has always the same value
  - e.g. state MeetingScheduled: always 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

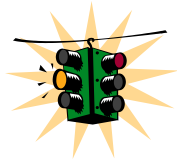


- Event occurrence is a *sufficient* condition for transition firing
  - Event can be external stimulus (e.g. meetingRequest) or application of internal operation (e.g. determineSchedule)
- Guard = *necessary* condition for transition firing
  - Item gets to *target* state …
    - **if** item is in *source*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



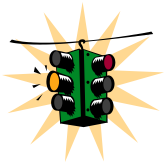
- SM **trace** = sequence of successive SM states up to some point
  - e.g. < GatheringMeetingData, RequestDenied >
  - always finite, but SM diagram may have infinitely many traces

- 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 >

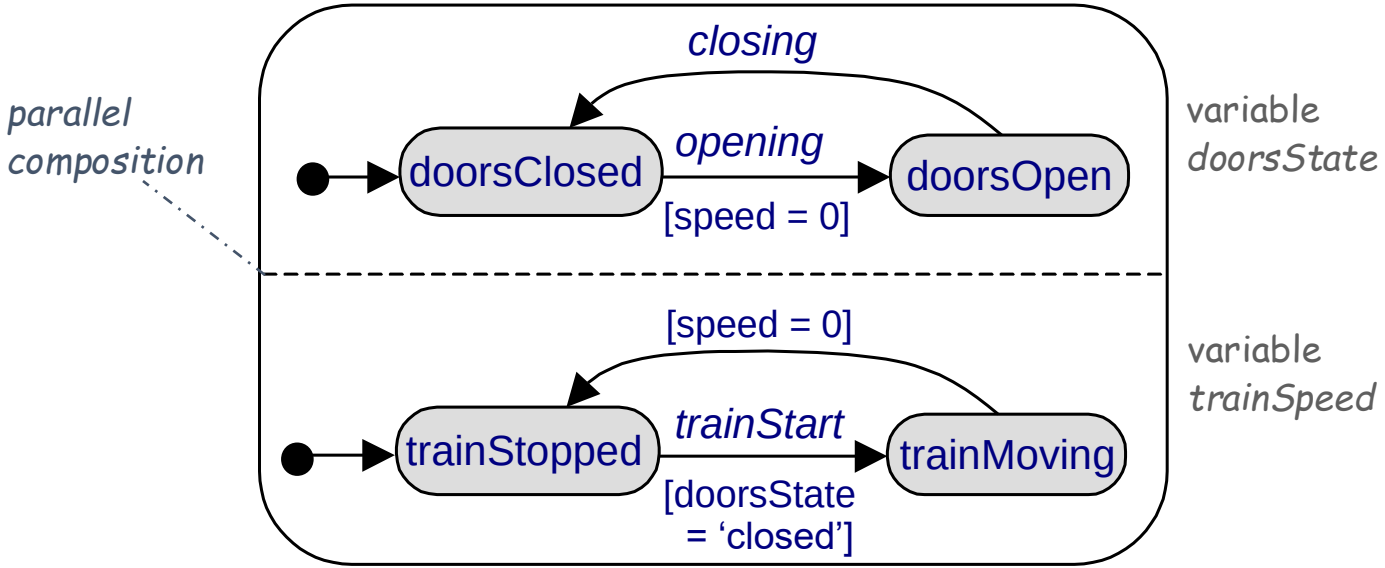
Concurrent behaviors and statecharts



- 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
  - 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



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

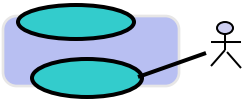
    "every used variable must be declared"
    "every declared variable must be used", …
  - can be used for inspection checklists
  - enforceable by tools
  - constrain diagram evolution

Multi-view specification in UML

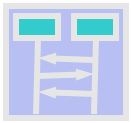
The Unified Modeling Language (UML) has standardized notations for diagrams relevant to RE

- **Class diagrams**: ER diagrams for structural view

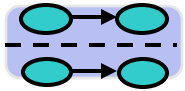
- **Use case diagrams**: outline of operational view



- **Sequence diagrams**: ET diagrams for scenarios



- **State diagrams**: SM diagrams for behavioral view



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*

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
  - Integrating multiple views, multi-view spec in UML