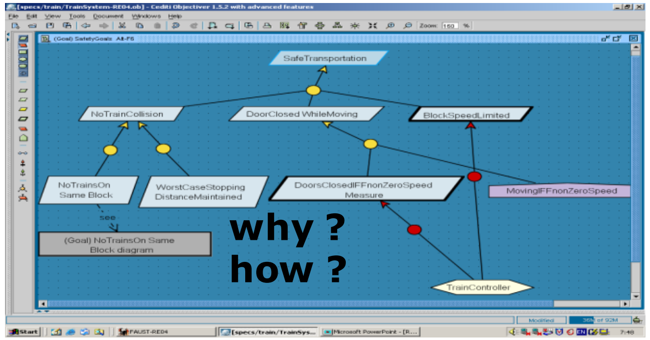
Modeling Conceptual Objects with Class Diagrams

Mariano Ceccato

mariano.ceccato@univr.it

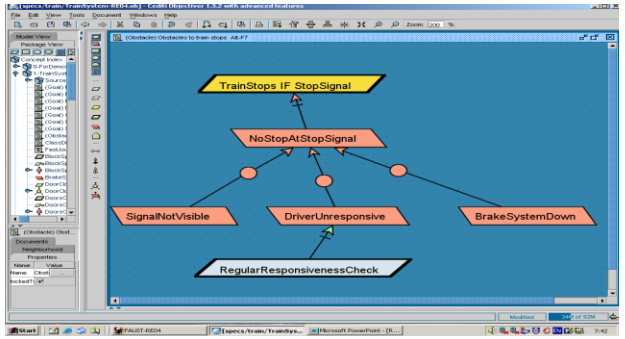
Building models for RE

Goals

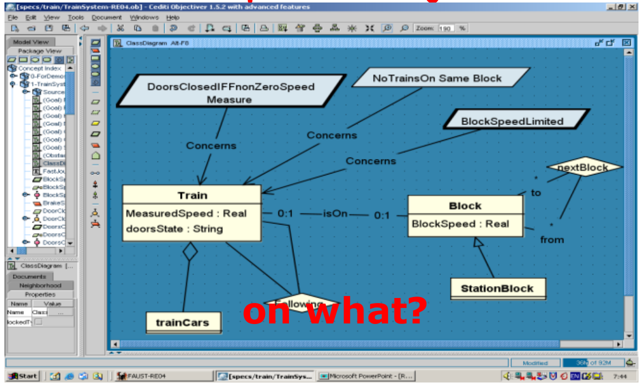


why ?
how ?

Risks

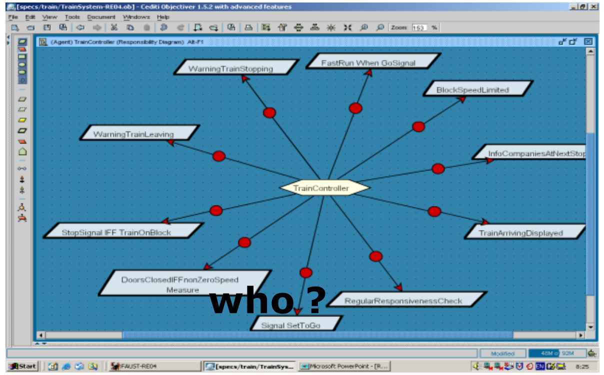


Conceptual objects



on what?

Agents



who ?

The object model

- **Structural** view of the system (-as-is or -to-be) being modeled
- Roughly, shows how relevant system concepts are structured and interrelated
- Represented by UML class diagram ...
  - “objects”, classes **not** in the OO design sense: RE is concerned with the problem world only!
  - classes with no operations: data encapsulation is a design concern; no design decisions here!
- Multiple uses ...
  - precise definition of system concepts involved in other views, their structure & **descriptive** properties
  - state variables manipulated in other views
  - common vocabulary
  - basis for generating a glossary of terms

Terminology

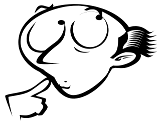
- *Object* -> class level abstraction ~~*[class object]*~~
  - e.g., Train

- *Object instance* -> their instances ~~*[object]*~~
  - e.g., train3432

Modeling conceptual objects: outline

- What is a conceptual object?
- Entities
- Associations & multiplicities
- Attributes
- Specialization
- Aggregation
- More on class diagrams
  - derived attributes, OR-associations, associations of associations
- Building object models: heuristic rules

What is a conceptual object?



- **Set of instances** of a system-specific **concept:**
  - **distinctly identifiable**
    - immutable, built-in identity
    - e.g. 2 string instances “Justine Henin” are the same,
      but 2 Patron instances named Justine Henin are different
  - **can be enumerated in any system state**
    - in any state we can list all instances of the Patron concept currently involved in the system
  - **share similar features**
    - common **name**, **definition**, **type**, **domain properties**,
    - common **attributes**, **associations**: see details later
    - e.g. Email attrib of Patron; Loan assoc linking Patron and BookCopy
  - may differ in their **individual states** and **state transitions**

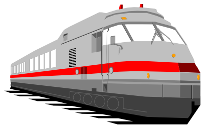
What is a state of an instance of conceptual object ?

- Tuple of functional pairs $\mathbf{x_i} \mapsto \mathbf{v_i}$

  $\mathbf{x_i}$: object attribute, association

  $\mathbf{v_i}$: corresponding value for that instance

- E.g. instance *tr* of Train object might be in state:
  - (*tr.Speed* $\mapsto$ *0*, *tr.Location* $\mapsto$ *9.25*, *tr.DoorsState* $\mapsto$ *Open*, *On* $\mapsto$ (*tr*, *block13*), *At* $\mapsto$ (*tr*, *platform1*))
  - different from state of Train instance *tr'*.



Object instantiation: classes & current instances

- Every conceptual object has a **built-in semantic relation** telling which instances are currently members of the object:

  InstanceOf (o, Ob) **iff** *o* is *currently* an instance of *Ob*

  - kept implicit in the object model, used for **Def** specification
  - "current" state = some arbitrarily chosen system state
  - e.g. *InstanceOf (bc, BookCopy)* says *bc* is currently member of set *BookCopy* of book copies manipulated in the library system
  - might not be the case 3 weeks earlier or 1 year later ...
- A set of object instances may evolve over time
- An instance may migrate from one object to another
  - e.g. StudentPatron instance $\rightarrow$ StaffPatron instance
- An instance may be member of multiple objects

Object instantiation: classes & current instances

- Every concept in object model must be defined by **Def annotation** specifying the necessary & sufficient condition for an individual to satisfy **InstanceOf (o, Ob)**
  - i.e. specific conditions for individual to appear & disappear as instance of this object
  - e.g. " A patron is any person who has registered to the corresponding library for the corresponding period of time and has not been excluded since then"
- When an individual becomes instance of an object, the object's attributes & associations get instantiated as **state variables** to characterize it
  - e.g. InstanceOf (tr, Train) $\rightarrow$ tr.Speed, tr.DoorsState, On (tr, ...)
- State variables of the system = set of state variables of all conceptual objects declared in the object model

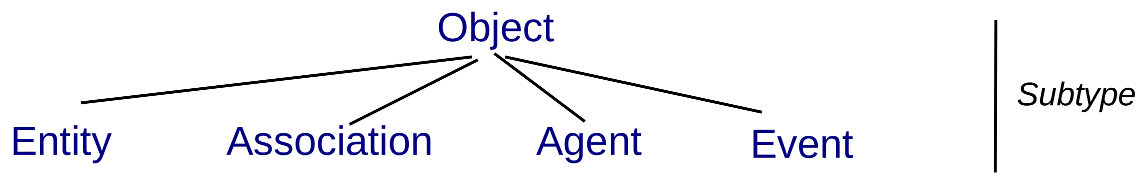
Types of conceptual object

- **Entity**: autonomous, passive object
  - instances may exist in system independently of instances of other objects
  - instances cannot control behavior of other objects
  - e.g. Book, BookCopy ; Train, Platform, ...
  - represented as UML class
- **Association**: object dependent on objects it links
  - instances are conceptual links among object instances
  - e.g. Loan linking Patron & BookCopy
    Copy linking BookCopy & Book
    At linking Train & Platform
    On linking Train & Block
  - represented as UML association

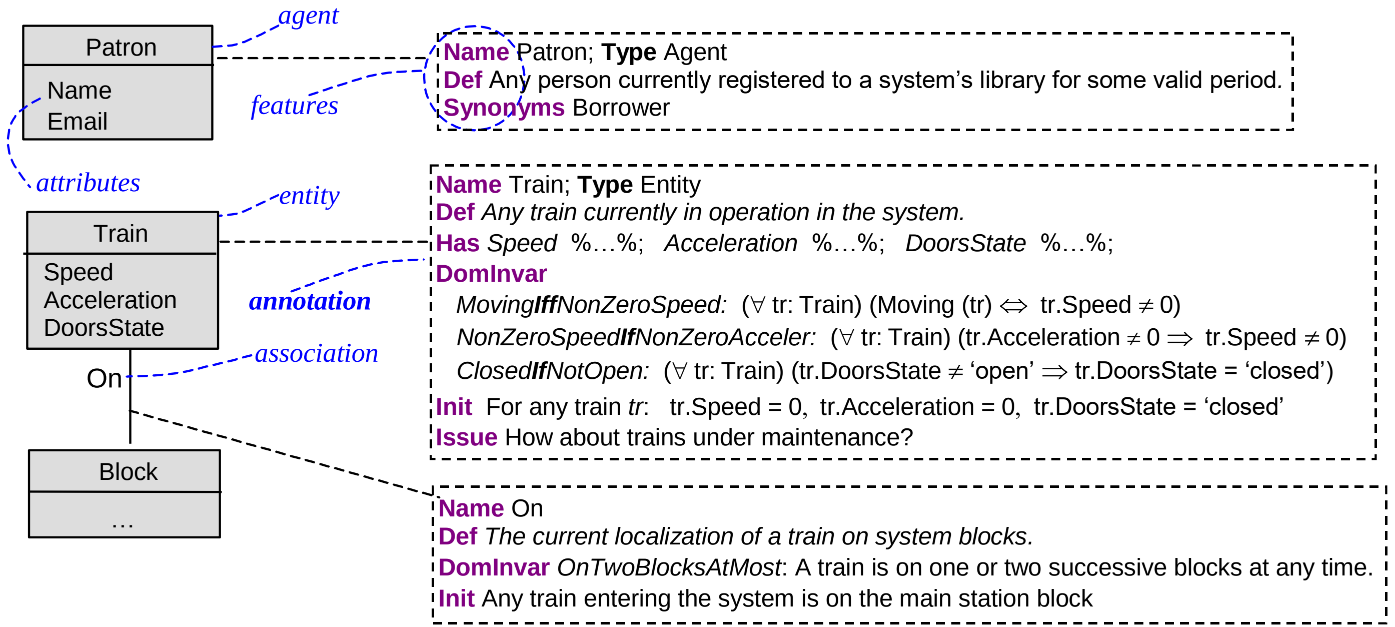
Types of conceptual object

- **Event**: instantaneous object
  - instances exist in single system state

    InstanceOf (ev, Ev) denoted by *Occurs* (Ev)
  - e.g. BookRequest ; StartTrain
  - represented as <u>UML class</u> if attributes, associations needed
- **Agent**: active, autonomous object
  - instances have individual behavior = sequence of state transitions for state variables they control
  - e.g. Patron, Staff ; TrainController, TrainDriver
  - represented as <u>UML class</u> if attributes, associations needed



Object features as model annotations



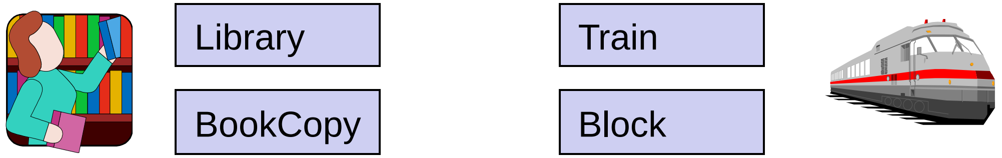
Name Patron; **Type** Agent
Def Any person currently registered to a system's library for some valid period.
Synonyms Borrower

Name Train; **Type** Entity
Def *Any train currently in operation in the system.*
Has *Speed* %…%; *Acceleration* %…%; *DoorsState* %…%;
DomInvar
*Moving**Iff**NonZeroSpeed:* $(\forall$ tr: Train) (Moving (tr) $\Leftrightarrow$ tr.Speed $\neq 0)$
*NonZeroSpeed**If**NonZeroAcceler:* $(\forall$ tr: Train) (tr.Acceleration $\neq 0 \Rightarrow$ tr.Speed $\neq 0)$
*Closed**If**NotOpen:* $(\forall$ tr: Train) (tr.DoorsState $\neq$ 'open' $\Rightarrow$ tr.DoorsState = 'closed')
Init For any train *tr*: tr.Speed = 0, tr.Acceleration = 0, tr.DoorsState = 'closed'
Issue How about trains under maintenance?

Name On
Def *The current localization of a train on system blocks.*
DomInvar *OnTwoBlocksAtMost*: A train is on one or two successive blocks at any time.
Init Any train entering the system is on the main station block

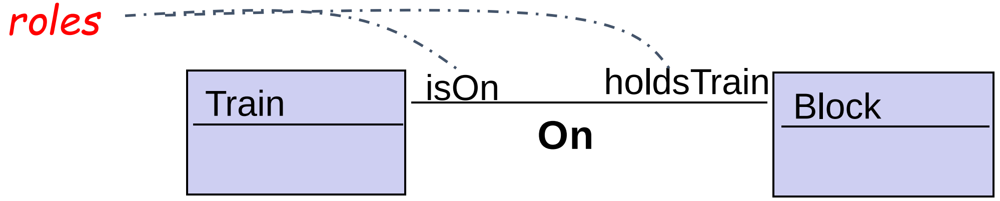
Entities

- As seen before: autonomous, passive object
  - instances may exist in system independently of instances of other objects
  - distinctly identifiable, can be enumerated in any system state, share similar features, may differ in individual states & transitions
  - instances can<u>not</u> control behavior of other objects
  - characterized by **Def**, domain invariants, attributes, initialization
- In the **Def** annotation, the conditions for an individual to appear & disappear as instance of this entity must not necessarily refer to other objects in the model



Associations

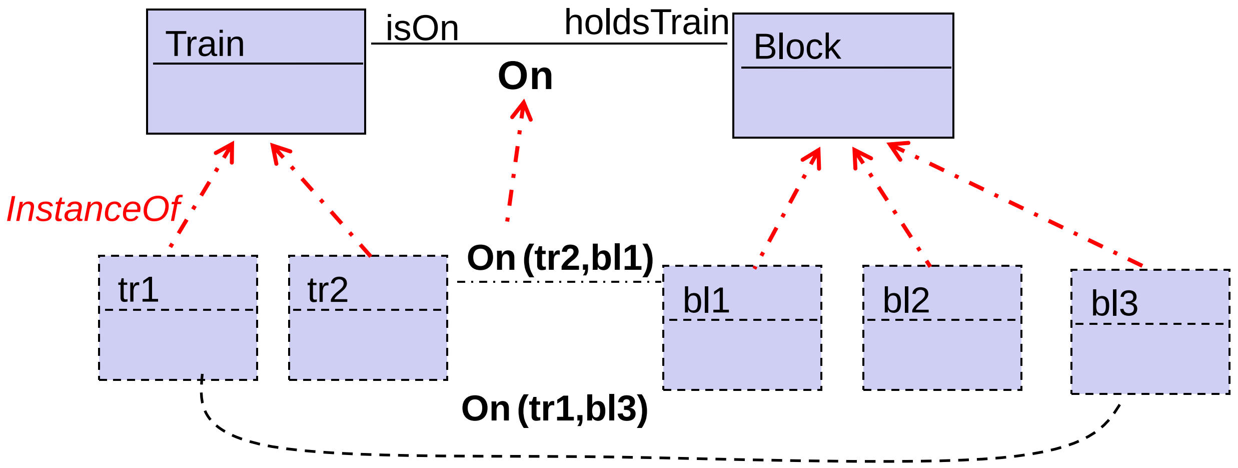
- **Association** = conceptual object linking other objects, each playing specific role
  - dependent on objects it links
  - linked objects may be entities, associations, events, agents



- **Association instance** = tuple of linked object instances, each playing corresponding role
  - may currently exist only if all instances are currently linked and currently instances of corresponding objects
- Predicate notation
  - *Assoc* $(o_1, \ldots, o_n)$ for *InstanceOf* $([o_1, \ldots, o_n],$ *Assoc*$)$

Association instances

- **Association instance** = tuple of linked object instances, each playing corresponding role



Associations & their instances

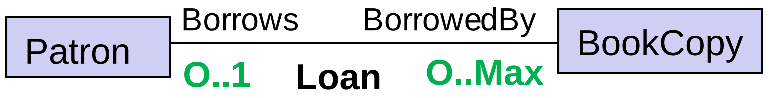
- Like for any object, association instances ...
  - are distinctly identifiable
    - built-in immutable identity = tuple of identities of linked object instances
  - can be enumerated in any system state
  - are characterized by common features
    - name, definition, attributes, domain invariants, initializations
  - evolve individually from state to state ...
    - instance **appears** as Ass (o_1, ..., o_n) gets *true* (link creation)
    - instance **disappears** as Ass (o_1, ..., o_n) gets *false* (link deletion)
    - **state change** as values of attached *attrib, assoc* are changing

Associations

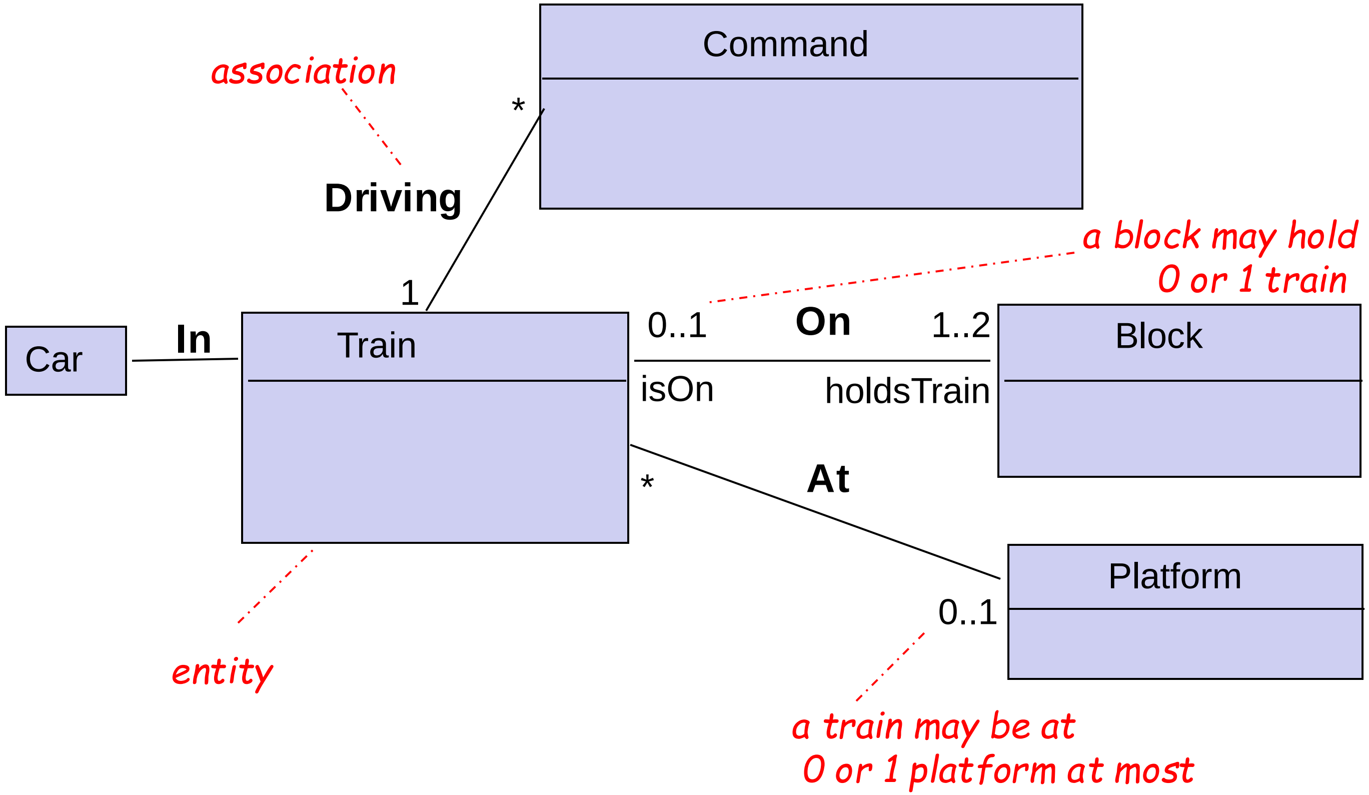
- Arity of association = number of objects linked by it
  - **Binary** associations: arity = 2
  - **N-ary** associations: arity > 2
    - needed when links involving more than 2 objects must be distinguished
    - e.g. Registration **linking:**
    - Patron (role MemberOf)
      Library (role hasMember)
      Period (role ValidityPeriod)
    - if binary, no distinction possible among registrations of same patron & library for different periods
- **Reflexive** association = same object appears under different roles
  - e.g. Following linking Train (role Follows), Train (role FollowedBy)

Multiplicities of n-ary association

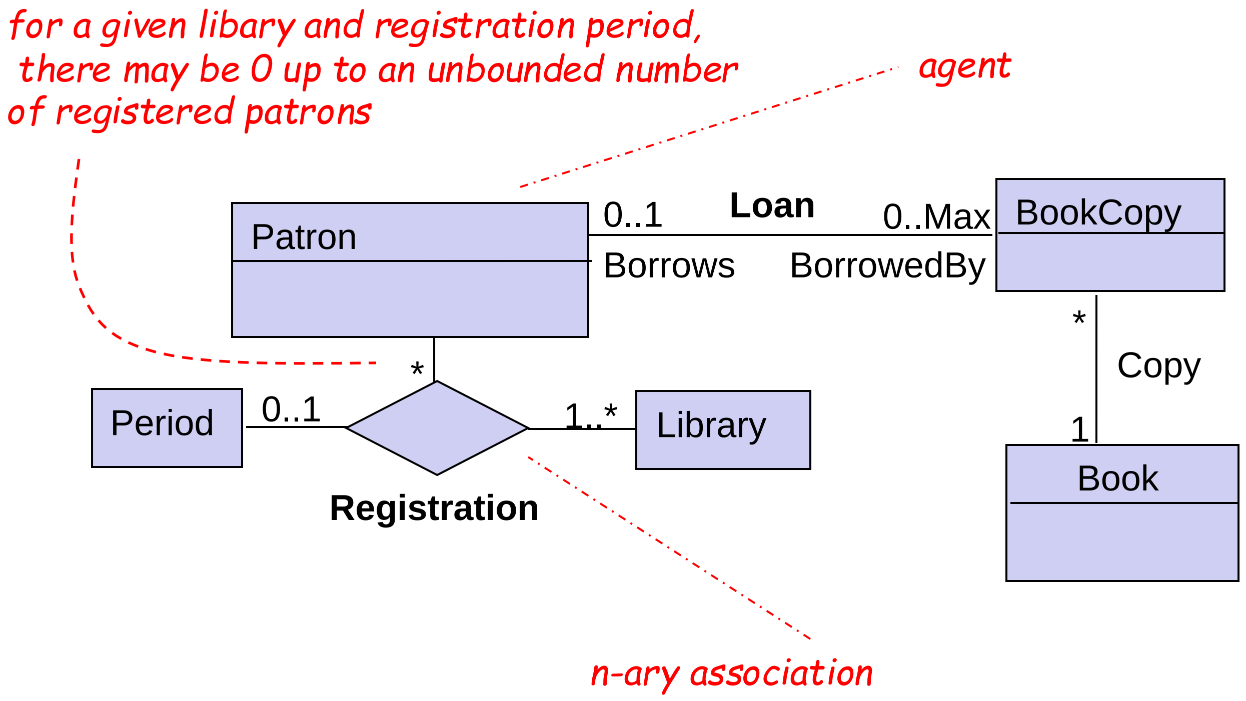
- From fixed **source**(n-1)-tuple of currently linked instances:
  - **min/max** number of linked **target** instances
  - attached to role of **target** instance
- For binary associations, express standard constraints ...
  - min = 0: optional link (possibly no link in some states)
  - min = 1:   mandatory link (at least one link to target in any state)
  - max = 1: uniqueness (at most one link to target in any state)
  - max = *: arbitrary number N of target instances linked to
    source instance, in any state (N > 0)
  - Notation: “k” for “k..k”,   “*” for “0..*”



Entities, associations in UML



Entities, associations in UML



Multiplicities, domain properties and goals

- Multiplicities may encode some
  - domain properties (descriptive)
    "A train may be at one platform at most at a time"
  - goals (prescriptive)
    "A block may not accommodate more than one train at any time"
    "A patron may not borrow more than Max book copies at a time"
    - to be found in the goal model as well!

    ⇒ source for goal elicitation (parent goals?, subgoals?)
- BUT ...
  - multiplicities mix prescriptive & descriptive assertions
  - most assertions are not expressible by multiplicities
    "A borrowed book must be returned within 2 weeks"
    "A copy may not be both borrowed and available"

  ⇒ need for other domain invariants, goals/requirements

Attributes

- Intrinsic feature shared by any instance of an object
  - entity, association, event, agent (like associations)
- An attribute *Att* of object *Ob* is a function:

$$Att: Ob \rightarrow SORT$$

- **Sort**: set of possible attribute values the attribute might take (function range)
  - NOT a conceptual object we want to model
  - may be declared by ...
    - predefined, domain-independent name

      CopyAvailable: Boolean   PatronName: String
    - domain-specific name

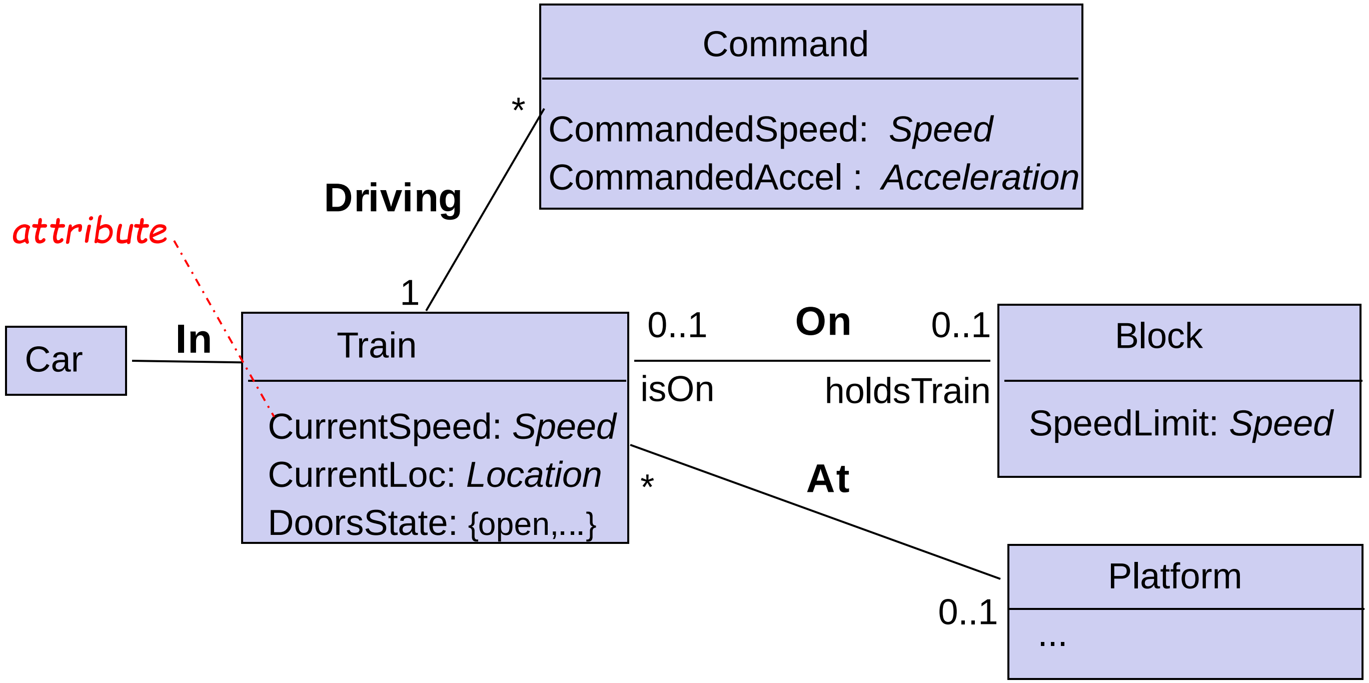
      BlockSpeedLimit: Speed   Keywords: Topics
    - enumeration

      GoSignal: {on, off}

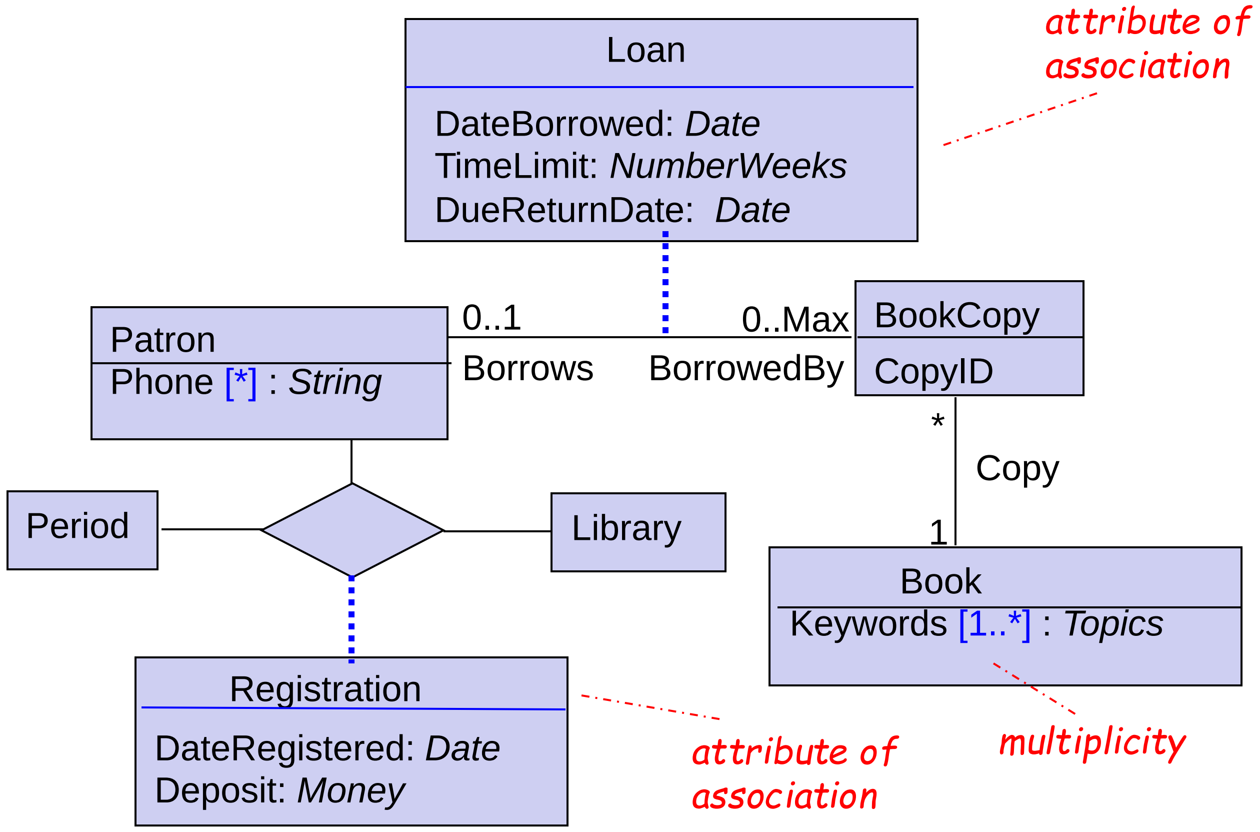
Attributes

- **Elementary** attribute: sort is a set of atomic values
  - e.g. DoorsState: {open, closed}
- **Structured** attribute: sort defined with type constructor
  - Tuple, SetOf, SequenceOf, Union
  - e.g. Keywords: SetOf [Topic] , dateRange: SeqOf [Date]
- Precise, domain-specific semantics of attribute must be defined in ***Has*** annotations
- Attribute **multiplicity**: min/max number of values the attribute may take
  - [0..x]: optional attribute
  - [x..*]: attribute value = set of values
  - [1..1]: mandatory attribute, single value: by default, omitted
  - e.g. PhoneNr [0..*]: String optional, possibly multiple values

Entities, associations, attributes in UML



Entities, agents, associations, attributes in UML



Modeling conceptual objects

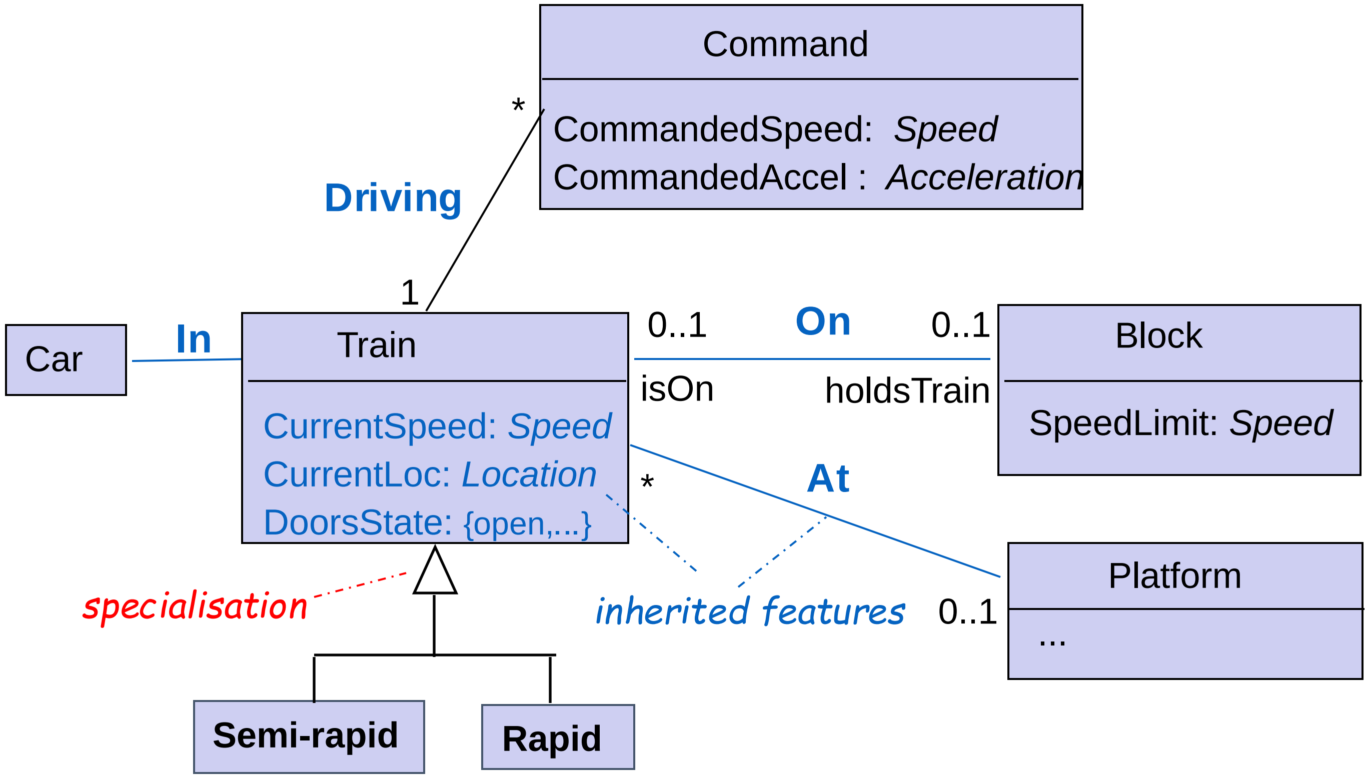
- What is a conceptual object?
- Entities
- Associations & multiplicities
- Attributes
- **Specialization**
- **Aggregation**
- More on class diagrams
- Building object models: heuristic rules

Built-in associations for structuring object models

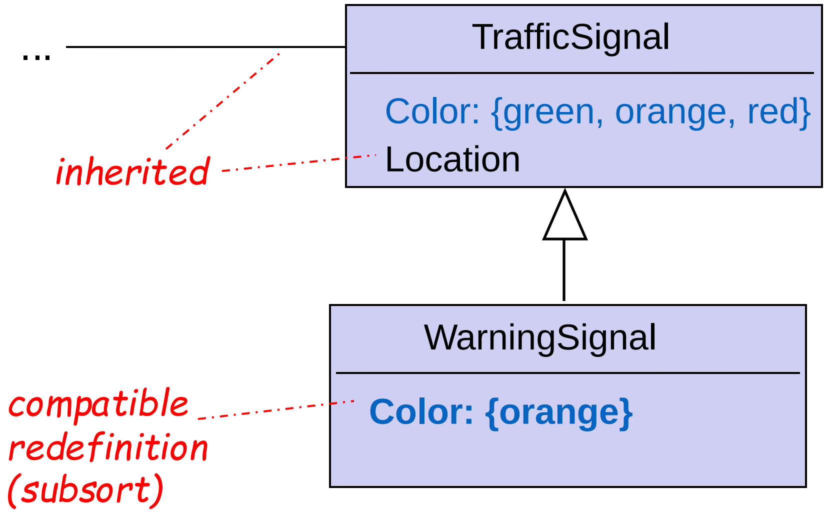
- Object specialization/generalization, decomposition/aggregation
  - applicable to entities, agents, events, associations
- **Specialization** = subclassing: object *SubOb* is a specialization of object *SuperOb* **iff** for any individual o:

  InstanceOf (o, SubOb) $\Rightarrow$ InstanceOf (o, SuperOb)
  - SubOb **specializes** SuperOb, SuperOb **generalizes** SubOb
  - amounts to set inclusion on set of current instances
- **Feature inheritance** as a consequence ...
  - by default, *SubOb* inherits from *SuperOb* all its attributes, associations, domain properties
    - while have its own distinguishing features
  - may be inhibited by compatible redefinition of feature with same name within specialized SubOb (“override”)

Object specialization with inheritance



Inhibiting inheritance

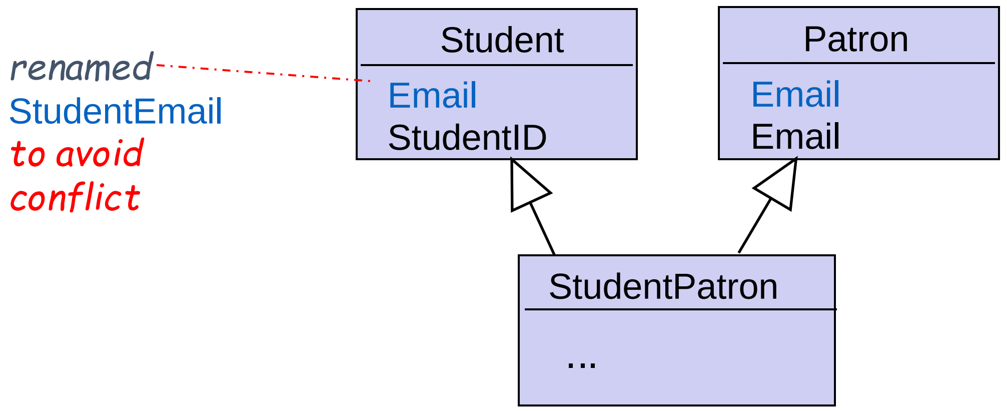


- The more specific feature always overrides the more general one

Multiple inheritance

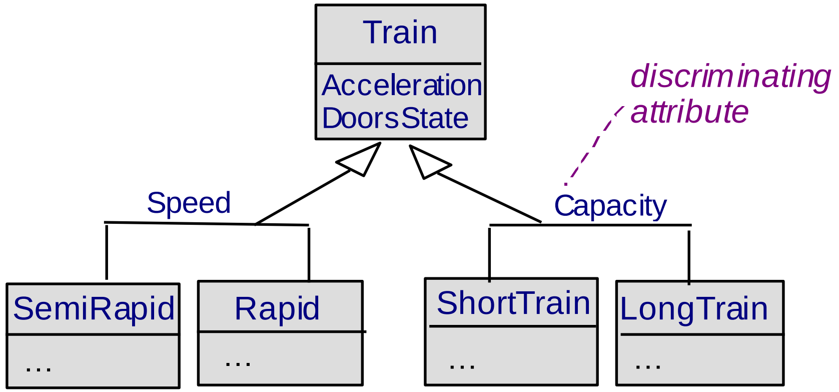
- Same object may be specialization of multiple super-objects
  - by default, inheritance of all features from all super-objects
- Can result in inheritance conflicts
  - different features with same name inherited from different super-objects

=> conflicting features first renamed to avoid this

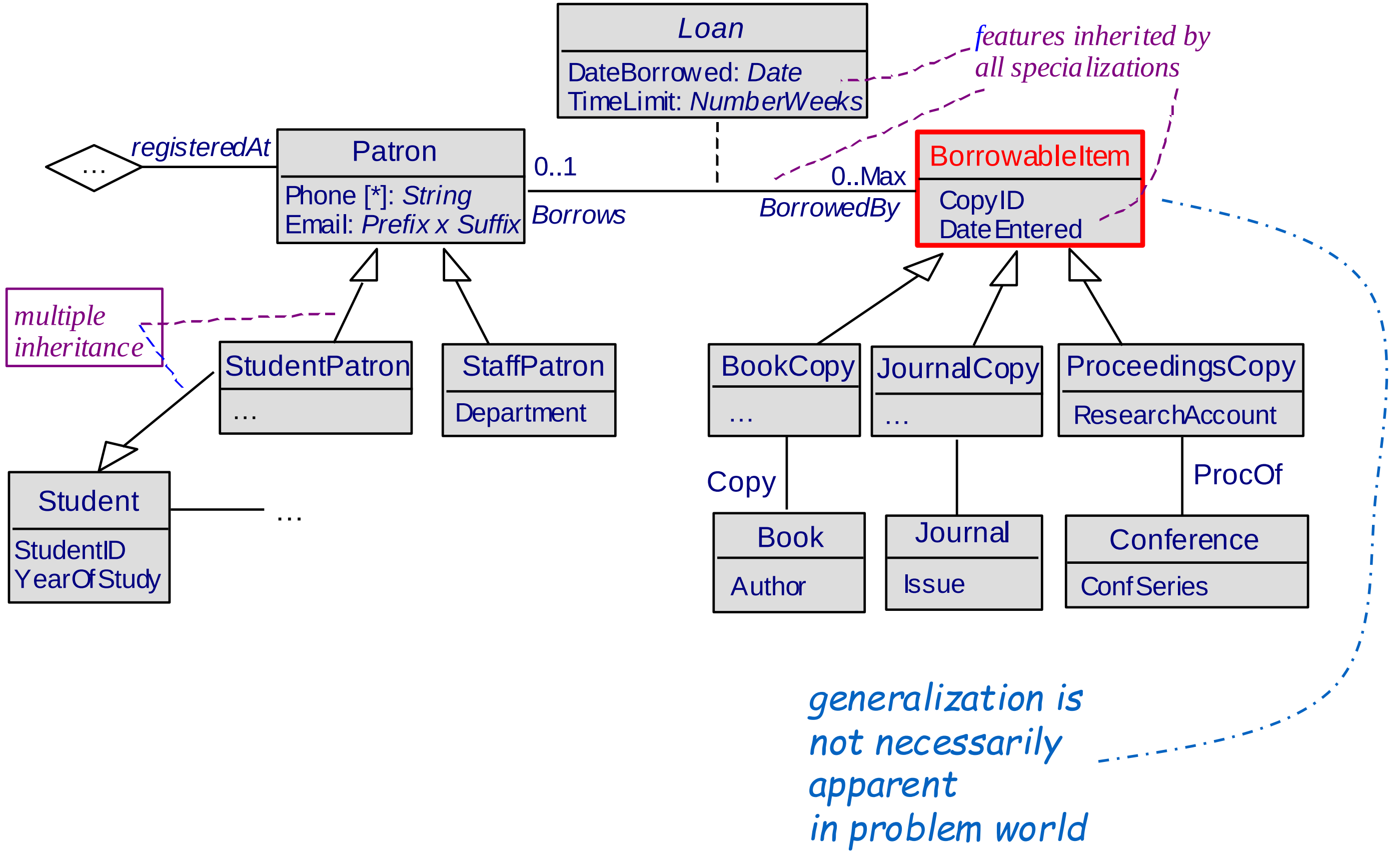


Multiple specializations

- Same object may have multiple specializations
  - Different subsets of object instances associated with different criteria
  - Same object instance may be member of different subsets (one per criterion)
- **Discriminator** = attribute of super-object whose values define different specializations (differentiation criterion)



Object generalization

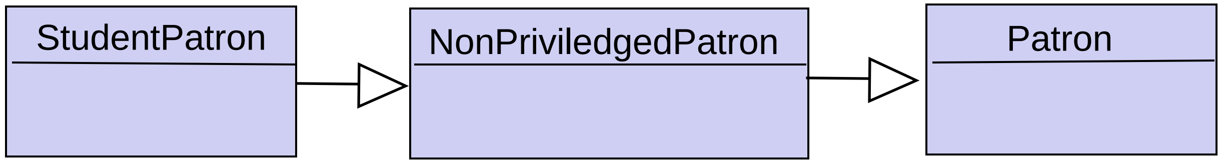


Benefits of generalization-based structuring

- Common features in multiple objects are factored out into single generalized object

  => simpler model, no duplication

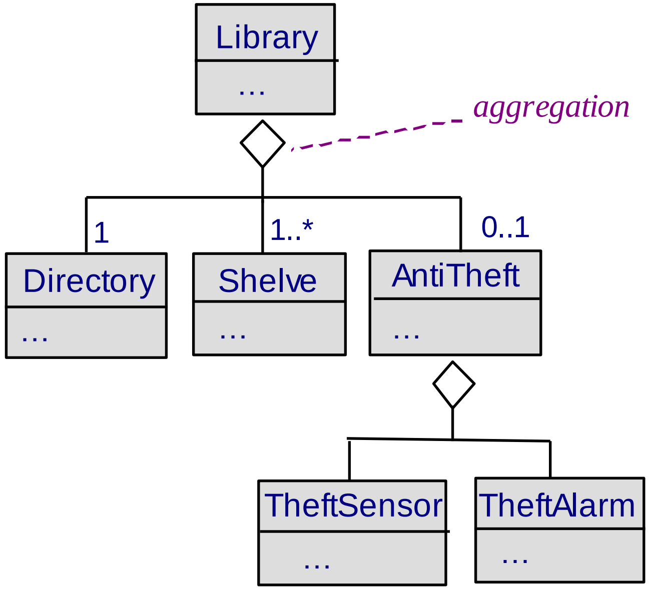
- Generalized objects & their structure are reusable in different contexts & systems (by specialization)
  - e.g. BorrowableItem --> CDCopy , VideoCopy
- Increased modifiability of large models
  - modifications of more general features are localized in more general objects, down-propagated to specialized objects

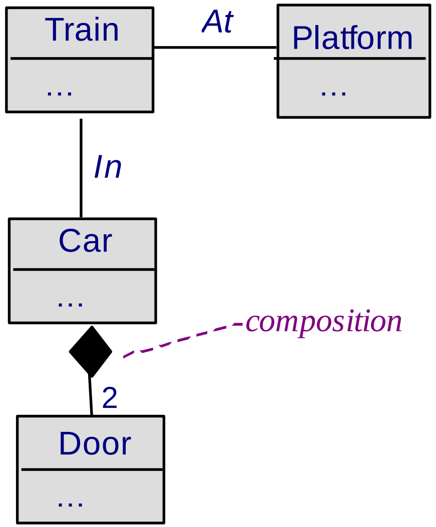


Object aggregation

- **Aggregation** = composite object whose components are objects ("parts")
  - applicable to entities, agents, events, associations
  - multiplicities may be attached to part-to-aggregation links
  - transitive, antisymmetrical links
- **Composition**: aggregation & parts <u>appear/disappear together</u>
  - part object may then be part of one aggregation object only

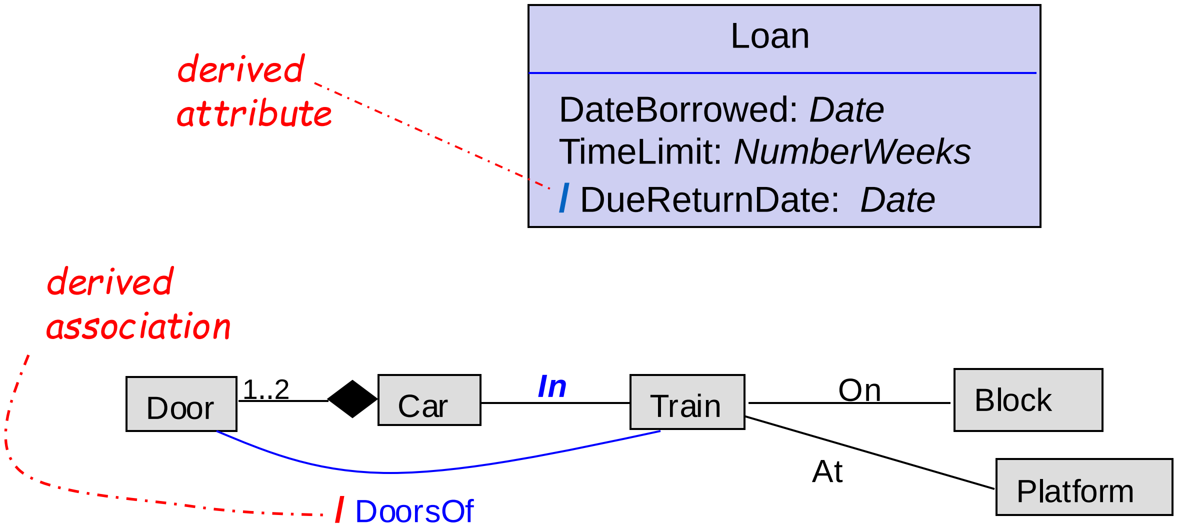
Object aggregation: examples





More on UML class diagrams

- **Derived** attribute, association = defined in terms of other attrib/assoc already in the model
  - controlled form of redundancy



Modeling conceptual objects

- What is a conceptual object?
- Entities
- Associations & multiplicities
- Attributes
- Specialization
- Aggregation
- More on class diagrams
- **Building object models: heuristic rules**

Building object models: heuristic rules

- Deriving **pertinent** & **complete** object models from goal models
  - deriving objects, associations, attributes
  - introducing software-environment tracking associations
  - identifying associations from domain invariants on multiple objects
- Object or attribute ?
- Entity, association, agent, or event ?
- Attribute of a linked object or of a linking association ?
- Aggregation or association ?
- Specializing, generalizing concepts
- Bad smells

Deriving objects, associations, attributes from the goal model

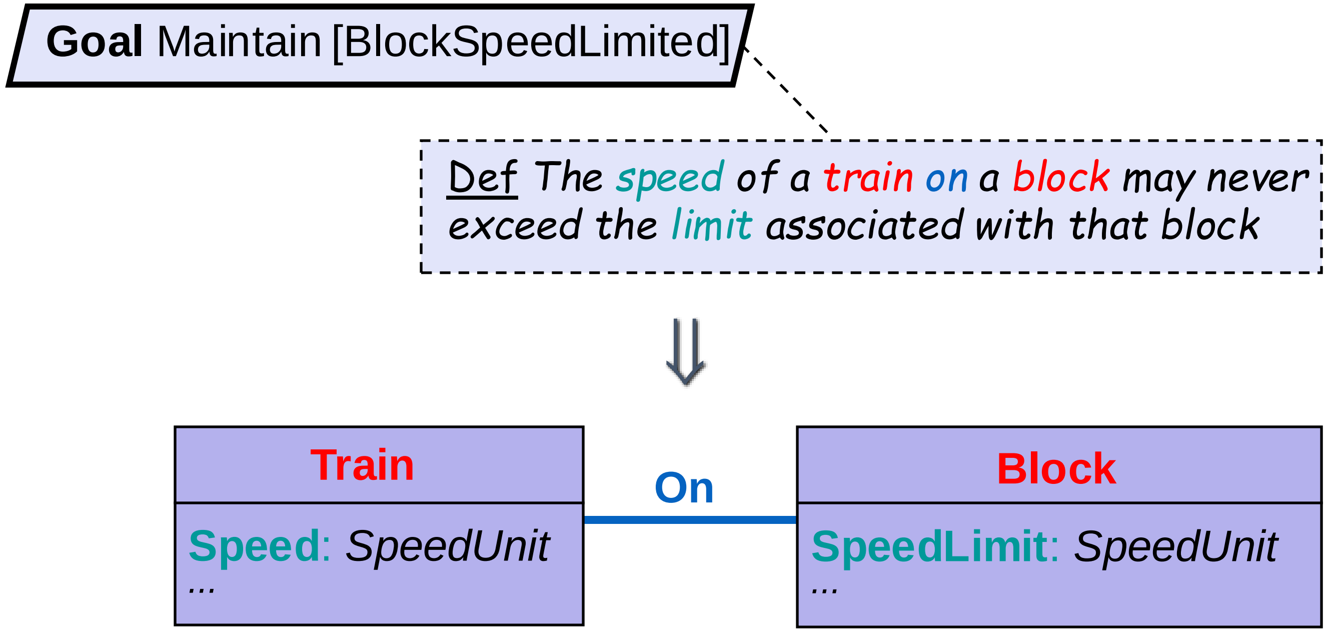
☞ Review all specs of goals & domain properties in goal model:

- take all **referenced** concepts meeting criteria for object, and only those
  - instances distinctly identifiable, enumerable in any state, sharing similar features, differing in individual states
- consider the others as candidate qualifying attributes
  - values are NOT concept instances to be characterized by common attributes, associations
- identify associations + participating objects from **linking expressions** in these specs

  <sourceObj> <**linkingVerb**> <targetObj(s)>

  <**linkingNoun**> of <targetObj> by <sourceObj>

Deriving objects, associations, attributes from goal specs: example



Goal Maintain [BlockSpeedLimited]

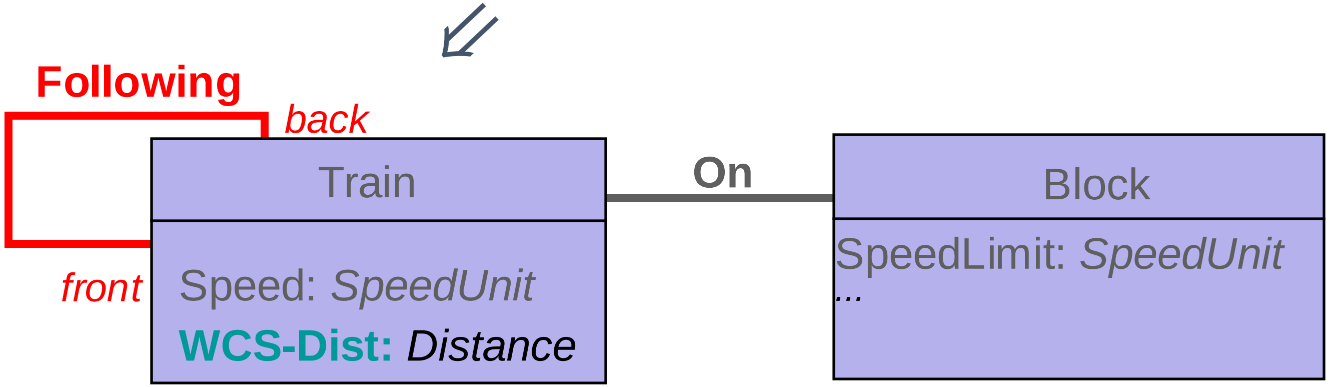
<u>Def</u> *The speed of a train on a block may never exceed the limit associated with that block*

- Rephrasing sometimes needed to highlight linking expressions
- Yet another reason for goal specs to be precise !

Deriving objects, associations, attributes from goal specs: example

Goal Maintain [WorstCaseStoppingDistance]

Def *The distance between two trains following each other shall be sufficient to prevent the back train from hitting the front train in case the latter stops suddenly*



Introducing software-environment tracking associations

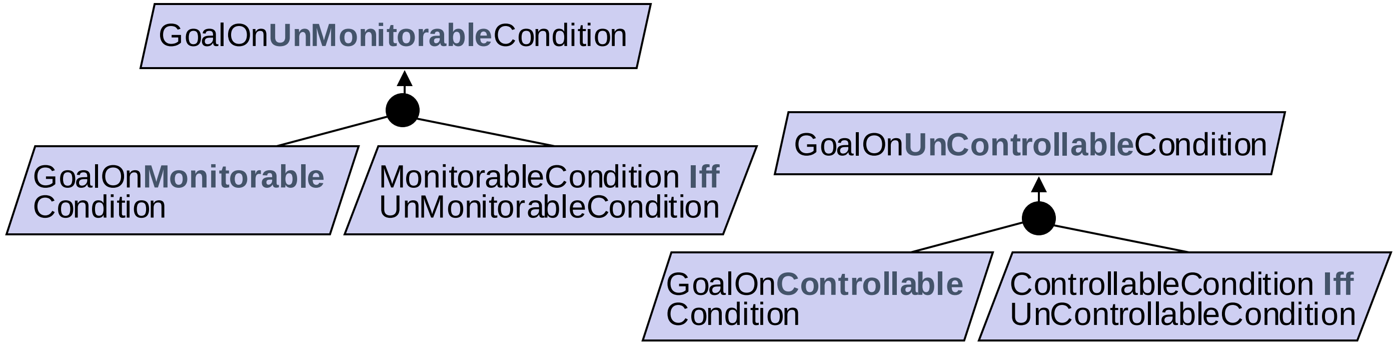
☞ For goal assignment to software-to-be, we must introduce shared "images" of environment objects referenced by the goal

- the shared object tracking its environment counterpart must accurately reflect it (=> new accuracy goal)

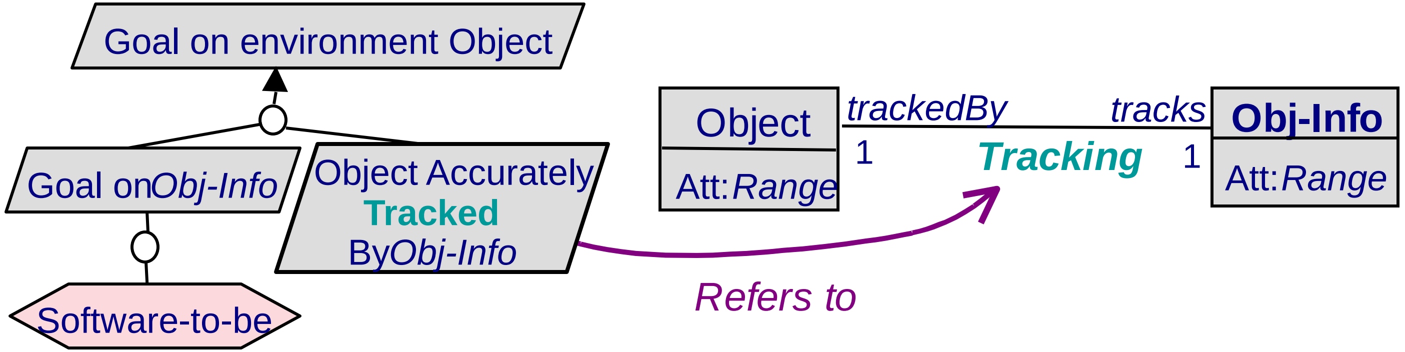
e.g. TrainInfo (Speed, Position) **tracking** Train (Speed, Position)

LoanInfo **tracking** Loan, PatronInfo **tracking** Patron

- cf. goal refinement pattern seen before:


GoalOnUnMonitorableCondition
GoalOnMonitorable Condition
MonitorableCondition Iff UnMonitorableCondition
GoalOnUnControllableCondition
GoalOnControllable Condition
ControllableCondition Iff UnControllableCondition

Introducing software-environment tracking associations: a general pattern



Introducing software-environment tracking associations: a general pattern

RapidTransportation

SafeTransportation

Train

On

Block

Speed
WCS-Dist
Acceleration
Position

SpeedLimit
…

… FastJourney

… NoTrainCollision

SpeedBelow
BlockLimit

Next

Guard

… FastRunToNext
BlockIfGoSignal

WorstCase
StoppingDistance

Following

Tracking

TrainInfo

Driving

Signal

Speed
Position

Position:
{go, stop}

SafeAcceleration
Commanded

TrainAcceleration
FromCommand

…

Command

Acceleration
Speed

AccelerationComputed
FromEstimates

AccurateSpeed&
PositionEstimated

Identifying associations from domain invariants on multiple objects

- Domain properties in goal refinements are to be defined in annotations of the object model (as seen before)
  - invariants on single objects they constrain

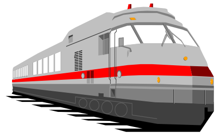
- Invariant seeming to constrain multiple objects ...

? => ? constrains **missing association** among these ?

e.g. “A platform cannot accommodate more than one train at a time”

... constrains **platforms**? ... or **trains**?

=> to be attached to missing association **At** linking **Train** and **Platform**



Derivation links between goal model & object model are bidirectional

- From goals to objects
  - as just seen
- From objects to goals
  - Domain concepts that should “obviously” appear in object model

    WHY? => missing goals in goal model
  - Systematic association decoration with multiplicities

    Prescriptive multiplicity => missing goals in goal model

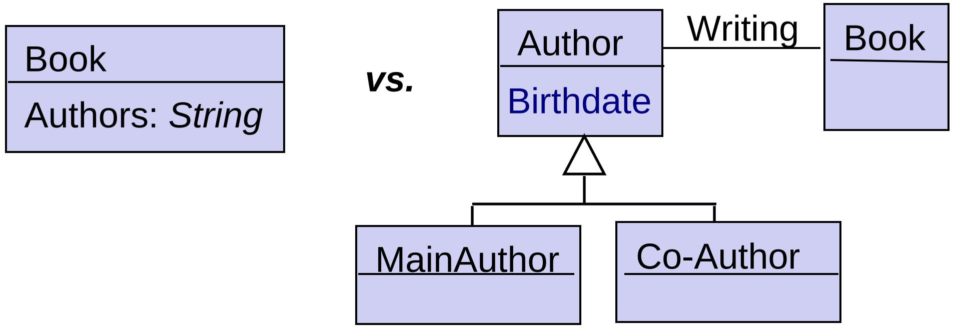
    WHY? => parent goals

    HOW? => subgoals

Object or attribute?

☞ For ***X***: conceptual item in goal specs, make ***X*** an attribute if

- ***X*** is a function: yielding one single value (possibly structured) when applied to conceptual instance
- instances of ***X*** need not be distinguished
- you don't want to attach attributes/associations to ***X***, specialize it, or aggregate/decompose it
- its range is not a concept you want to specialize or attach attributes/associations



Entity, association, agent, or event?

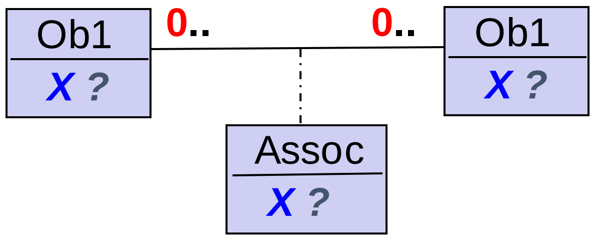
☞ For ***X***: conceptual object in goal specs ...

- instances of ***X*** are defined in one single state
  $\Rightarrow$ event e.g. StartTrain
- instances of ***X*** are active: control behaviors of other object instances
  $\Rightarrow$ agent e.g. DoorsActuator
- instances of ***X*** are passive, autonomous
  $\Rightarrow$ entity e.g. Train
- instances of ***X*** are passive, dependent on other, linked object instances
  $\Rightarrow$ association e.g. Following (Train, Train)

  ☞ N-ary if each of the N parties ...
  - need be considered as objects
  - yields tuples to be distinguished

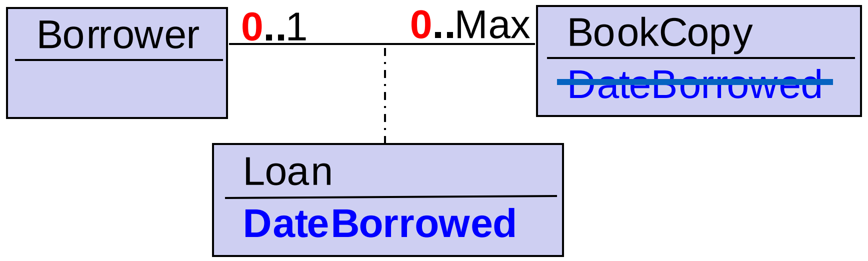
  e.g. Allocation (Meeting, Room, Equipment)

Attribute of a linked object or of a linking association?



☞ Attach attribute to association if it explicitly or implicitly characterizes all participating objects

- esp. if possibly no instance currently in some role, to avoid losing info
- e.g. who did borrow this book copy?

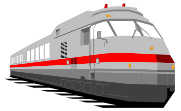


Aggregation or association?

☞ For ***X*** a structural link between "composite" & "component" objects, make it an association if any of these holds:

- ***X*** has a domain-specific *InstanceOf* semantics
- component & composite objects seem independent
  - component not subordinate to composite as in composition
- attributes or associations need be attached to the link type
- in case of doubt (prefer association over aggregation)

e.g. In (Car, Train)



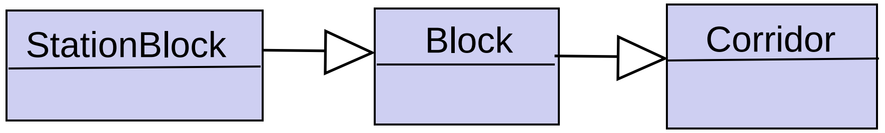
Specializing, generalizing concepts

☞ Identify **specializations** from classification expressions & discriminant factors in goal/domprop specs

- taxonomical keywords *"types of"*, *"kinds of"*, *"category"*, *"class"*, ...
- meet object criteria?
- relevant commonalities to factor out, specifics to discriminate?
- multiple classifications under discriminating attributes?

☞ Identify **generalizations** from objects characterized by similar attributes, associations, domain invariants

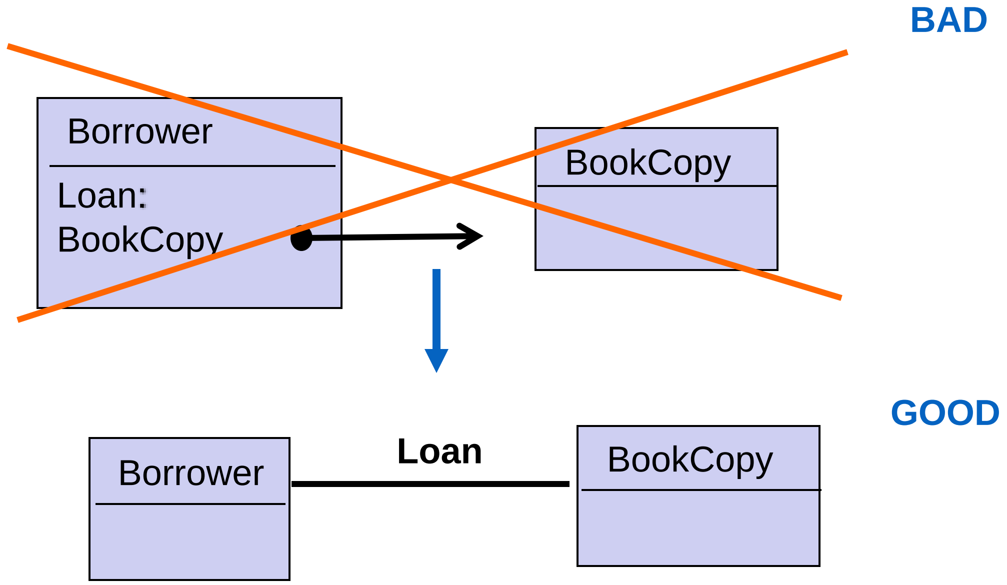
- bottom-up search for common abstractions, not necessarily visible in the system e.g., BorrowableItem
- if worth doing so, without cluttering the model



Building object models: bad smells ☠

☞ Avoid "pointers" to other objects as attributes

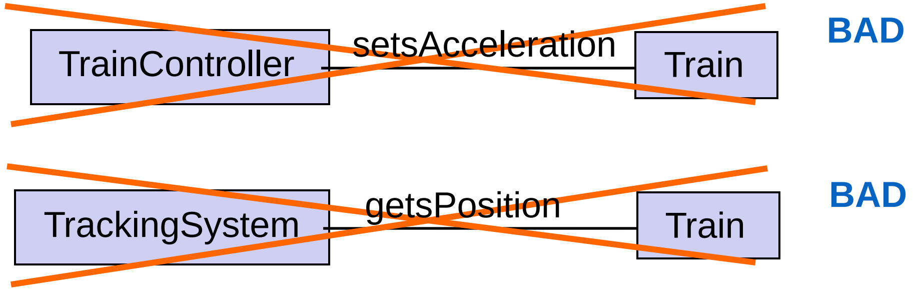
- use binary associations instead

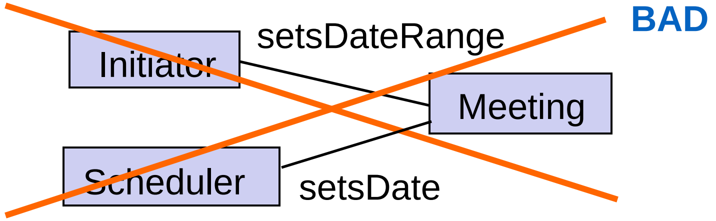


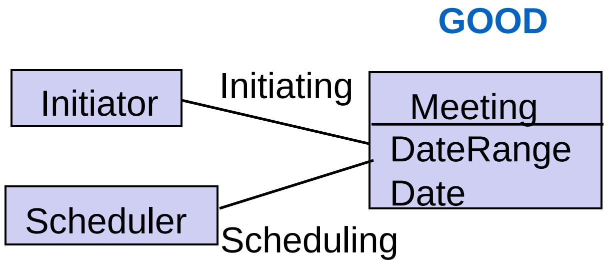
Building object models: bad smells

☞ Avoid non-structural links pertaining to other views

- ~~**monitoring/control** links from agent model (context diagram)~~
- Static and structural links only



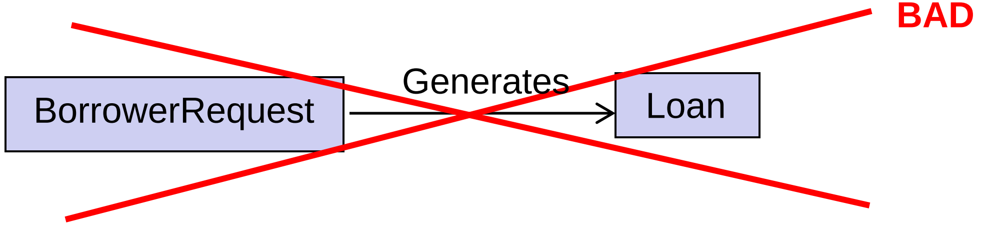


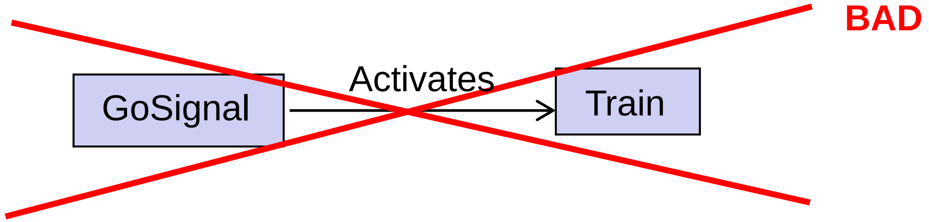


Building object models: bad smells

☞ Avoid non-structural links pertaining to other views

- dynamic links from behavior model (state diagram)





Building object models: bad smells

☞ Avoid obscure names for objects & attributes

- suggestive shortcut of their annotated definition
  - don’t forget precise definition!
  - don’t confuse terms! e.g. Book vs. BookCopy
- from problem world, NOT implementation-oriented
  - Bad JPEG_File , Book_File
  - Good Picture , Directory
- specific, NOT vague
  - Bad Person , Form
  - Good Patron , RegistrationForm
- commonly used, NOT invented
  - Bad PersonalIdentificationCard, ConferenceBook
  - Good StudentCard, Proceedings