Advanced ***Visual*** Modeling:
Beyond UML

Yossi Gil
Technion, Israel & IBM Research, USA
John Howse
University of Brighton, UK
Stuart Kent
University of Kent @ Canterbury, UK

http://www.cs.ukc.ac.uk/constraintdiagrams http://www.it.bton.ac.uk/Research/vmg
yogi@cs.technion.ac.il, John.Howse@brighton.ac.uk, sjhk@ukc.ac.uk

---

Some things need fixing in UML…

- Needs rearchitecting
  - a family of languages, profiles
  - distinction between concrete/abstract syntax, semantics
- Needs a precise definition
  - to unify concepts and integrate notations
  - to remove ambiguity
- Needs richer model management constructs
  - package composition and merging
  - patterns
- Needs better tool support
  - eXtreme modeling

This tutorial…

- <u>NOT</u> about fixing UML, <u>but</u> looking at what might succeed it in terms of notations and tools
- Focus on ***visual*** notations and tools to support them
  - expressivity
  - intuitiveness
  - coherence
- Focus on specifying behavioral constraints, not model management
- Aware of need for precision and to unify and integrate concepts underpinning the various notations

---

Outline

PART I: Static Behaviour

- ➡ **From UML to sets and spiders**
- **Spider diagrams - the details**
- **Constraint diagrams - the details**
- **Constraint trees**
- **Theoretical foundations**

PART II: Dynamic Behaviour

- **3D filmstrips & 3D sequence diagrams**
- **Contract boxes**
- **Other 3D diagram ideas**

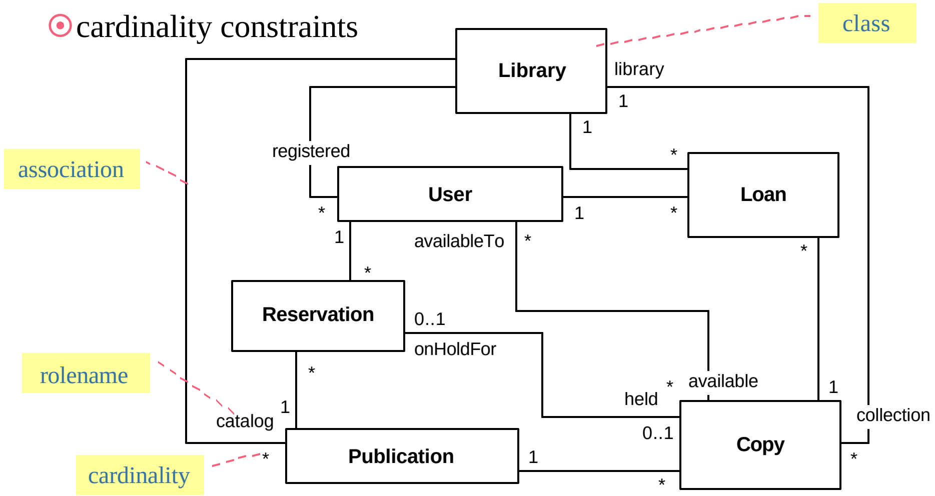
PART III: Tools

- **Extreme modeling - a tools manifesto**
- **Tools architecture and interchange**
- **Tools available now**
- **Plans for the future**

Current Notations: Class diagrams

Provide:

- vocabulary
- cardinality constraints



class

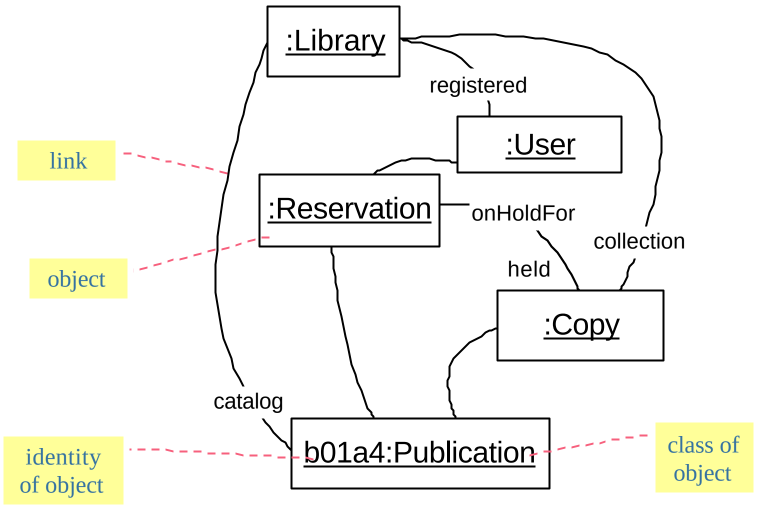
association

rolename

cardinality

Current Notations: Object diagrams

Examples/instances/snapshots of system state



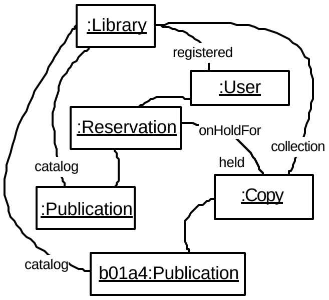
link

object

identity of object

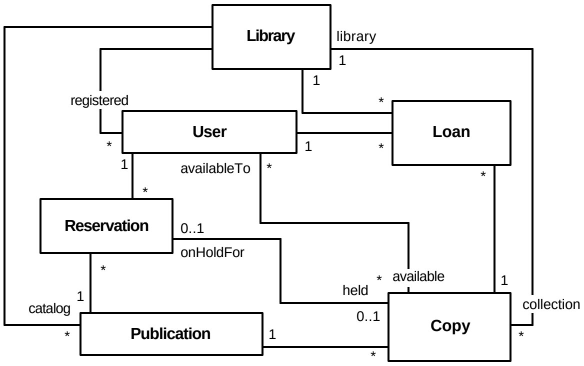
class of object

Class diagrams are not enough



Is the snapshot:

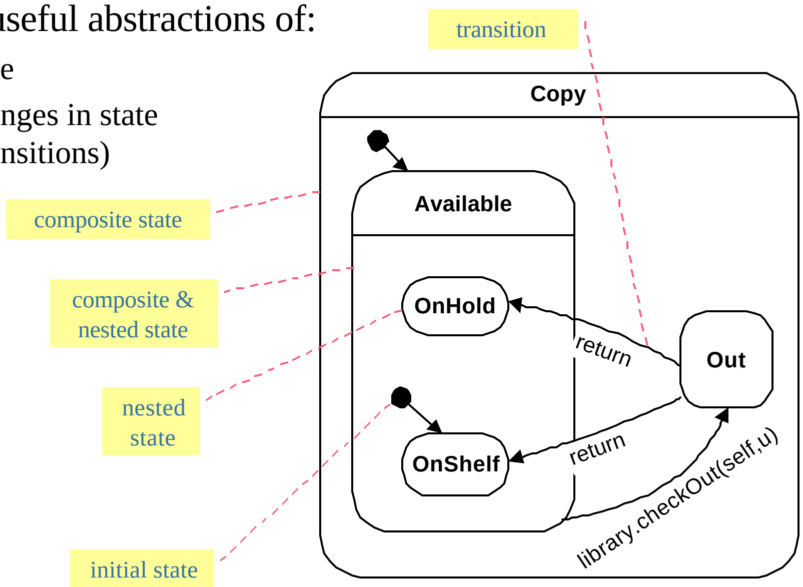
- a valid instance of the class diagram?
- valid for the domain?



Current Notations: State diagrams

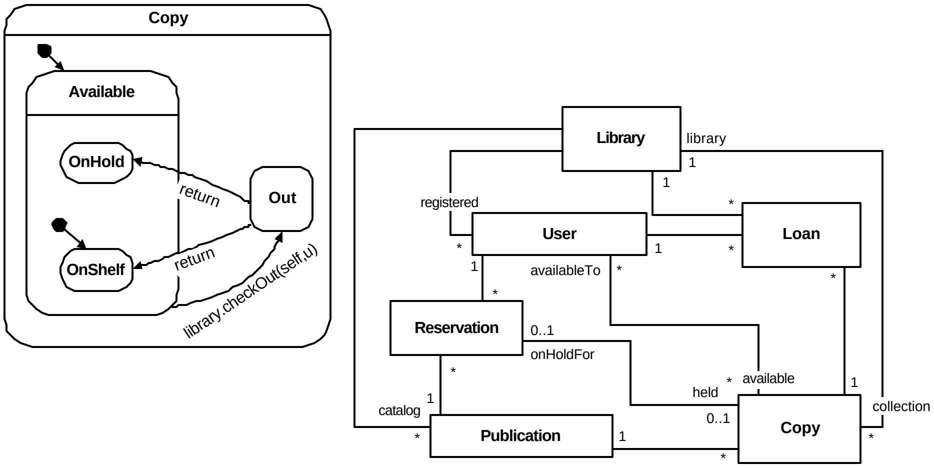
Shows useful abstractions of:

- state
- changes in state (transitions)



States & class diagrams

What are the values of the association links from any c:Copy, when c is OnHold / OnShelf / Out ?



Pros (+) and Cons (-)

- *Class diagrams*
  - + concise
  - + defines language & cardinalities
  - – poor at showing relationships between associations and navigation routes
- *Object Diagrams*
  - + examples are always informative
  - + can show navigation routes and relationships between associations…
  - – but only for specific examples
- *States*
  - + useful abstractions of state space
  - – relationship with data not clear

What’s missing?

- A notation for expressing
  - constraints involving navigation routes, and
  - relationships between navigation routes
- Mapping between state and class diagrams
  - depends on your interpretation of state diagrams

Existing Solutions

- A notation for expressing
  - constraints involving navigation routes, and
  - relationships between navigation routes

The object constraint language (OCL): ***a (textual) variant of (1st order predicate logic) FOPL for expressing constraints***

  - ***textual not visual***
    (ok, if you think UML is as about as visual as you’ll ever get)

- Mapping between state and class diagrams
  - depends on your interpretation of state diagrams

states as Boolean attributes

states as dynamic classes

...

Our approach

Step 1: States are classes

Step 2: Classes are sets

Step 3: “Venn” diagrams show relationships between sets

Step 4: Arrows for navigation routes

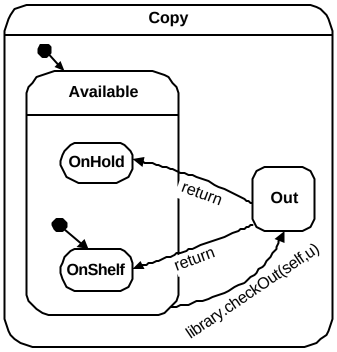
Step 5: Quantified elements

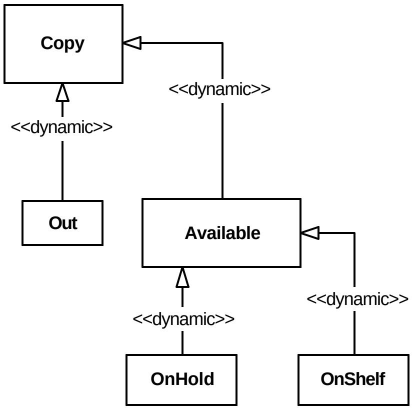
- Steps 1-3 unify state and class diagrams
- Steps 3-5 allow navigational constraints to be expressed
- The solution is entirely ***visual***

Spider/Constraint diagrams

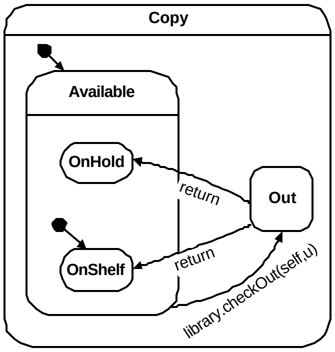
- Described in a series of papers
  - description of notation
  - applications
  - formal semantics
  - mathematical support for tools
- Tool support now available
- Tried out in various applications
  - industrial: telecomms, business rules
  - meta-modeling
- Used in teaching OO modeling

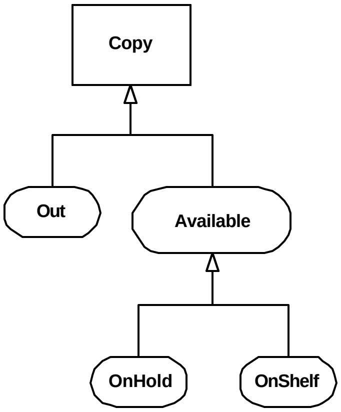
Step 1: States are classes



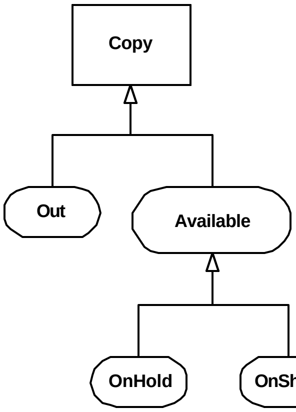


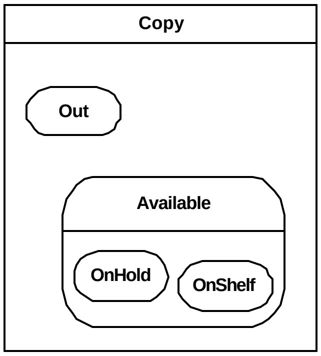
Step 1: States are classes





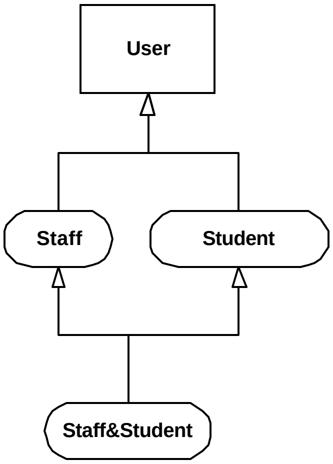
Step 2: Classes are sets; Step 3 Venn diagrams

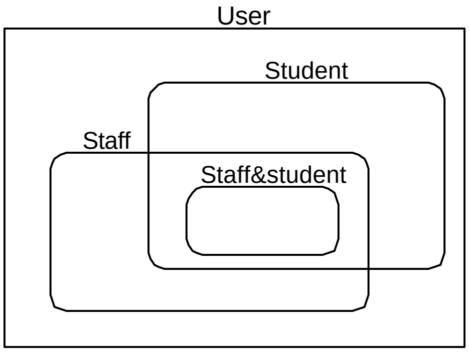




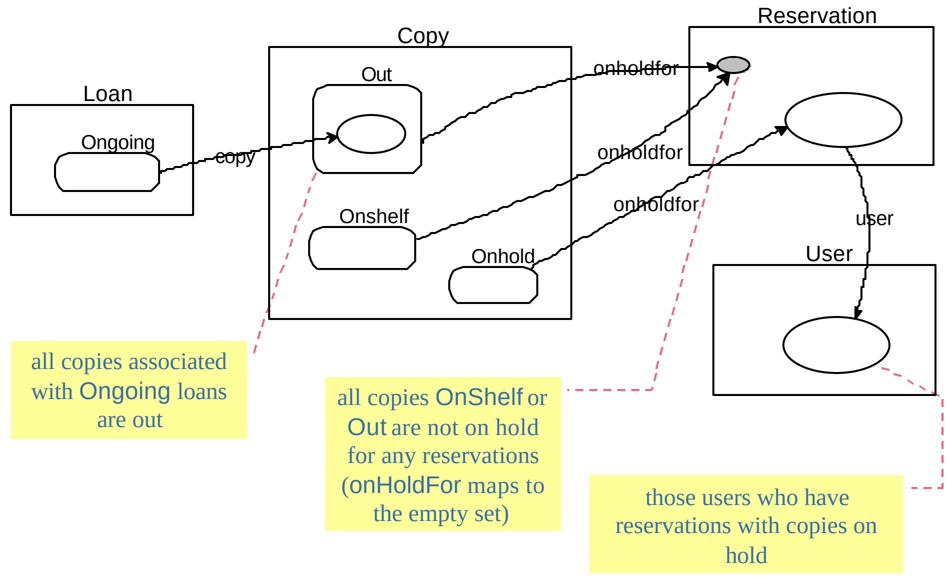
A Venn diagram?
A state diagram?

Step 2: Classes are sets; Step 3 Venn diagrams





Step 4: Arrows for navigation



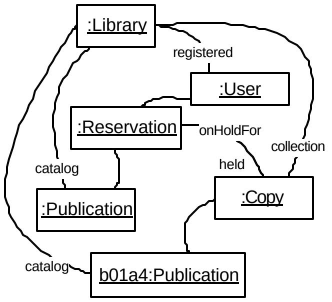
all copies associated with Ongoing loans are out

all copies OnShelf or Out are not on hold for any reservations (onHoldFor maps to the empty set)

those users who have reservations with copies on hold

Step 5: Quantification

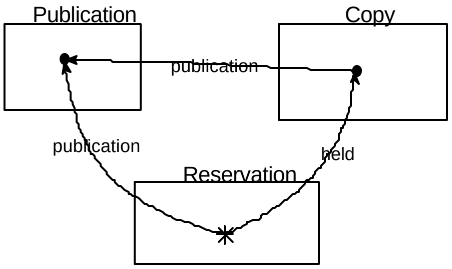
Remember this?



Informally…

"The publication reserved is the publication of the copy put on hold"

The constraint diagram...



In *OCL* (Object Constraint Language of UML)

context r:Reservation **inv**: r.held.publication = r.publication

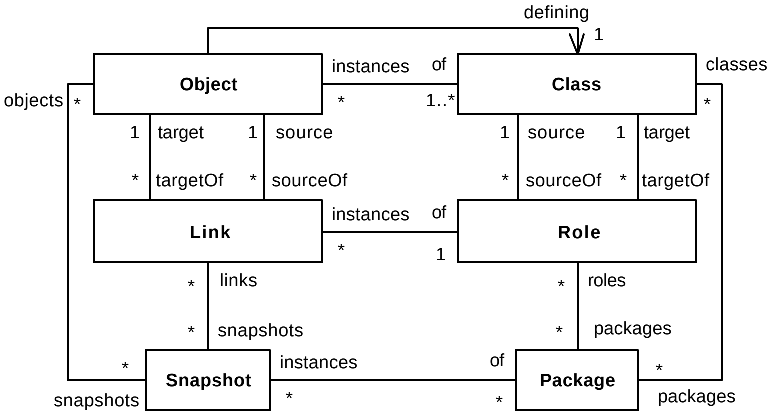
Step 5: Quantification

One for later...

> "In any library, a copy which is on the shelf is available to all registered users; a copy which is on hold is available only to the user who made the reservation."

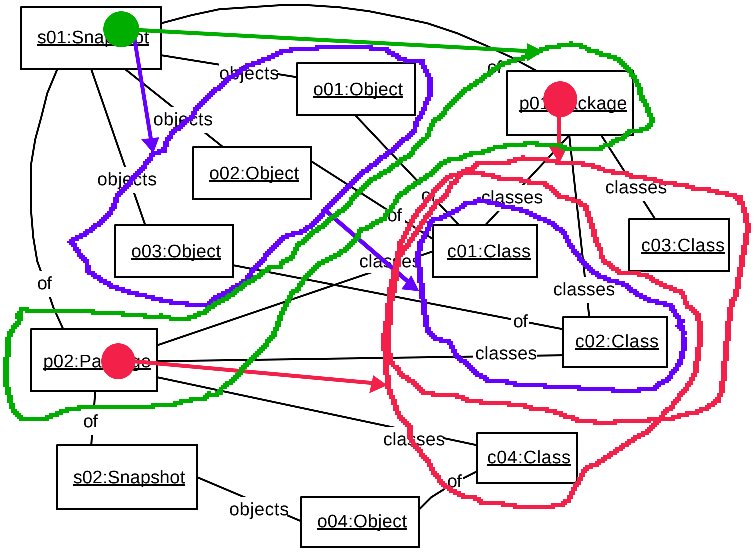
Intuitive? Another example…

A fragment of **a** UML meta-model

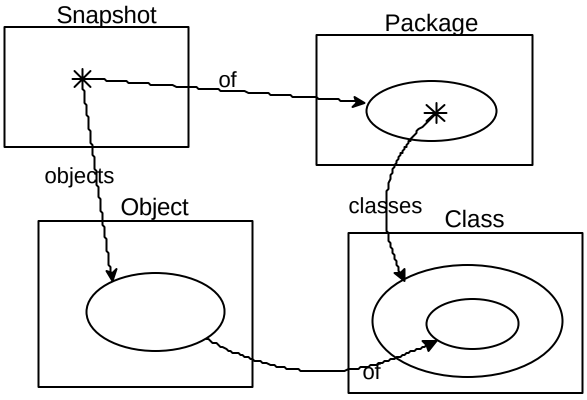


Understanding an OCL constraint

context s:Snapshot: s.of -> forAll(p | p.classes->includesAll(s.objects.of))



The equivalent constraint diagram



Outline

PART I: Static Behaviour

- From UML to sets and spiders
- ➡ Spider diagrams - the details
- Constraint diagrams - the details
- Constraint trees
- Theoretical foundations

PART II: Dynamic Behaviour

- 3D filmstrips & 3D sequence diagrams
- Contract boxes
- Other 3D diagram ideas

PART III: Tools

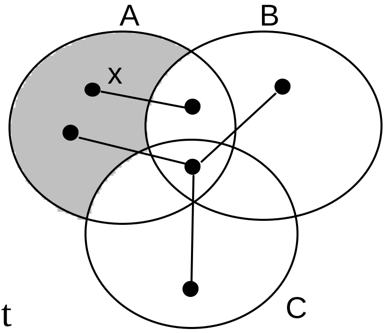
- Extreme modeling - a tools manifesto
- Tools architecture and interchange
- Tools available now
- Plans for the future

What are spider/constraint diagrams?

- ✳ A visual language for first order logic.
  - **Intuitive:** replaces textual based OCL and other mathematical symbolic languages
  - **Familiar:** based on Venn diagrams and Euler Circles
  - **Expressive:** extends the above notations
    - → Elegantly solve many of the topological restrictions and clunkiness of Venn diagrams
- ✳ **Scope**
  - Anywhere where first order logic constraints are required
  - Specifically, an alternative syntax for OCL in UML
    - → system and class ***invariants***
    - → ***pre*** and ***post*** conditions

A simple example

- **Participants**: sets *A*, *B*, and *C*.
- **Diagram asserts**:
  - There is at least one element which is either contained in one of these sets but in none of the others, or which is contained in all of them
  - There is at least one element, different than the first which is in *a* but not in *c*.
  - Other than these, there are no other elements contained in A but in none of the others.



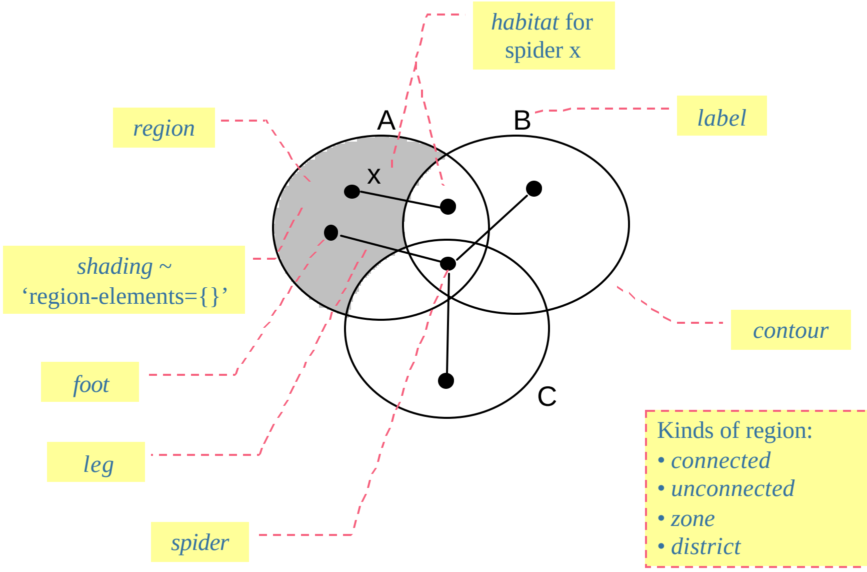
$$\exists x, y \bullet x \neq y \wedge x \in A - C \wedge y \in \left(A \cup B \cup C \cup A \cap B \cap C - A \cap B - A \cap C - B \cap C\right) \wedge$$
$$\forall z \bullet z \in A - B - C \Rightarrow (z = x \vee z = y)$$

`(A-C)->exists(x | (A->union(B)->…)->exists y | x <> y and A-B-C->forAll(z | …))`

Virtues of the notation

- **Concise** and **Precise**
  - compare to English text description
- **Familiar** and **Intuitive**
  - based on concepts introduced in elementary school
  - compare to mathematical notation.

Key terminology



habitat for spider x

label

region

shading ~ ‘region-elements={}’

contour

foot

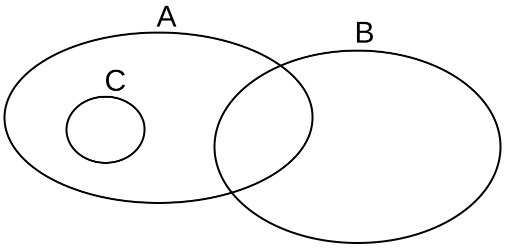
leg

spider

Kinds of region:
- *connected*
- *unconnected*
- *zone*
- *district*

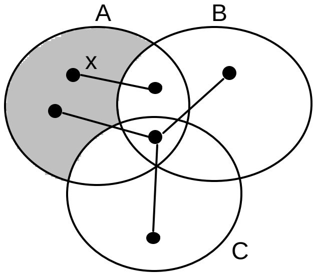
Contours & regions (sets)

- *Contours* denote **sets**.
  - Contours may intersect, dividing the plane into minimal regions also called *zones* which correspond to set intersection.
  - A *region* is any non-empty collection of zones.
    - →This collection is not necessarily *connected*.
  - A *basic region* or *district* is the region that consists of all zones bounded by a contour.



Spiders (elements)

- *Spiders* denote **elements**
  - Spiders may span more than one zone. They may have a *foot* in each zone, which means that the element may be contained in the region that is the union of those zones
  - A *shaded zone* has no elements other than those designated by the spiders in it.
- A spider resides in a region, sometimes called its *habitat*.

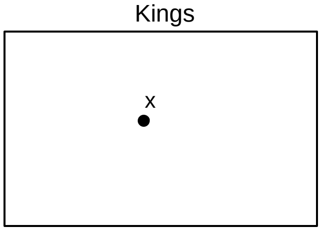


Spiders (cont.)

- Spiders can be *given* or *existentially* quantified

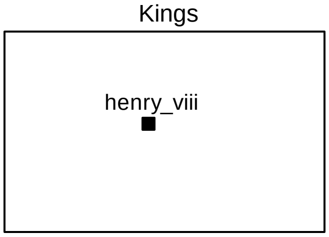
$\exists x \bullet x \in Kings$

Kings->exists(x | true)

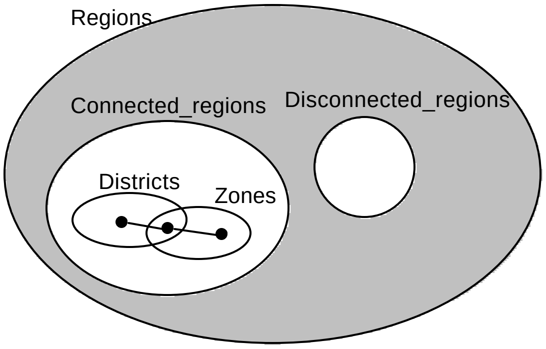


$henry\_viii \in Kings$

Kings->includes(henry_viii)

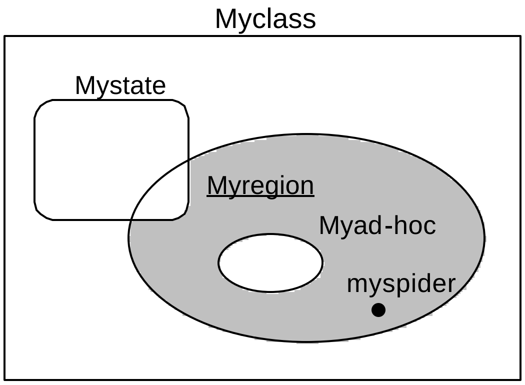


A meta-modeling example

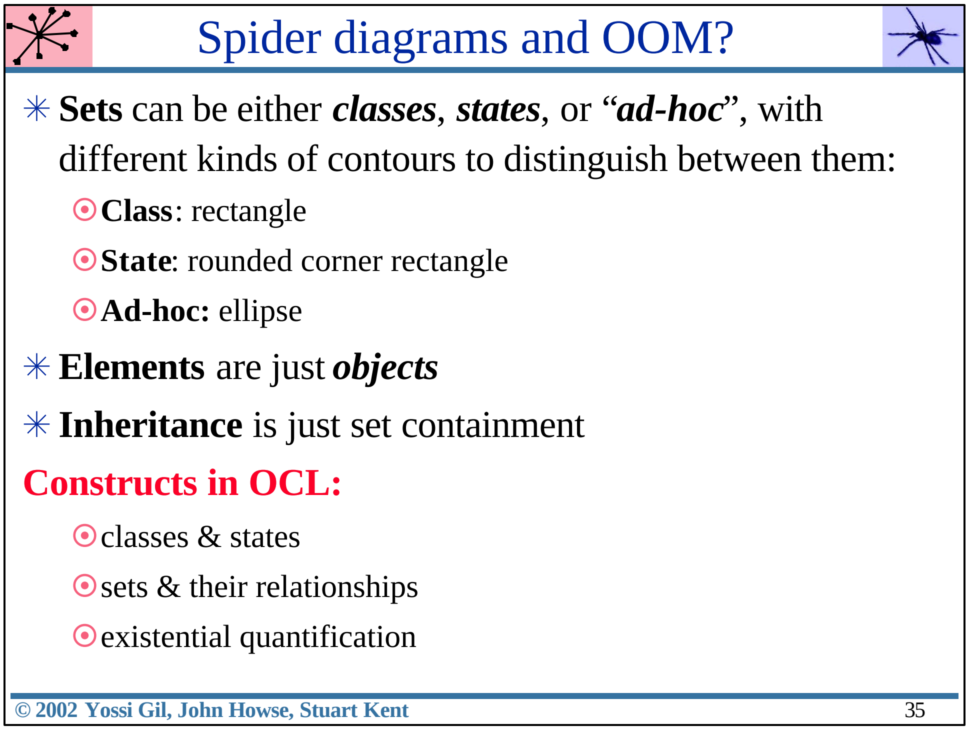


The diagram specifies all spider diagrams (including empty): they must have at least one connected region which can be either basic or minimal or both. All spider diagrams have a boundary contour (we don't usually bother to draw it).

Labels



- Alternative labeling schemes could be invented
  - and are admitted by our tool

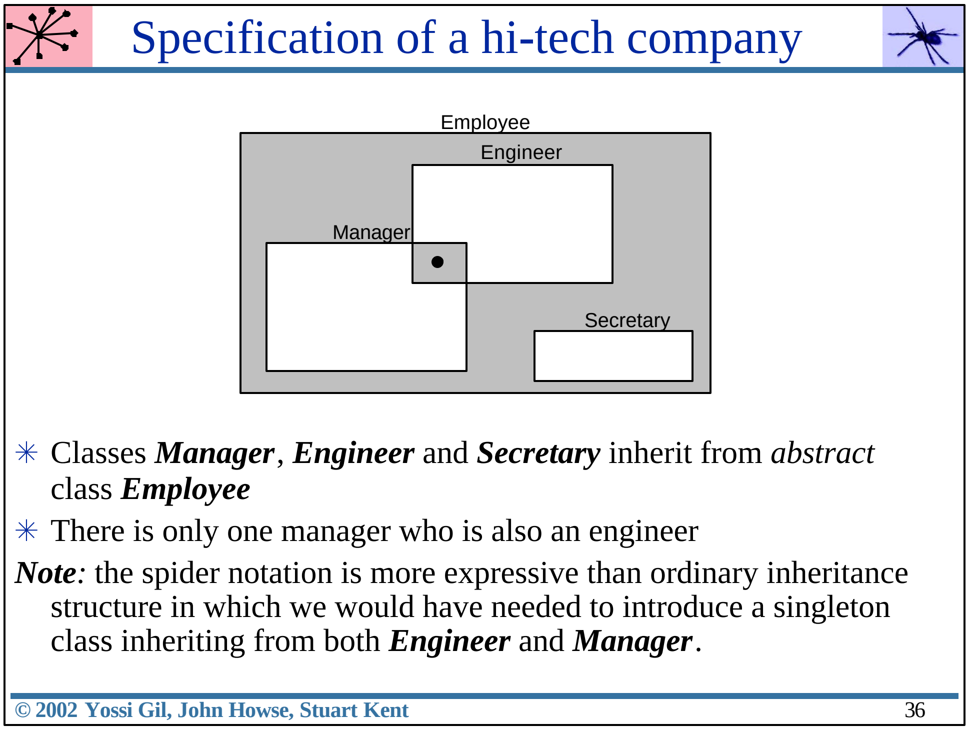


Spider diagrams and OOM?

- **Sets** can be either ***classes***, ***states***, or "***ad-hoc***", with different kinds of contours to distinguish between them:
  - **Class**: rectangle
  - **State**: rounded corner rectangle
  - **Ad-hoc:** ellipse
- **Elements** are just ***objects***
- **Inheritance** is just set containment

Constructs in OCL:

- classes & states
- sets & their relationships
- existential quantification

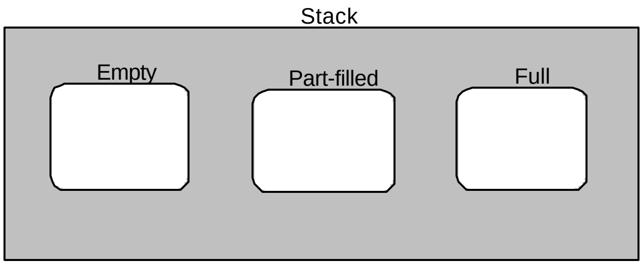


Specification of a hi-tech company

- Classes ***Manager***, ***Engineer*** and ***Secretary*** inherit from *abstract* class ***Employee***
- There is only one manager who is also an engineer

Note: the spider notation is more expressive than ordinary inheritance structure in which we would have needed to introduce a singleton class inheriting from both ***Engineer*** and ***Manager***.

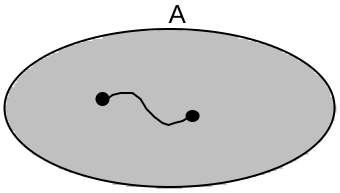
States and sets



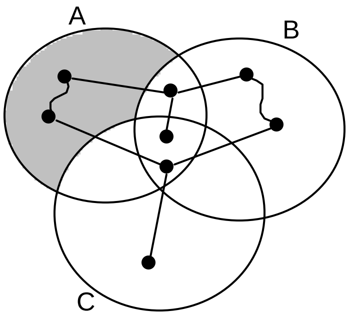
- A state (as in Harel statecharts) is identified with the ***set*** of all objects which are in that state.
  - The state ***Empty*** is the set of all stacks which are empty
- We see from the diagram that all stacks are either empty, full, or part-filled.

Sociable spiders: friends & strands

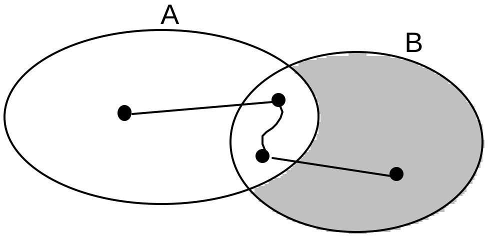
- Spiders connected by a *strand* are called *friends*.
  - This notation means that the two elements *may* be equal.



- Strands actually connect feet, which must be in the same zone.
  - Thus, two spiders may be connected more than once.
  - This gives rise to very expressive and interesting semantics.
  - In the example, the two elements may be equal only if they occur in A – (B ∪ C), or in B – (A ∪ C).

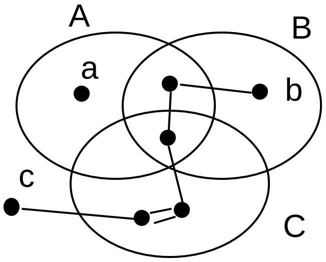


What does this diagram say?

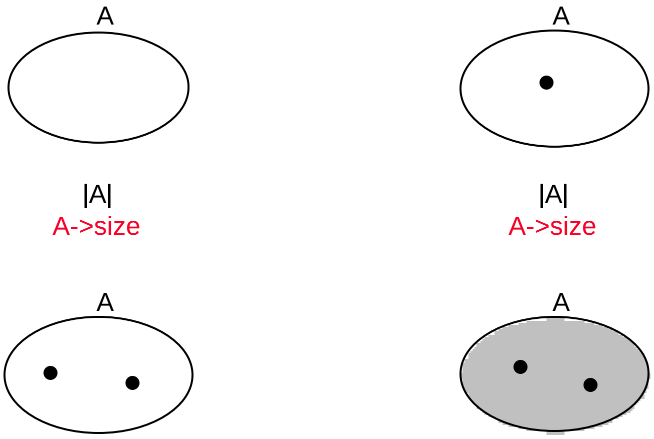


Ties - mates

- Spiders connected by a ***tie*** are called ***mates***.
- This notation means that the two elements *must* be equal within the zone in which the tie appears.
- In the example, the two elements *b* & *c* are the same if they both belong to region (C – A) – B, otherwise they are distinct.



Specifying set cardinalities



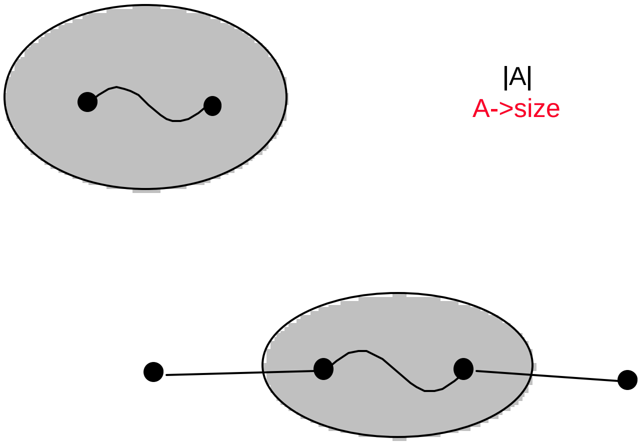
|A|
A->size

|A|
A->size

|A|
A->size

|A|
A->size

How many elements?

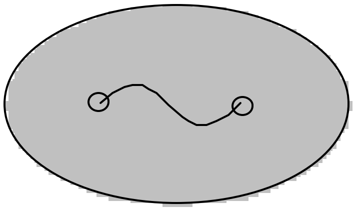


|A|
A->size

|A|
A->size

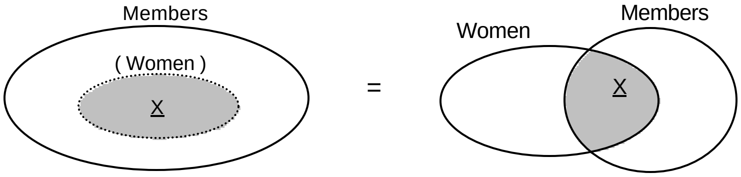
Schrödinger spiders

Denotes a **set** with zero or one element. Like a Schrödinger cat - existence (of element) is in question!

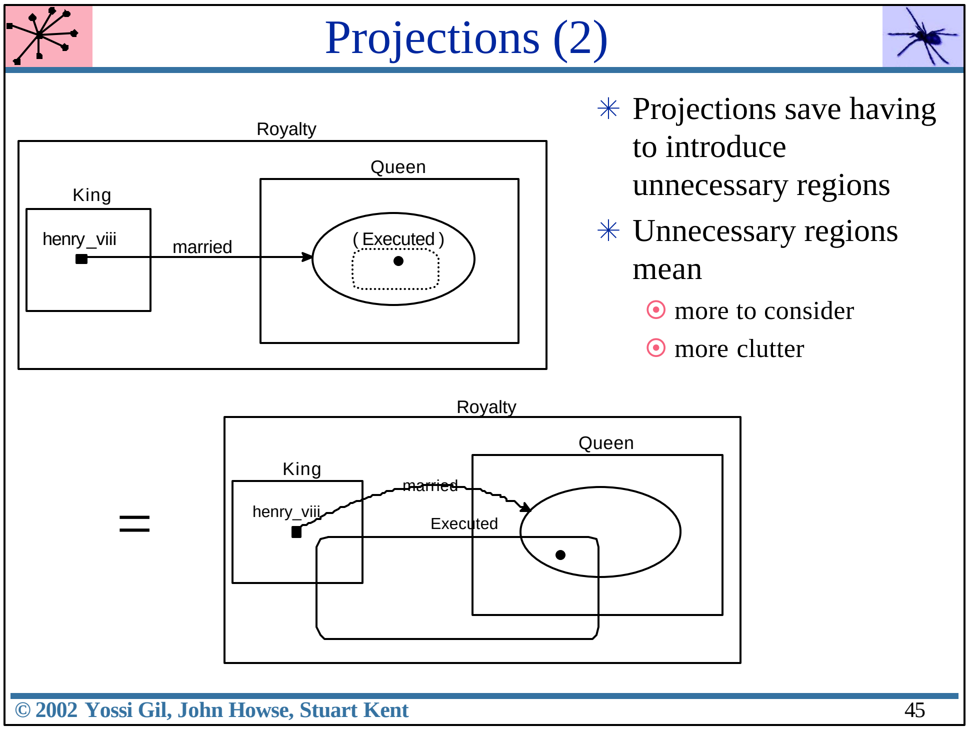


Projections (1)

- **Projections**: used to denote a set taken in a specific context.
  - **Metaphor**: set is projected into the plane of interest.
  - **Notation**: dotted contours

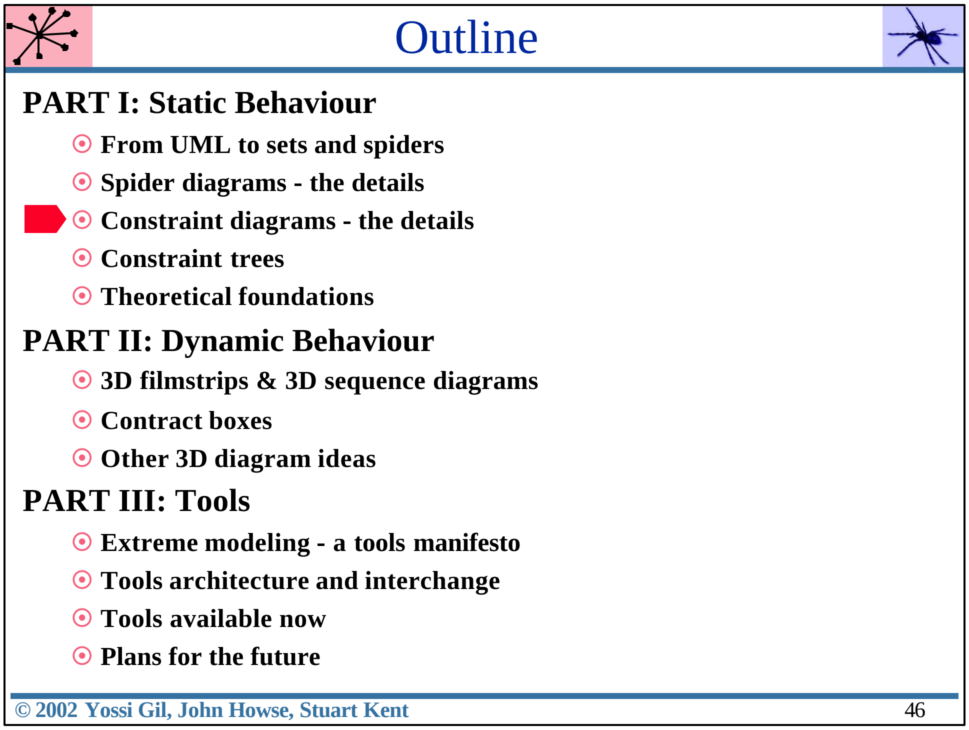


Clubs in the St. James area of London!



Projections (2)

- Projections save having to introduce unnecessary regions
- Unnecessary regions mean
  - more to consider
  - more clutter



Outline

PART I: Static Behaviour

- **From UML to sets and spiders**
- **Spider diagrams - the details**
- ➡ **Constraint diagrams - the details**
- **Constraint trees**
- **Theoretical foundations**

PART II: Dynamic Behaviour

- **3D filmstrips & 3D sequence diagrams**
- **Contract boxes**
- **Other 3D diagram ideas**

PART III: Tools

- **Extreme modeling - a tools manifesto**
- **Tools architecture and interchange**
- **Tools available now**
- **Plans for the future**

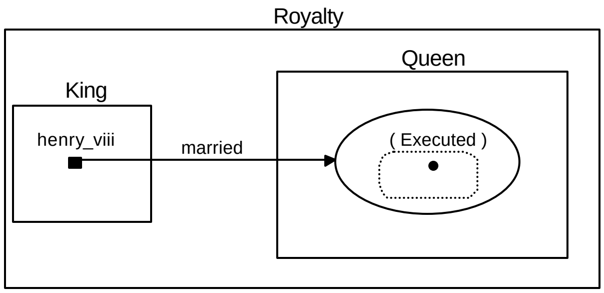
Constraint diagrams

- **Spider Diagrams:** set theoretical expressions
- **Constraint Diagrams**: relations between sets
  - **New Notations:** *Arrows* and *Wildcards*
  - **Expressive power:**
    - →arrows represent *navigation* of relations between sets
    - →generalize commutative diagrams
      - **Commutative diagrams:** result of taking two paths is *exactly* the same.
      - **Constraint diagrams:** result of taking two paths are sets which can be related by venn/euler diagrams
    - →universal quantification

Constructs in OCL:

- navigation expressions
- existential quantification

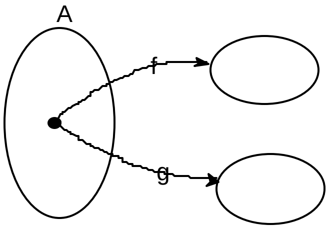
One-arrow example



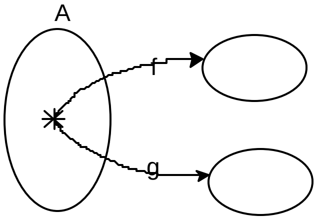
- Asserts
  - The class King has an object named Henry VIII in it.
  - All women that Henry VIII married were queens.
  - There was at least one women he married who was executed.

King->includes(henry_viii) and
Queen->includesAll(henry_viii.married->asSet) and
henry_viii.married->asSet->intersection(Executed)->exists(x | true)

Universal vs. existential quantification



There exists an *x* in *A*, such that the sets *x.f* and *x.g* are disjoint.



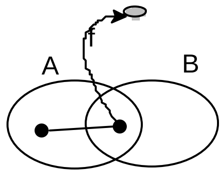
For all *x* in *A*, the sets *x.f* and *x.g* are disjoint.

x.f is a shorthand for the set {y | (x,y)∈ f}
or just x.f->asSet in OCL

What’s in an arrow?

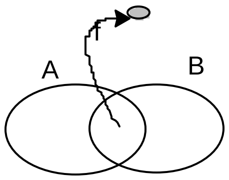
- **Label:** the name of the relationship
- **Source:** a ***set*** or an ***element*** from which ***navigation*** (the relationship computation) begins:
  - Wildcard (universal spider)
  - Existential spider
  - Given spider
  - Contour
  - Schroedinger (optional set) or derived spider (singleton set)
  - Zone
  - Collection of zones (a region!)
- **Target:** the map of the source. If the source consists of more than one element, then the target is the set formed by the union of the maps of all elements in the source.

Arrow source: examples



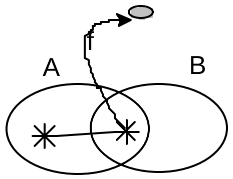
$\exists x \bullet x \in A \wedge x.f = \{\}$

A->exists(x | x.f->isEmpty)



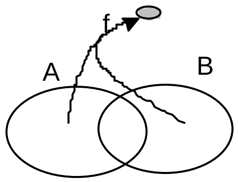
$(A \cap B).f = \{\}$

A->intersection(B).f->isEmpty)



$\forall x \bullet x \in A \wedge x.f = \{\}$

A->forAll(x | x.f->isEmpty)

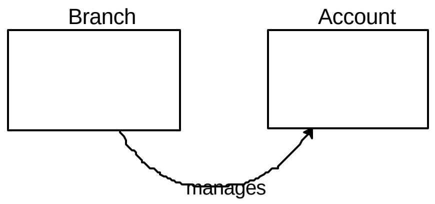


$((A-B) \cup (B-A)).f = \{\}$

(A-B)->union(B-A).f->isEmpty)

More on targets

- The target of an arrow is a set.
  - **Given contour (class/state/ad-hoc or regions defined by them)**: arrow reads as set equality, e.g. Branch.manages->asSet = Account



all accounts are managed by some branch

should really be Account.AllInstances

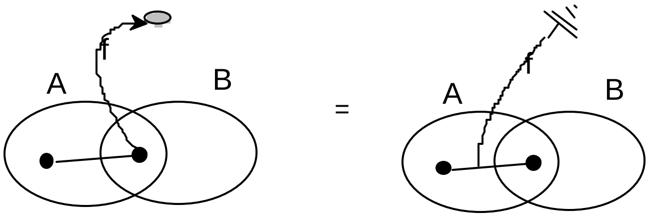
  - **Derived contour:** the arrow defines the set (most common case).
    - →Ellipse that is not target of arrow is ad-hoc.
  - **Derived spider:** Treated as derived, singleton sets (not existentially quantified). No wildcards.
  - **Schroedinger spider:** Treated as derived, optional set

Different kinds of spider/contour

- **Spiders**
  - Wildcard (universal spider): ✳
  - Existential spider: ●
  - Given spider: ■
  - Schroedinger spider (= optional ad-hoc/derived set): ○
  - Derived spider (= singleton, derived set): ●
- **Contours**
  - Given contour
    - → class - rectangle
    - → state - rounded rectangle
    - → ad-hoc - ellipse
  - Derived contour - ellipse (at target of an arrow)

Syntactic sugar

- An arrow ending in an electrical ground symbol
- Arrow sourced on leg rather than foot



Exercise: remember this...

"In any library, a copy which is on the shelf is available to all registered users; a copy which is on hold is available only to the user who made the reservation."

Outline

PART I: Static Behaviour

- From UML to sets and spiders
- Spider diagrams - the details
- Constraint diagrams - the details
- ➡ Constraint trees
- Theoretical foundations

PART II: Dynamic Behaviour (3D modeling)

- Why 3D
- 3D filmstrips & 3D sequence diagrams
- Contract boxes
- Other 3D diagram ideas

PART III: Tools

- Extreme modeling - a tools manifesto
- Existing Tools
- Plans for the future

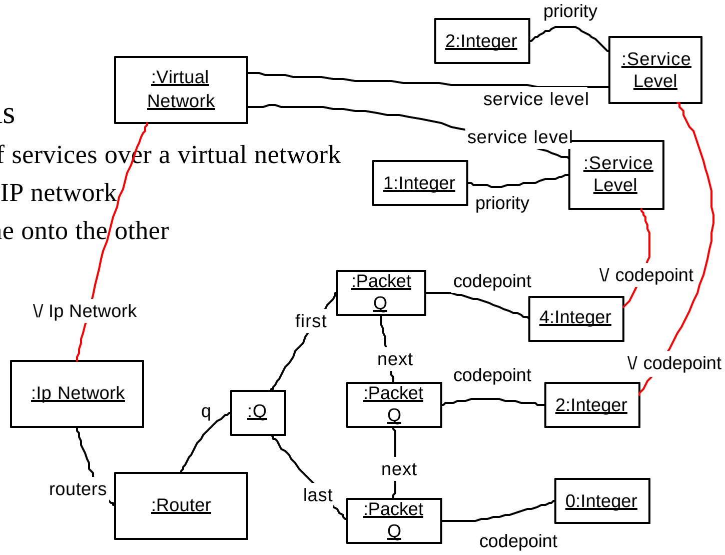
Motivation

- There is difficulty expressing some things on a CD
  - sophisticated constraints on set cardinalities
  - select and reject in OCL (there are work arounds in some cases)
  - negation and disjunction (again, some work arounds)
  - some orderings of quantifers (∃∀ or ∀∃)
  - familiar types such as numbers etc.
    - →can be clumsy e.g. try visualising n<=m<=r
- Mixing diagrams with text means they can be used to partially express a constraint - no longer binary choice
- Mixing mechanism pulls on the idea of a syntax tree
  - allows diagrams to be nested - fixes ordering of quantifiers
  - allows industrial-sized constraints to be organised into multiple, related diagrams

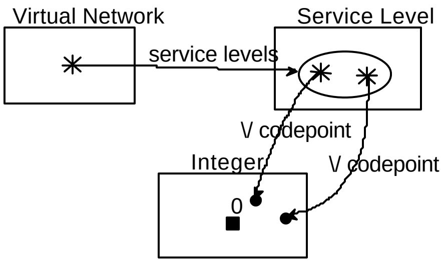
---

Case study (thanks to Nortel)

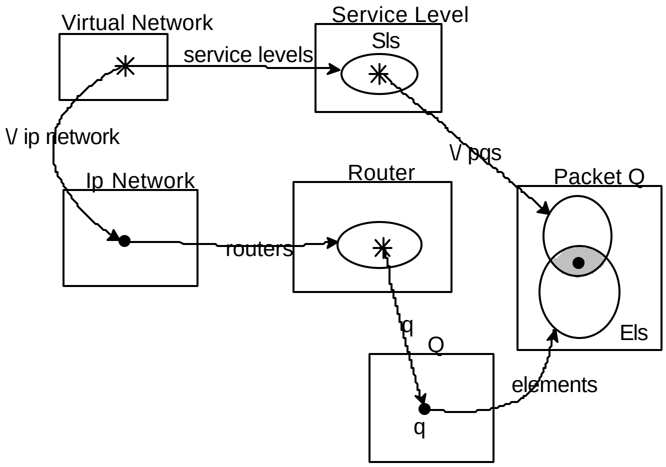
- IP network needs to be configured to deliver services at different levels
- Two models
  - models of services over a virtual network
  - model of IP network
  - realise one onto the other



Service levels to codepoints

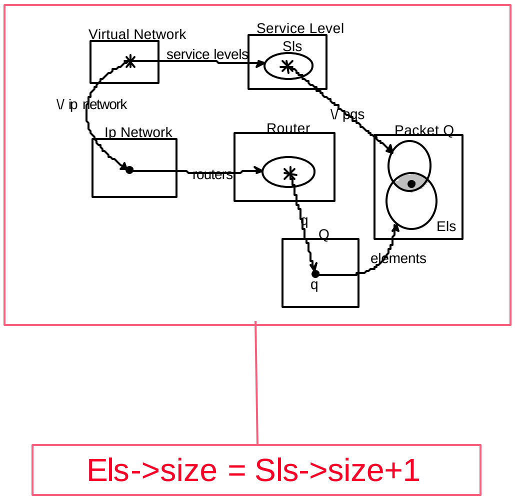


Service levels to Q’s



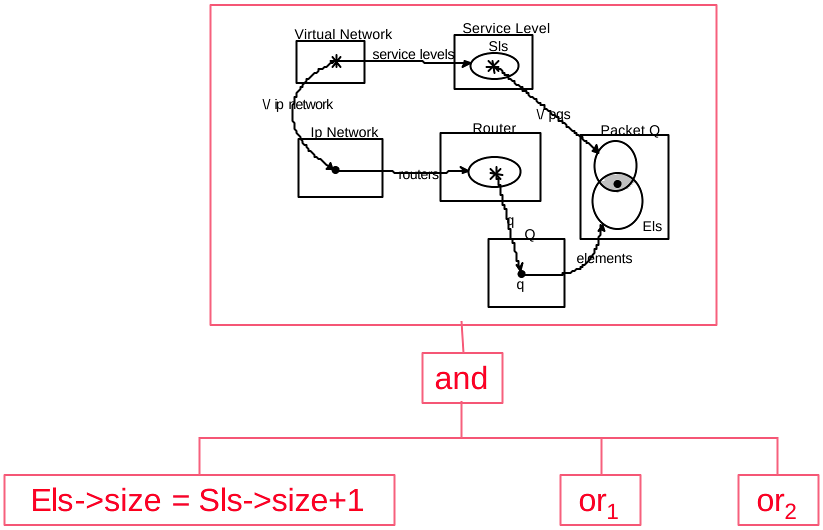
Would also like to say that Els->size = Sls->size+1

Solution - constraint trees



Els->size = Sls->size+1

More generally…



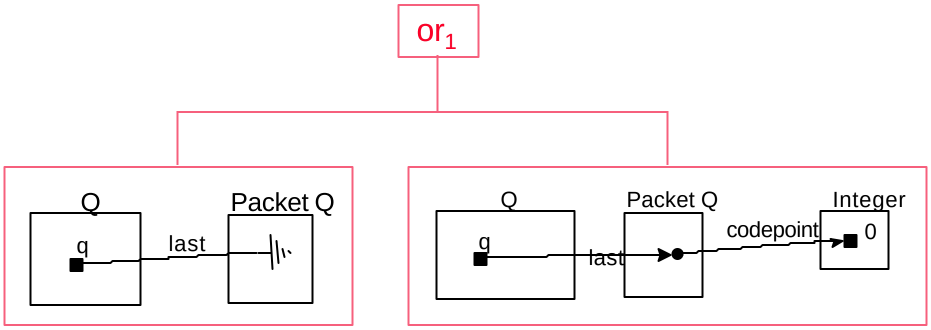
and

Els->size = Sls->size+1

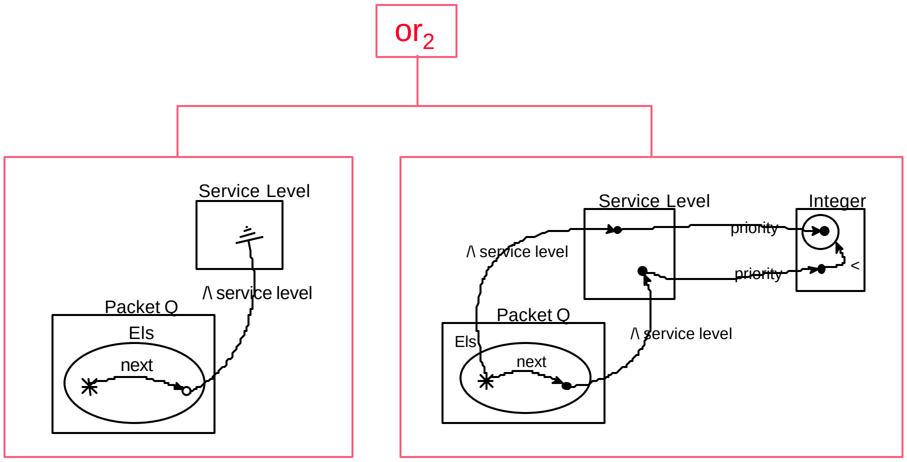
or$_1$

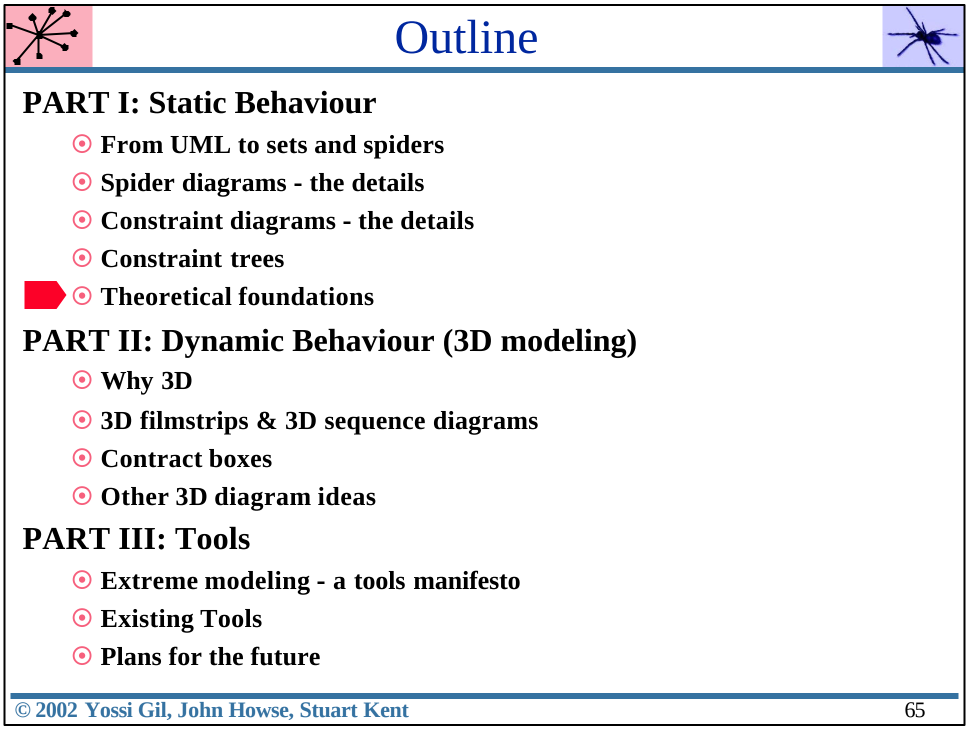
or$_2$

no *last* or codepoint of *last* is 0



Packet Q ordering





Outline

PART I: Static Behaviour

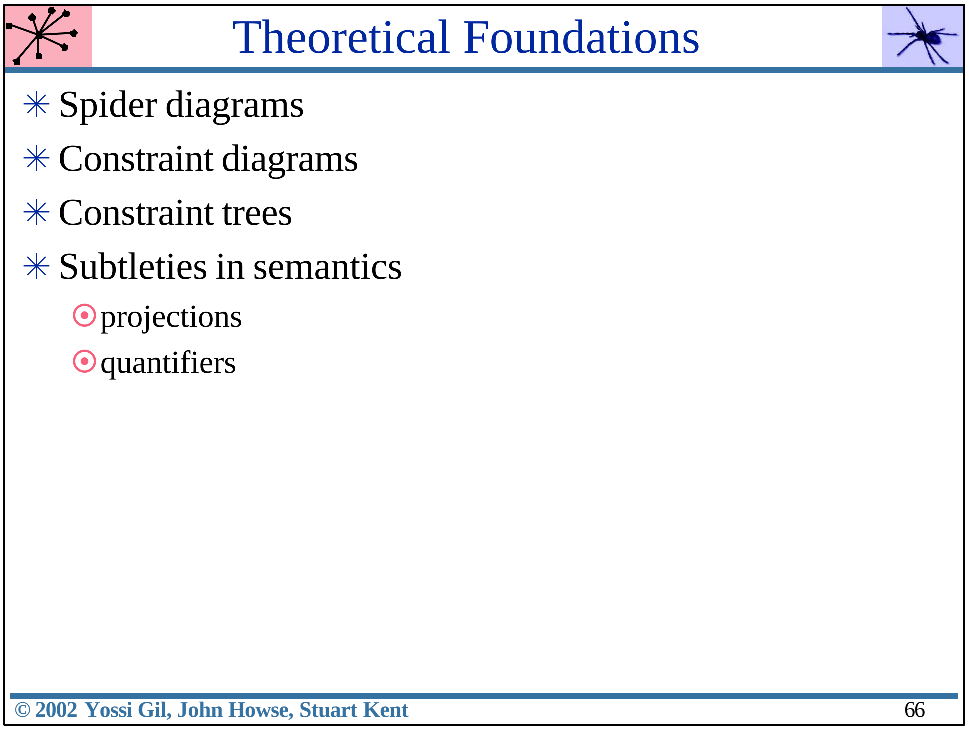
- From UML to sets and spiders
- Spider diagrams - the details
- Constraint diagrams - the details
- Constraint trees
- ➡ Theoretical foundations

PART II: Dynamic Behaviour (3D modeling)

- Why 3D
- 3D filmstrips & 3D sequence diagrams
- Contract boxes
- Other 3D diagram ideas

PART III: Tools

- Extreme modeling - a tools manifesto
- Existing Tools
- Plans for the future



Theoretical Foundations

- Spider diagrams
- Constraint diagrams
- Constraint trees
- Subtleties in semantics
  - projections
  - quantifiers

Spider Diagram Theory

- ✓ Semantics
  - ⦿ m is a model of a spider diagram d if it satisfies the semantics predicate P_d
  - ⦿ projections are non-trivial
  - ⦿ **Result**: spider diagrams are always consistent
- ✓ Reasoning Rules
  - ⦿ **Result**: rules are sound and complete with respect to semantics
- ✳ Algorithms
  - ⦿ checking that a diagram is well-formed (based on definition of syntax)
  - ⦿ translating a diagram into a first order logic formula (based on P_d)
  - ⦿ translating a diagram into OCL (abstract syntax) when used in OO context
- ? Reasoning rules for projections
- ? Compaction of the logical formulae
- ? Exact characterization of expressive power
- ? The inverse translation problem, from formula to diagram

Constraint Diagram Theory

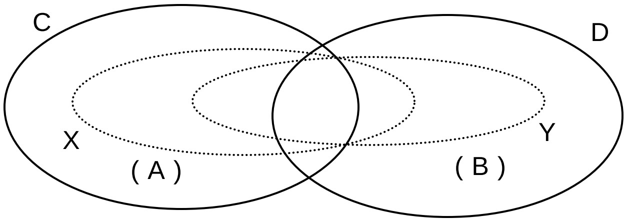
- ✳ Semantics
  - ⦿ m is a model of a constraint diagram d if it satisfies the semantics predicate P_d
  - ⦿ default ordering of quantifiers is non-trivial
  - ? constraint diagrams are always consistent
- ✳ Algorithms
  - ⦿ checking that a diagram is well-formed (based on definition of syntax)
  - ⦿ translating a diagram into a first order logic formula (based on P_d)
  - ⦿ translating a diagram into OCL (abstract syntax) when used in OO context
- ? Reasoning Rules
- ? Compaction of the logical formulae
- ? Exact characterization of expressive power
- ? The inverse translation problem, from formula to diagram

Constraint tree theory

- Semantics
  - m is a model of a constraint tree t if it satisfies the semantics predicate P_t
  - syntax tree says how to compose predicates derived from sub-diagrams
- Algorithms
  - checking that a tree is well-formed (based on definition of syntax)
  - translating a tree into a first order logic formula (based on P_t)
  - translating a tree into OCL (abstract syntax) when used in OO context
    - →allows OCL to be interchanged with diagrams
- Reasoning Rules
  - we note that constraint trees are required to support reasoning rules for spider diagrams

---

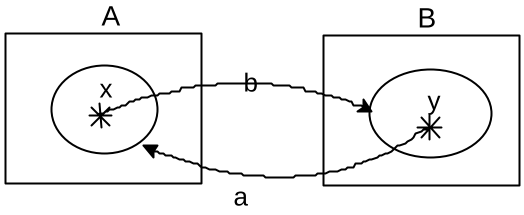
The subtleties of projections



- What's the semantics of the projections X and Y?
- Can be found by a Gaussian elimination procedure on a system of set equations:
- Turns out there is a simpler semantics (phew!)

$$X = A\cap(Y\cup C)$$
$$Y = B\cap(X\cup D)$$

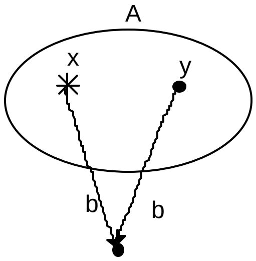
The subtleties of quantifiers



A->forAll(x | x.b->forAll(y | y.a->includes(x)))

or

B->forAll(y | y.a->forAll(x | x.b->includes(y)))



A->exists(y | (A-{y})->forAll(x | x.b=y.b))

or

A->forAll(x | (A-{x})->exists(y | x.b=y.b))

Outline

PART I: Static Behaviour

- **From UML to sets and spiders**
- **Spider diagrams - the details**
- **Constraint diagrams - the details**
- **Constraint trees**
- **Theoretical foundations**

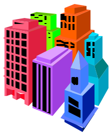
PART II: Dynamic Behaviour (3D modeling)

- ➡ **Why 3D**
- **3D filmstrips & 3D sequence diagrams**
- **Contract boxes**
- **Other 3D diagram ideas**

PART III: Tools

- **Extreme modeling - a tools manifesto**
- **Existing Tools**
- **Plans for the future**

Motivation - Why 3D Modelling?



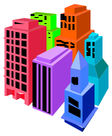
- Technology: Java3D, VRML, ...
- Experience: a lot of work on utilization of 3D
  - 3D Software Visualization
  - 3D Debugging
  - 3D Visual Programming Languages
- Need: Complexity of Software Modeling
  - Only a small amount of information can be displayed in any diagram.
  - Every little bit can help.

3D Modeling seems to be most useful to give a high level overview + zooming

2½D may be more useful than true 3D

3rd dimension can be time (and other things)

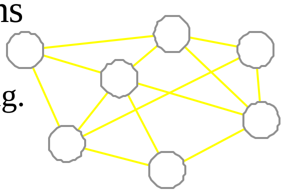
2D Diagrams and Graphs



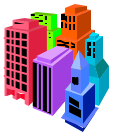
- Examples: E-R, Call-graph, UML
- Recurring theme: graph metaphor
  - Partially ordered sets (procedures)
  - More expressive than sequential text
    - → ~O(n lg n) bits to represent linear ordering of *n* elements
    - → ~$O(n^2)$ bits to represent a graph of *n* elements

    In other words, there are $2^{O(n \lg n)}$ possible linear orderings, which is much smaller than $2^{O(n^2)}$ different possible graphs of *n* nodes.
- 2D: Visual clue to lack of order of nodes
- 3D: Virtually useless for rendering graphs
  - Mostly useful for avoiding intersections, but this advantage is lost in any 2D rendering.

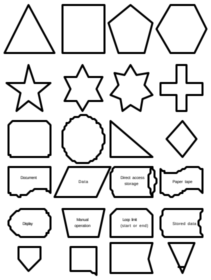
Does this mean that we should not use 3D?



Graph Theme: Variations



- **Problem**: Mathematical notion of simple graphs is not expressive enough. We need nodes and edges of different kinds.
- Kinds of nodes
  - Booch Object Diagram: 7 kinds of nodes
  - Booch Module Diagram: 10 kinds of nodes



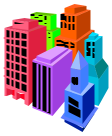
- **Problem**: 2D limits the <u>variety</u> of edges which can be distinguished iconically

---

3D Edges

- **In 2D**: edges can use different <u>texture</u> or <u>thickness</u>, to designate different types of edges, but the range of <u>shapes</u> is limited (wavy, straight, curved).
- **In 3D**: range of shapes is greater...
- Edges with a Z coordinate have different semantics
- Use the third dimension to show a variety of edges:
  - Lightning bolt
  - Helix
  - …
- Connect two graphs together

More 3D Techniques



- **Port** = connection point between edge and node.
  - 2D diagrams: ports are 1D entities
  - 3D diagrams: ports are 2D entities
    - →Rich semantics can be drawn on face of a port of connection of an edge to a 3D shape.
- **Nesting**: more degrees of freedom in depicting nested objects
- **Projections**: retrieve 2D diagrams

Don't confuse with projections in SDs and CDs

3D also provides a better grip on the recalcitrant "combined semantics" problem, that is how to tie together the semantics of diagrams of different kinds.

Outline

PART I: Static Behaviour
- **From UML to sets and spiders**
- **Spider diagrams - the details**
- **Constraint diagrams - the details**
- **Constraint trees**
- **Theoretical foundations**

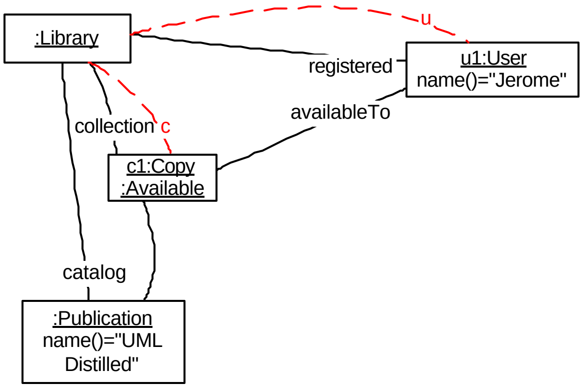
PART II: Dynamic Behaviour (3D modeling)
- **Why 3D**
- ➡ **3D filmstrips & 3D sequence diagrams**
- **Contract boxes**
- **Other 3D diagram ideas**

PART III: Tools
- **Extreme modeling - a tools manifesto**
- **Existing Tools**
- **Plans for the future**

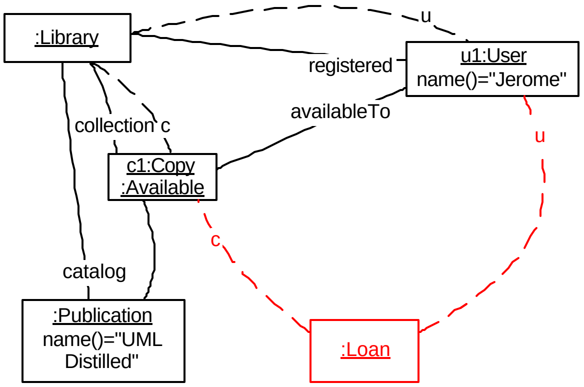
Filmstrip

- Sequence of snapshots of state
- Accompanied by a script
- Shows state change as script is played out

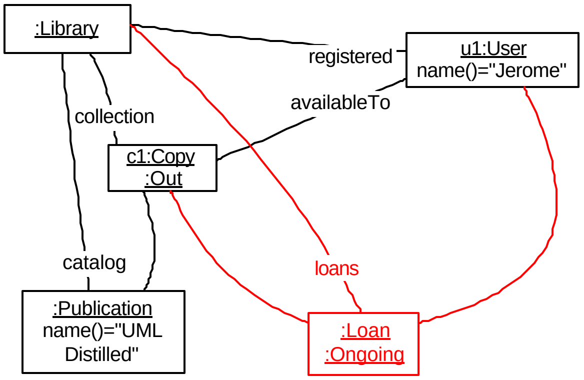
checkOut(c1,u1)[…



checkOut(c1,u1)[Loan(u,c,self)[…

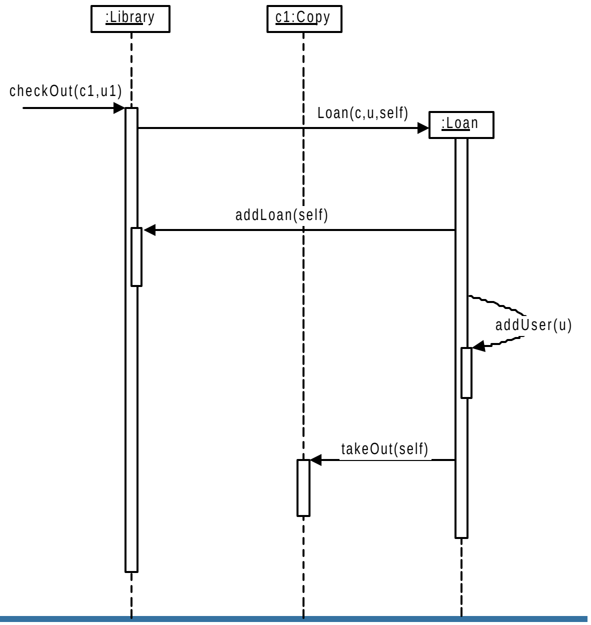


checkOut(c1,u1)[…]



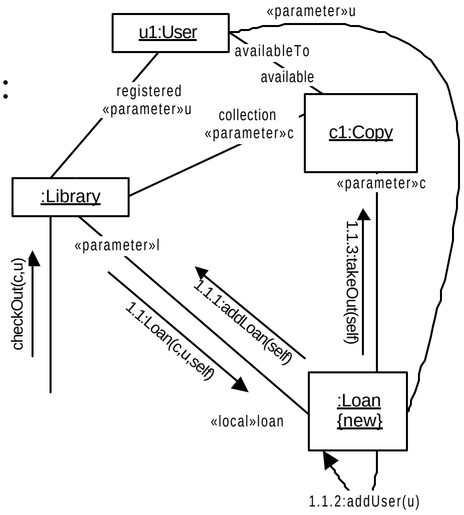
Sequence diagrams

- ✳ Life Lines
- ✳ Arrows: calls
- ✳ Box overlaps:
  - ⦿ Procedure nesting



Collaboration diagrams

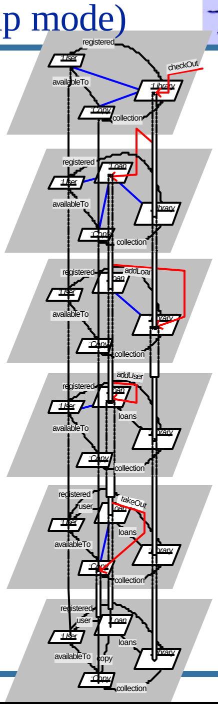
- ✳ Nodes: Objects
- ✳ Edges: Calls
- ✳ Dewey Numbering System:
  - ⦿ Call ordering
- ✳ Arrows:
  - ⦿ Invocation
  - ⦿ Return



Collaboration vs. Sequence Diagram

- Collaboration:
  - Relations: shown
  - Ordering: not shown
  - Participants in an operation:
    - →must be read from text
  - Nesting: not shown
  - Arrows: both time and relations
  - Change of state: partial
    - →shown by annotation (e.g. {new})
- Sequence:
  - Relations: not shown
  - Ordering: shown
  - Participants in an operation:
    - →Partial representation
  - Nesting: shown
  - Arrows:
    - →X-direction: message send
    - →Y-direction: time flow
  - Change of state: not shown

3D Sequence diagrams (filmstrip mode)



- Combine filmstrips with sequence diagrams
- 3D Effects:
  - Lightning bolt: message send
  - Blue connectors: parameters
  - Barrel nesting: call nesting
- Projection:
  - Sequence diagrams (instance mode)
  - Snapshots and filmstrips
  - Collaboration diagrams (instance mode)

Outline

PART I: Static Behaviour

- From UML to sets and spiders
- Spider diagrams - the details
- Constraint diagrams - the details
- Constraint trees
- Theoretical foundations

PART II: Dynamic Behaviour (3D modeling)

- Why 3D
- 3D filmstrips & 3D sequence diagrams
- ➡ Contract boxes
- Other 3D diagram ideas

PART III: Tools

- Extreme modeling - a tools manifesto
- Existing Tools
- Plans for the future

Contracts

- Example contract

  checkOut(c:Copy, u:User)

  pre

  --c is available for loan to u

  c.availableTo->includes(u)

  post

  --a new, ongoing loan is created which is linked to c and u

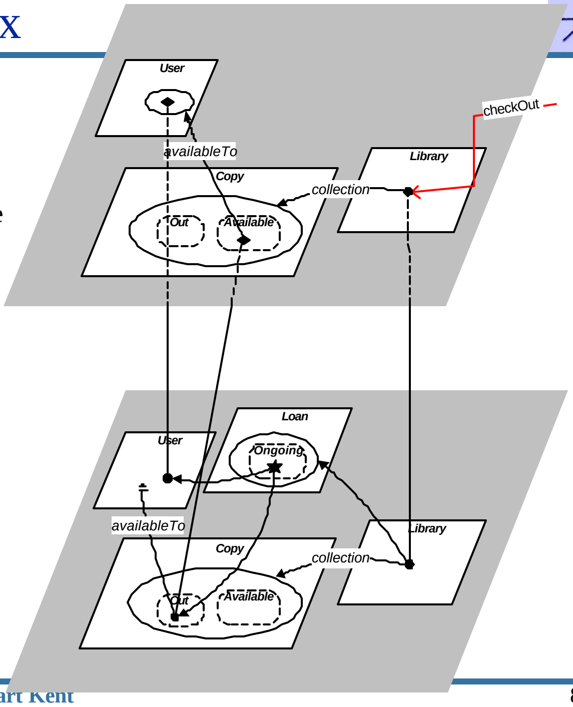
  --and c is marked as Out and unavailable for lending

  loans->exists(l | l.isNew & l.copy=c & l.user=u & l.ongoing)
  & c.availableTo->isEmpty & c.oclIsKindOf(Out)

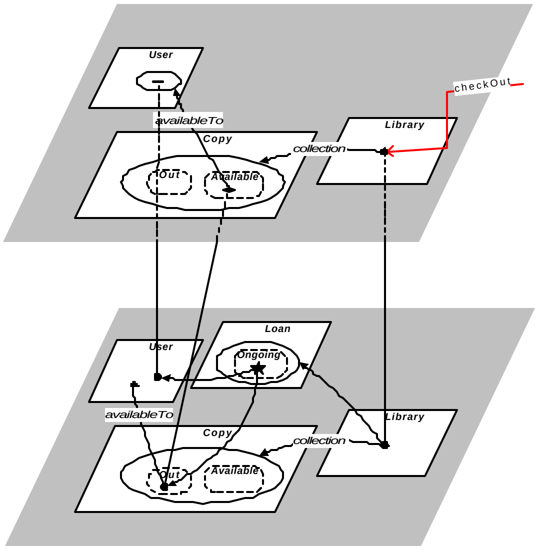
- Transitions on state diagram are abstractions of pre/post
  - they can be translated into pre/post

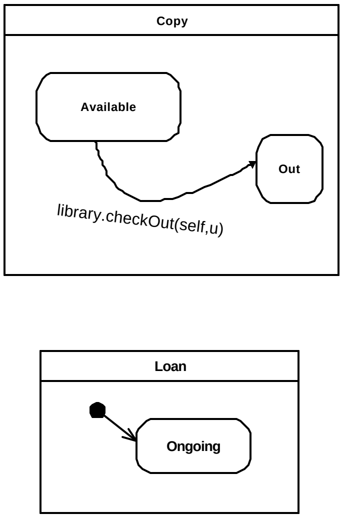
Contract box

- Show: pre- and post-conditions
- Main Idea:
  - Z-edges show life line
- Spider symbols:
  - ◆ = parameter
  - ★ = new object



State diagram: projection of contract box(es)





Outline

PART I: Static Behaviour

- From UML to sets and spiders
- Spider diagrams - the details
- Constraint diagrams - the details
- Constraint trees
- Theoretical foundations

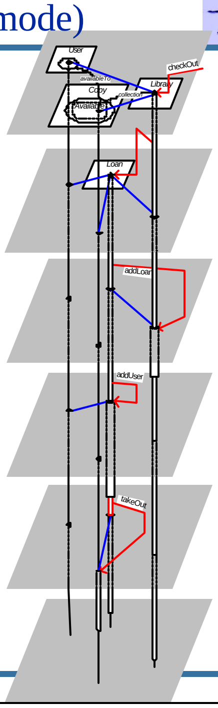
PART II: Dynamic Behaviour (3D modeling)

- Why 3D
- 3D filmstrips & 3D sequence diagrams
- Contract boxes
- ➡ Other 3D diagram ideas

PART III: Tools

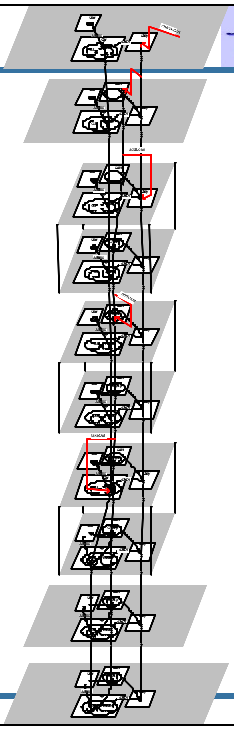
- Extreme modeling - a tools manifesto
- Existing Tools
- Plans for the future

3D Sequence Diagrams (spec. mode)



- Combine constraint diagrams with sequence diagrams
- 3D Effects:
  - Lightning bolt: message send
  - Blue connectors: parameters
  - Barrel nesting: call nesting
- Projection:
  - Sequence diagrams (spec. mode)
  - Constraint diagrams
  - Generalised collaboration diagrams
    - →a combination of collaboration & constraint diagrams

Nested box diagrams



- Provides an alternative approach to nesting
- More suitable for zooming in and out
  - exploding boxes!
- Boxes *are* contract boxes
  - changes are in relation to top of box, not previous plane
- Example in VRML
  - thanks to Jonathan Roberts from UKC

Outline

PART I: Static Behaviour
- **From UML to sets and spiders**
- **Spider diagrams - the details**
- **Constraint diagrams - the details**
- **Constraint trees**
- **Theoretical foundations**

PART II: Dynamic Behaviour (3D modeling)
- **Why 3D**
- **3D filmstrips & 3D sequence diagrams**
- **Contract boxes**
- **Other 3D diagram ideas**

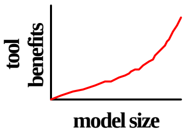
PART III: Tools
- ➡ **Extreme modeling - a tools manifesto**
- **Existing Tools**
- **Plans for the future**

Need - eXtreme modeling (1)

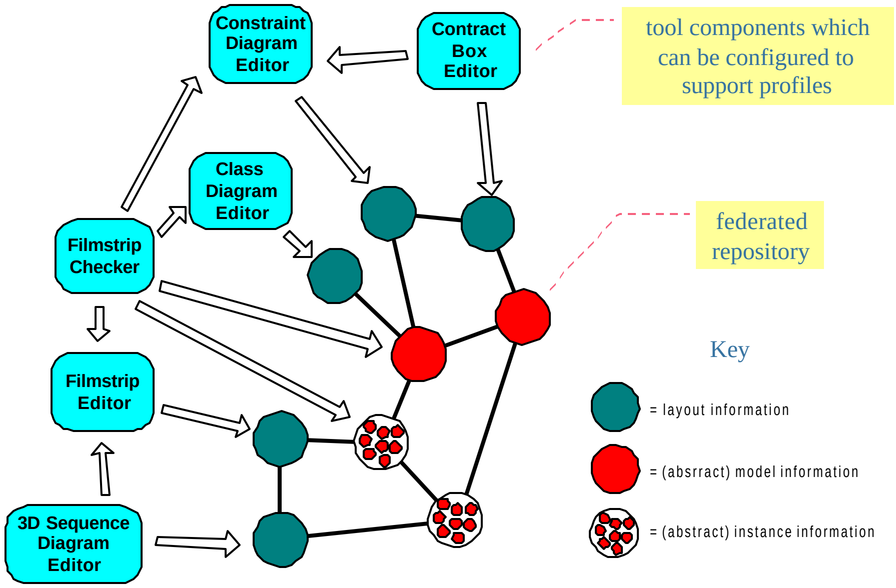
- eXtreme Programming (XP)
  - automated testing
  - testing supports refactoring, maintenance etc.
  - good tools essential
    - testing
    - editing
    - debugging
    - refactoring
    - organisation of code, version control, working in teams etc.

Need - eXtreme modeling (2)

- XM
  - testing models = setting up scenarios (filmstrips etc.) and keeping these in sync. with models
  - good tools essential
    - testing:
      - filmstrips, checking filmstrips against model, etc.
      - internal integrity of model itself
    - editing: good visual editors
    - debugging: give feedback through diagrams
    - refactoring: as for programs; tests can be be refactored as well
    - organisation:
      - model management (patterns, templates, packages, tests)
      - 3D to see whole picture



Tool Architecture



Constraint Diagram Editor

Contract Box Editor

tool components which can be configured to support profiles

Class Diagram Editor

Filmstrip Checker

federated repository

Filmstrip Editor

3D Sequence Diagram Editor

Key

● = layout information

● = (absrract) model information

● = (abstract) instance information

Outline

PART I: Static Behaviour

- From UML to sets and spiders
- Spider diagrams - the details
- Constraint diagrams - the details
- Constraint trees
- Theoretical foundations

PART II: Dynamic Behaviour (3D modeling)

- Why 3D
- 3D filmstrips & 3D sequence diagrams
- Contract boxes
- Other 3D diagram ideas

PART III: Tools

- Extreme modeling - a tools manifesto
- ➡ Existing Tools
- Plans for the future

Existing Tools

- Constraint Diagrams Editor
  - Developed by students @ Technion
  - Download via http://www.ukc.ac.uk/people/staff/sjhk/cds.html

- USE tool, BoldSoft tool (instance versus model, OCL)
- …

Outline

PART I: Static Behaviour
  - **From UML to sets and spiders**
  - **Spider diagrams - the details**
  - **Constraint diagrams - the details**
  - **Constraint trees**
  - **Theoretical foundations**

PART II: Dynamic Behaviour (3D modeling)
  - **Why 3D**
  - **3D filmstrips & 3D sequence diagrams**
  - **Contract boxes**
  - **Other 3D diagram ideas**

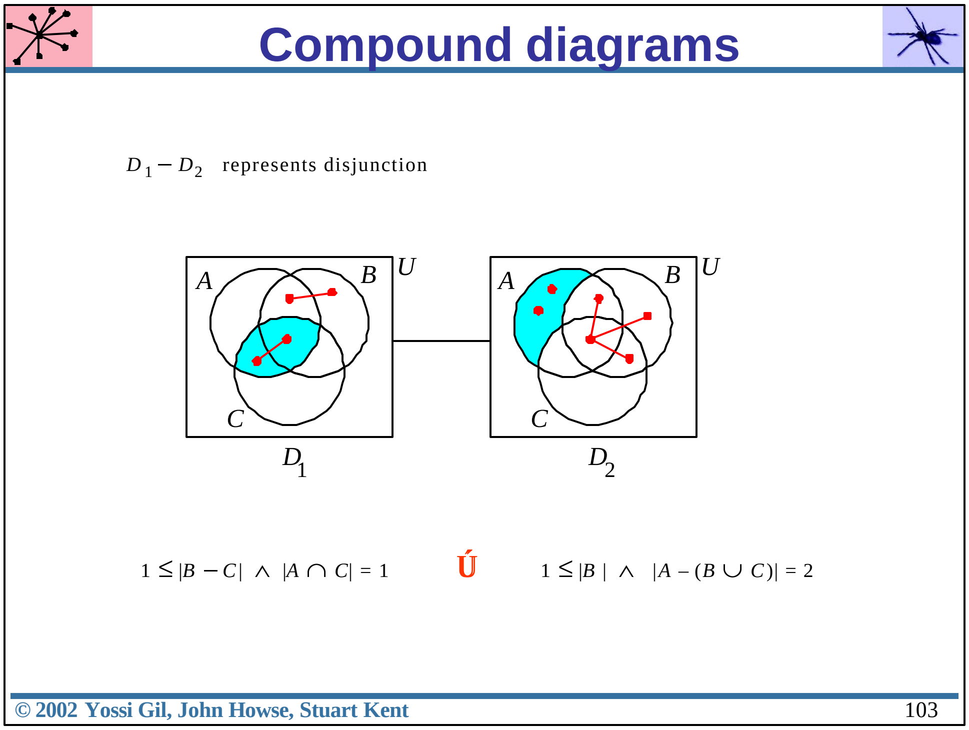
PART III: Tools
  - **Extreme modeling - a tools manifesto**
  - **Existing Tools**
  - ➡ **Plans for the future**

Editors - Where to next?

- Focus on visualization
- Filmstrip editor
  - using time
  - 3D sequence diagram - *overview with zoom*
- Contract box editor
- 3D sequence diagram editor (spec. mode)
  - *overview with zoom*
- Box diagram editor
  - *overview with zoom*
- Model management and pattern editors
  - some exciting work ahead

Semantics tools - Where to next?

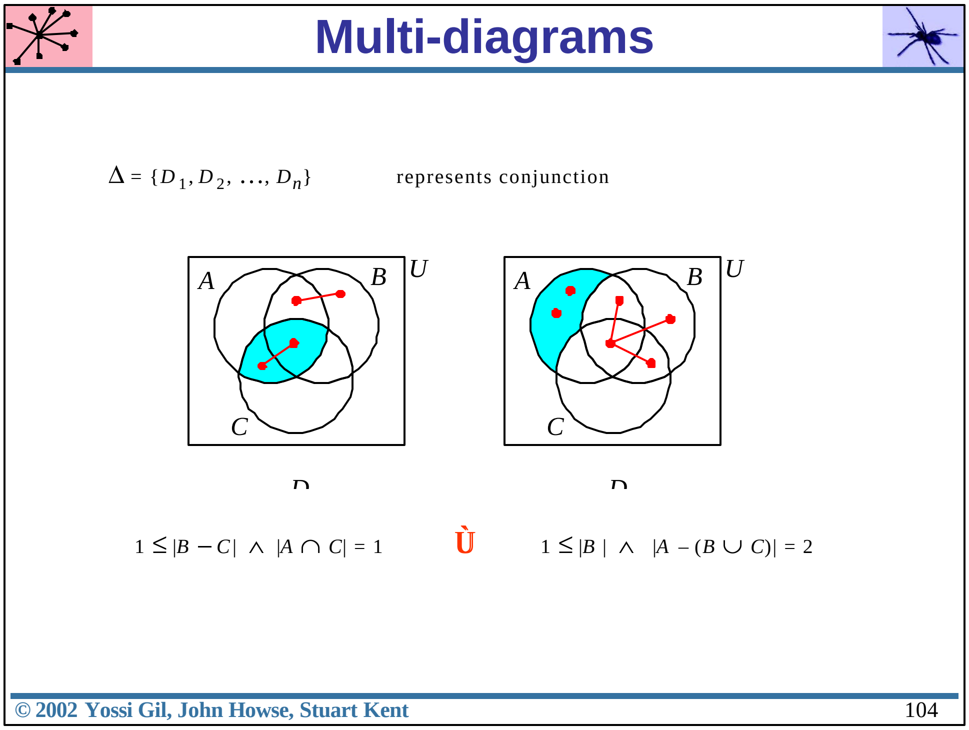
- wff and type checking
- model/instance consistency (USE tool etc.)
- model -> instance generation
- instance -> model generation
- consistency of models
  - model checking?
- theorem proving
  - main challenge is to allow reasoning and provide error information in notations being used to model - raw logic is not an option!
- etc.



Compound diagrams

$D_1 - D_2$ represents disjunction

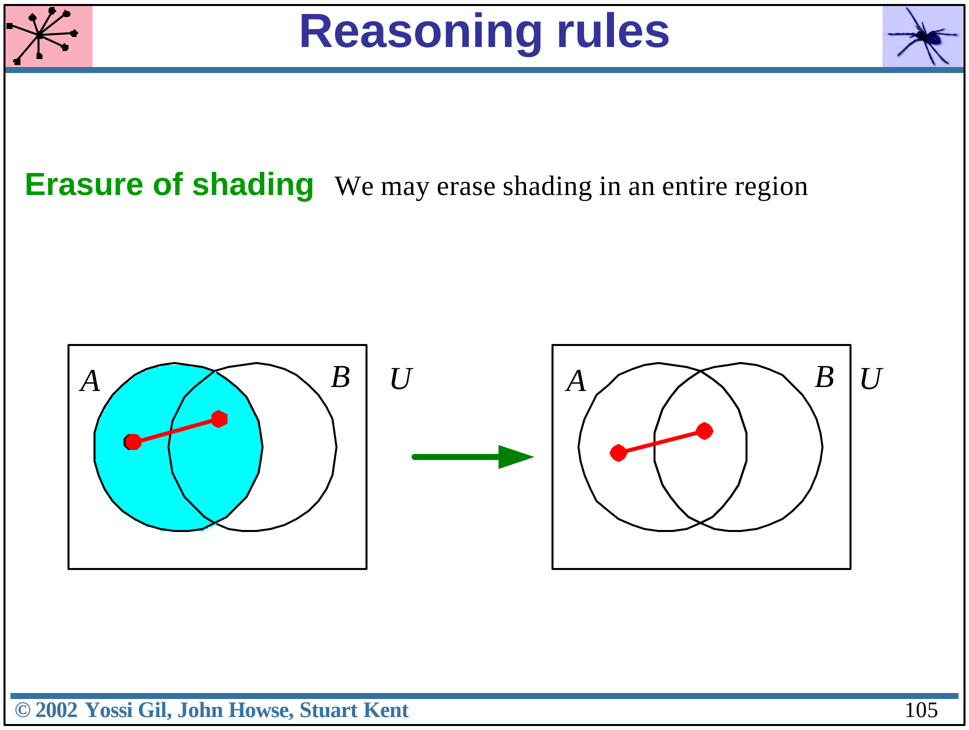
$1 \leq |B - C| \wedge |A \cap C| = 1$ Ú $1 \leq |B| \wedge |A - (B \cup C)| = 2$



Multi-diagrams

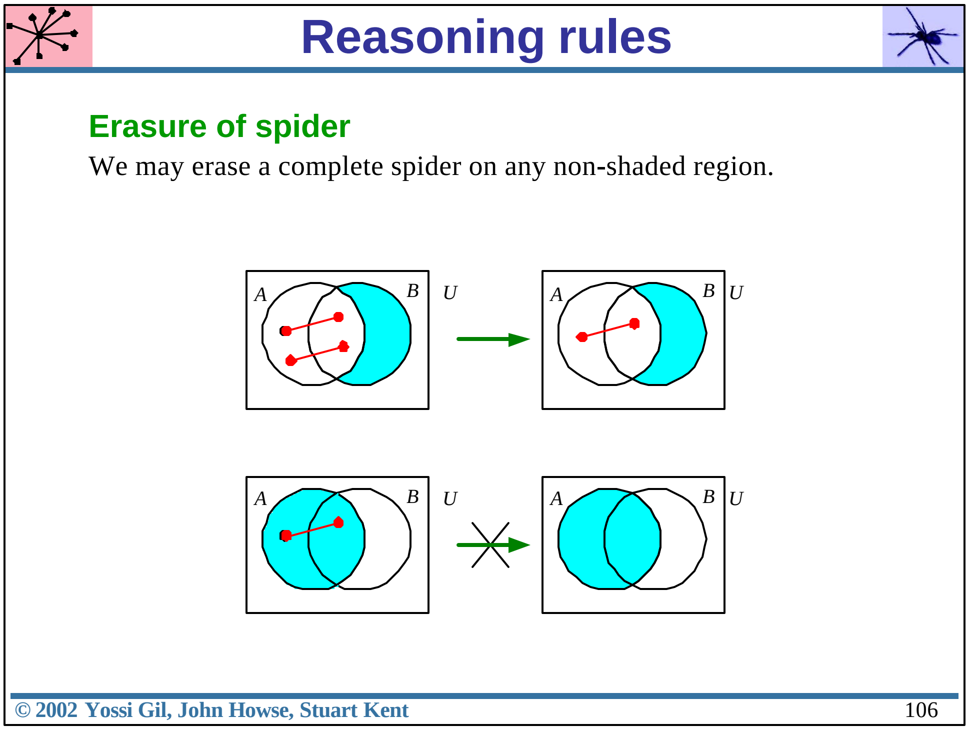
$\Delta = \{D_1, D_2, \ldots, D_n\}$ represents conjunction

$1 \leq |B - C| \wedge |A \cap C| = 1$ Ù $1 \leq |B| \wedge |A - (B \cup C)| = 2$



Reasoning rules

Erasure of shading We may erase shading in an entire region



Reasoning rules

Erasure of spider

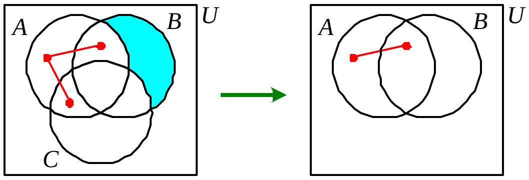
We may erase a complete spider on any non-shaded region.

Reasoning rules

Erasure of contour

We may erase a contour provided

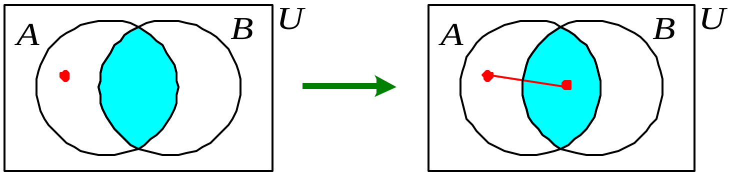
- remove ‘partial’ shading
- combine spider’s feet as necessary



Reasoning rules

Spreading feet

Given a spider *s*, draw a foot in any ‘new’ zone and connect it to *s*

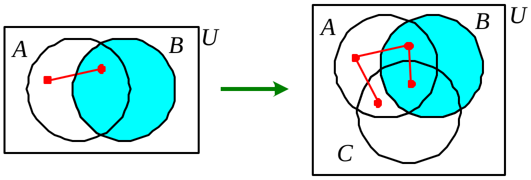


Reasoning rules

Introduce contour

Introduce new contour so that

- each zone bifurcates
- each spider’s foot bifurcates

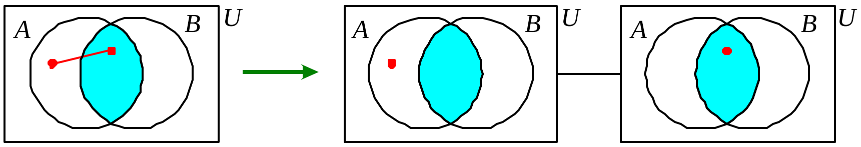


Reasoning rules

Splitting spiders

Spider *s* has *n*-zone habitat

® disjunction of *n* diagrams each containing a single-footed spider in one of the zones



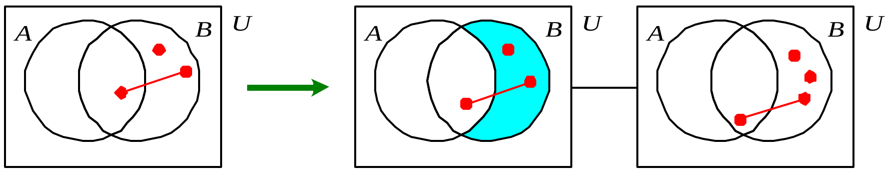
Reasoning rules

Excluded middle

Non-shaded zone z touched by n spiders

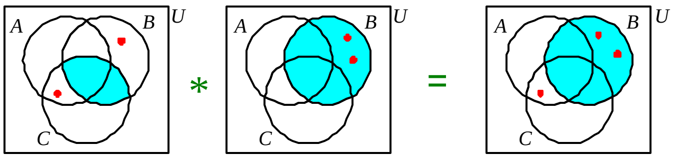
® disjunction of two unitary diagrams

- z shaded in one component touched by n spiders
- z not shaded in other component touched by $n + 1$ spiders



Combining diagrams

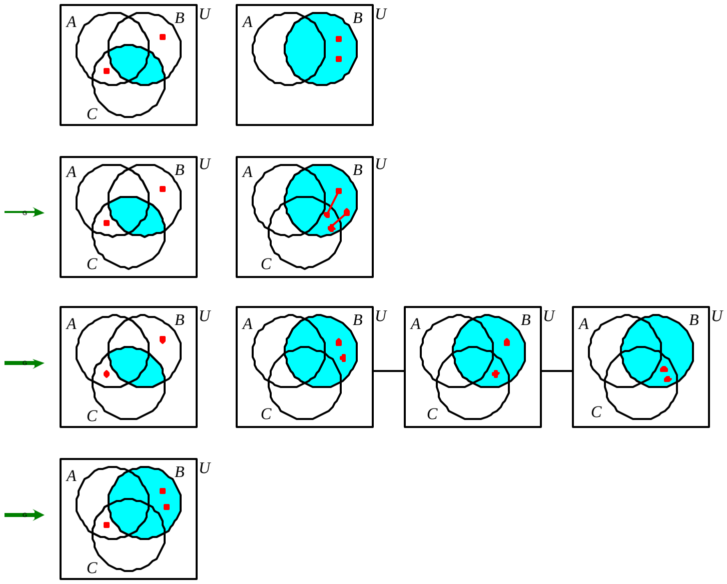
Unitary a-diagrams with same contour labels



D^1 D^2 $D^1 * D^2$

- a zone is shaded in $D^1 * D^2$ $\Leftrightarrow$ it is shaded in at least one of D^1 or D^2
- number of spiders in a zone of $D^1 * D^2$ equals the maximum number of spiders in the zone in D^1 and D^2

Combining diagrams



Reasoning rules

Inconsistency

Given an inconsistent multi-diagram Δ, we may replace Δ with any other multi-diagram.

Combining

Given a consistent multi-diagram $\Delta = \{D^1, D^2, \ldots, D^n\}$ we may replace Δ with the combined diagram $D^1 * D^2 * \ldots * D^n$.

Soundness and Completeness

Obtainability

$\Delta \vdash D$ D can be obtained from Δ by applying a sequence of transformations.

Consequence Relation

$\Delta \vDash D$ every compliant model for Δ is also a compliant model for D.

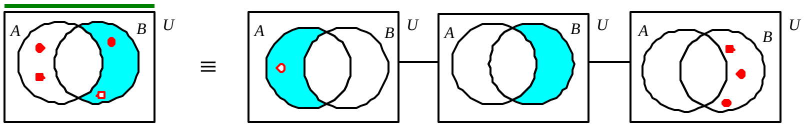
Soundness Theorem

If $\Delta \vdash D$ then $\Delta \vDash D$

Completeness Theorem

If $\Delta \vDash D,$ then $\Delta \vdash D$

Negation



Outline

PART I: Static Behaviour

- From UML to sets and spiders
- Spider diagrams - the details
- Constraint diagrams - the details
- Constraint trees
- Theoretical foundations

PART II: Dynamic Behaviour (3D modeling)

- Why 3D
- 3D filmstrips & 3D sequence diagrams
- Contract boxes
- Other 3D diagram ideas

PART III: Tools

- Extreme modeling - a tools manifesto
- Existing Tools
- Plans for the future

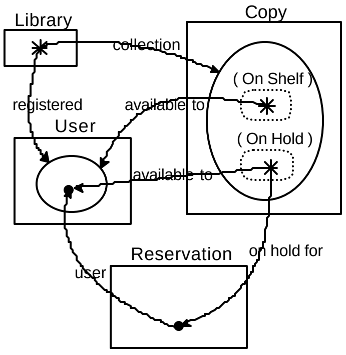
For more info see:

http://www.it.bton.ac.uk/Research/vmg

http://www.cs.ukc.ac.uk/constraintdiagrams

Exercise: solution...

"In any library, a copy which is on the shelf is available to all registered users; a copy which is on hold is available only to the user who made the reservation."



context l:Library **inv**:
l.collection->forAll(c | (c.oclIsKindOf(OnShelf) implies c.availableTo=l.registered)
and (c.oclIsKindOf(OnHold) implies c.onHoldFor.user=c.availableTo))