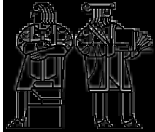


6.170 Lecture 16 Object Models



John Chapin
MIT EECS
3-8-2000

6-170

Contents

Motivation

why have design notations?
why object modelling?

Elements of an OM

sets, domains and subsets
relations & multiplicity
mutability

2 hard concepts today

OM records structure of the problem, not the program
No state in the objects, all state in the relations

© John Chapin/John Guttag

Spring 2000

2

6-170

Reasons for design notations

Communication is key

among people on a project
to yourself in the future

Code

communicates implementation ideas effectively
communicates design ideas poorly
expensive to create

Design notations

communicate design ideas effectively
help articulate ideas
find problems early
basis for division of labor
support rigorous testing and debugging

© John Chapin/John Guttag

Spring 2000

3

6-170

Types of design notations

Criteria

precise	unambiguous, can analyze
abstract	can suppress irrelevant details
expressive	can capture essence
lightweight	economical & easy to use

Two key notations

object models	structure of state
module dependency diagrams	code organization

Other notations

state machines	structure of events & state sequences
architectural sketches	process structure & communication paths

© John Chapin/John Guttag

Spring 2000

4

6-170

Object models (OMs)

Record state structure

what objects, what relationships among them
mistake in understanding state structure ==> bad bug

Two kinds of state structure

inherent structure of the problem
implementation structure of the program

In industry

UML for both problem and program

In this class

Alloy, an OM language developed at MIT
Used to record structure of the problem only

© John Chapin/John Guttag

Spring 2000

5

6-170

Problem state vs program state

Problem state

each student is registered
for 1 or more courses

Program state

store a pointer from each student
object to a list of course objects
store a pointer from each course object
to a list of student objects

the transmission is in
neutral, drive, or reverse

store the value 0,1, or 2 in the
transmission object
store a pointer in the engine object
to a behavior obj that moves the
car in the correct direction

© John Chapin/John Guttag

Spring 2000

6



Overview of the OM notation

An OM is a graph

nodes are sets of objects
 arcs are relations or subset relationships
 two kinds of markings: multiplicity & mutability
 describes what system states are possible

Details

sets
 relations
 multiplicity
 constraints
 snapshots
 mutability



Summary

OM gives

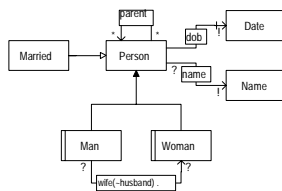
an invariant on the state space
 which states are permissible
 basic constraints about how state changes
 mutability markings

OM is

abstract but precise
 invaluable in early stages of design
 useful later for understanding runtime structures
 programming language independent



family tree example



sets, domains & subset

sets

a box represents a set of objects
 objects are structureless entities: no state is "contained"

subset

closed arrow denotes subset
 can read subset as "is-a": a *Man* is-a *Person*

domains

sets without supersets are called domains
 domains are disjoint: no object is both a *Person* and a *Date*

examples, with domains underlined

family: Person, Date, Name, Married, Man, Woman



disjoint subsets & partitions

shared subset arrows

say that subsets are disjoint
 no *Person* is both a *Man* and a *Woman*
 but *Person* may be both *Married* and a *Man*
 when arrowhead is filled, subsets are exclusive too
 every *Person* is a *Man* or a *Woman*

domains

are implicitly disjoint



relations

relations

arc with open arrow denotes a relation
 a relation is a mapping (ie, a set of pairs)
 relation $r: S \rightarrow T$ contains pairs (x, y) with x in S
 and y in T

examples

parents maps x to y when *Person* x has parent *Person* y
wife maps x to y when *Man* x has wife *Woman* y

transpose

the label $p(\sim q)$ introduces two relations; second is transpose of first
wife(\sim *husband*): wife maps x to y when husband maps y to x



notes about relations

non-disjoint sets

relations don't just map elements of sets with arrows
 wife maps objects in Married, even though arrow is from Man to Woman
 since Man and Married are not necessarily disjoint

what relations don't say

anything about references in objects
 anything about direction of navigation



multiplicity

how many?

instances of a set?
 instances mapped by a relation?

multiplicity markings

+ means one or more
 * means zero or more
 ! means exactly 1
 ? means zero or 1
 omission equivalent to *

which way round?

A * -> ! B means
 each A is mapped to one B
 each B is mapped to by zero or more A's



multiplicity examples

family

each Person has zero or more parents
 each Man has zero or one wife



constraints

some constraints

can't be expressed graphically
 just express in text, informally

examples

family
 a Man with a wife is Married
 x has wife y -> x.parents and y.parents are disjoint
 nobody is their own parent



snapshot semantics

an OM denotes

a set of snapshots, usually infinite

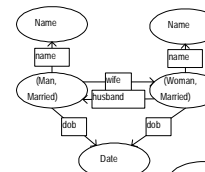
snapshot is graph

nodes are objects
 marked with names of sets they belong too
 (can omit superset when one of its subsets is included)
 arcs are pairs in a relation
 labelled with name of relation

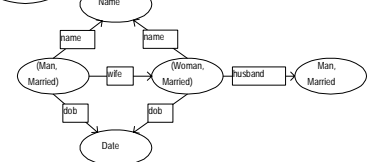


sample snapshots: family

good



bad





mutability

two useful kinds of constraint on changes

- no change to classification of an object
- no change to which objects an object maps to

static sets (shown with vertical stripe)

- a set S is static when
 - an existing object can't move in and out of the set

static relations (shown with hatch on line end)

- for relation r from A to B
 - left static (hatch on A end): each B, during its life, is mapped to by same A's
 - right static (hatch on B end): each A, during its life, maps to same B's



examples of mutability

family

- Man, Woman static (no sex change)
- Married not static (divorce)
- dob is right-static: can't change your date of birth



notes on design OMs

what's abstracted away

- localization of state
 - no instance vars, references etc
 - all state is global, in relations and subsets
- navigation issues
 - direction of relation does not imply navigability
- PL notions: subclasses vs. interfaces, methods, etc.

OMs are tricky!

- dob right static? not if system must allow corrections
- every Married Man has a wife? not if program allows incomplete info
- at most one parent who's a Man? not if step & adoption handled