6170 · laboratory in software engineering

lecture 12 · march 1, 1999 · object models

Daniel Jackson
Lab for Computer Science
MIT

contents

motivation

· why have design notations?
· why object modelling?

elements of an OM

· sets, domains and subsets
· relations & multiplicity
· mutability

from design to code

· how design & code OMs differ

why have design notations?

design stage
· help articulate ideas
· find problems early
· exploit idioms

implementation stage
· clear basis for delegation & division of labour
· touchstone for lower-levels of design

later stages
· hard to debug or maintain without design

what kind of design notations?

criteria

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

two key notations

· object models
    structure of state
· module dependency diagrams
    code organization, coupling

other notations

· state machines
    structure of events & state sequences: good for reactive systems
· architectural sketches
    process structure & communication paths

object models (OMs)

why

· state structure is major source of complexity
· helps bridge gap between problem and solution
    code state can be compared to problem state
· in OO languages, state structure *is* system structure

in industry

· OMs form basis of all current OO development methods
    UML, Catalysis, Fusion, Syntropy, OMT
· UML has been made an industrial standard
    see <http://www.rational.com/uml>

our notation

· Alloy, an OM language developed at MIT
· a clarified version of UML's "static structure notation"
· simpler than UML, but analyzable & more precise

exactly how do OMs help?

in design, OMs help you figure out

· what information system must retain

what state components are needed

how these fit together

· which constraints you can exploit

to simplify implementation

in coding, OMs tell you

· where to use containers

sets, tables, etc

· about sharing and mutability

when to watch for aliasing & rep exposure

design OM

a design OM is a graph

- nodes are sets of objects
- arcs are relations or subset relationships
- two kinds of markings: multiplicity & mutability

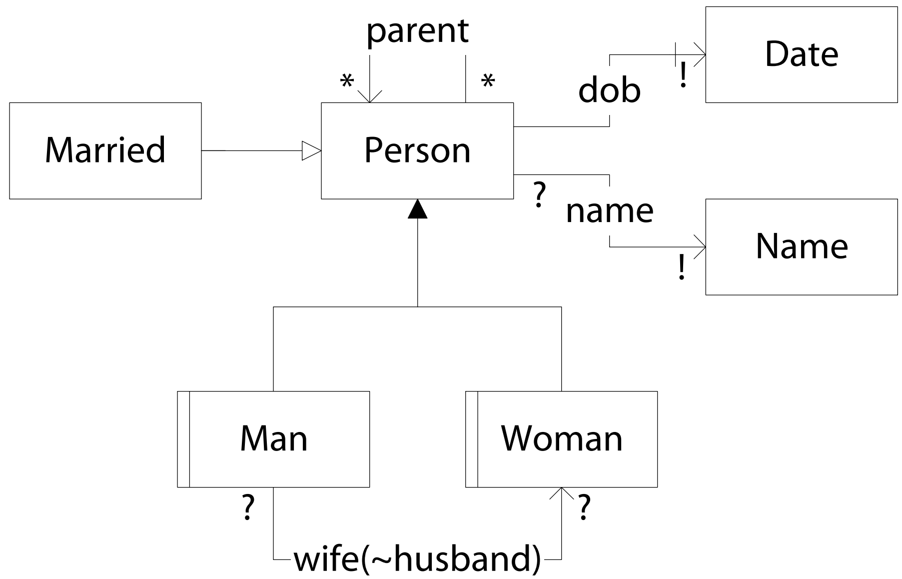
a design OM describes

- what system states are possible
- basic temporal properties (mutability)

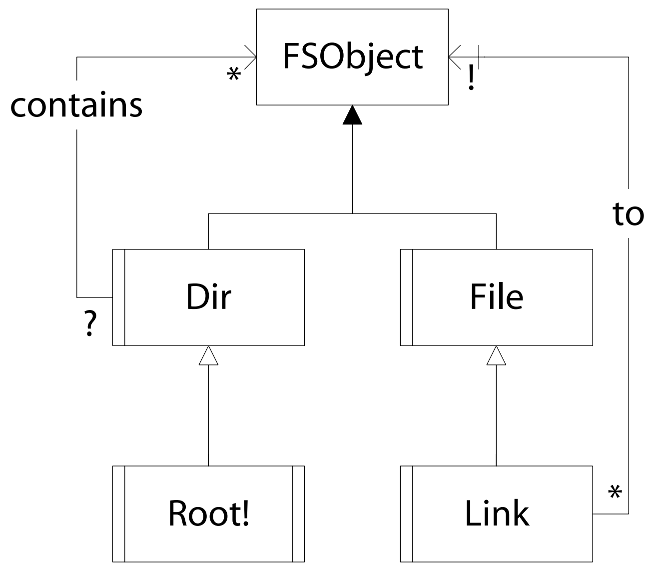
examples

- family tree: state is information about a family
- file system: state is structure of files, dirs & links
- employment database: state is employment & recommendation records

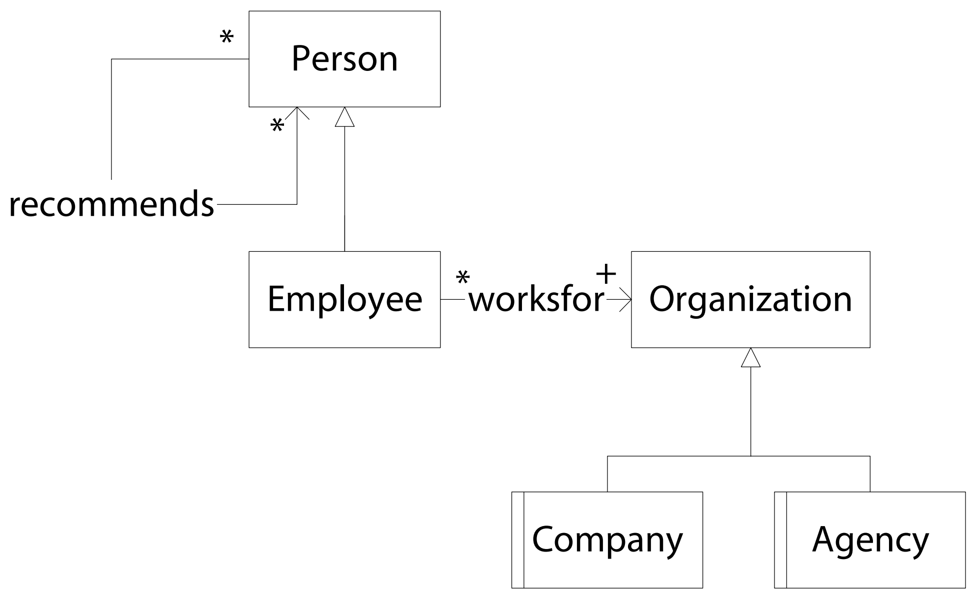
family tree example



file system example



employment database 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 underlines

- family: Person, Date, Name, Married, Man, Woman
- file system: FSObject, File, Dir, Link, Root
- employment DB: Person, Organization, Employee, Agency, Company

disjoint subsets & partitions

shared subset arrows

· say that subsets are disjoint

no Person is both a Man and a Woman

no FSObject is both a File and a Dir

no Organization is a Company and an Agency

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

every FSObject is a File or a Dir

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 -> 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

· *to* maps x to y when Link x points to the FSObject y

· *recommends* maps x to y when Person x recommends Person y

transpose

· the label p(~q) introduces two relations; second is transpose of first

· wife(~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

direction is just for semantics: a Dir *contains* FSObjects

but could do other way round: an FSObject *inside* a Dir

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

can use for sets too

- Root! is a set of Dirs with one element (ie, there's only one file system root)

multiplicity examples

family

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

file system

- each Link points to exactly one FSObject
- each Dir contains zero or more FSObjects

employment DB

- each Employee works for one or more Organizations

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
· employment database
  no Employee works for an Agency and a Company
  no self-recommendations
  every Employee has a recommender
· file system
  no Dir contains Root

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

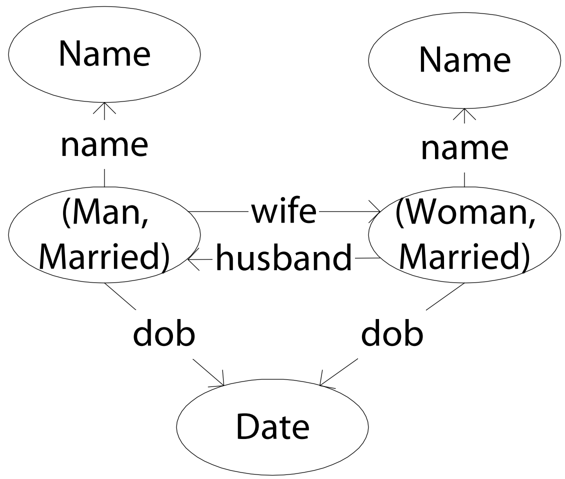
(can omit superset when one of its subsets is included)

· arcs are pairs in a relation

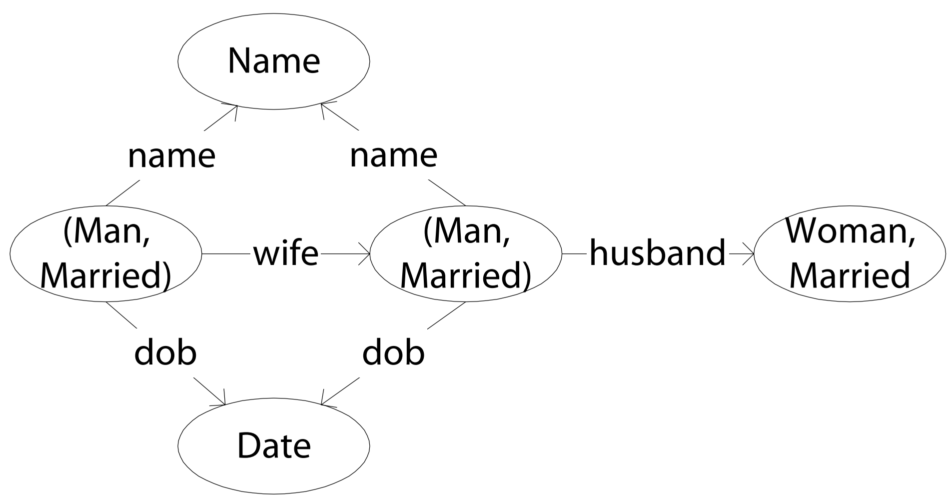
labelled with name of relation

sample snapshots: family

good

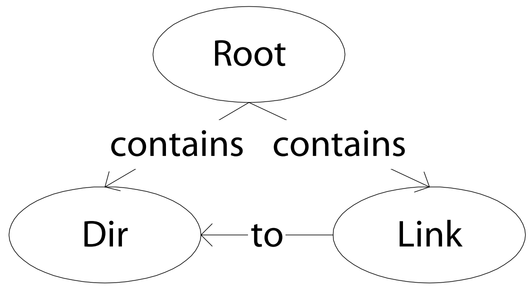


bad

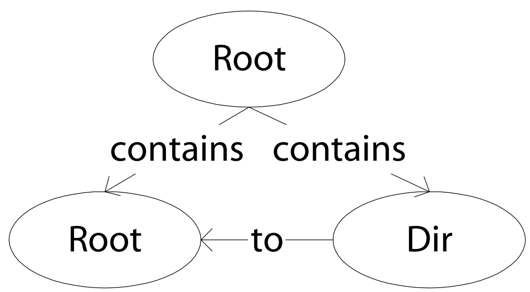


sample snapshots: file system

good

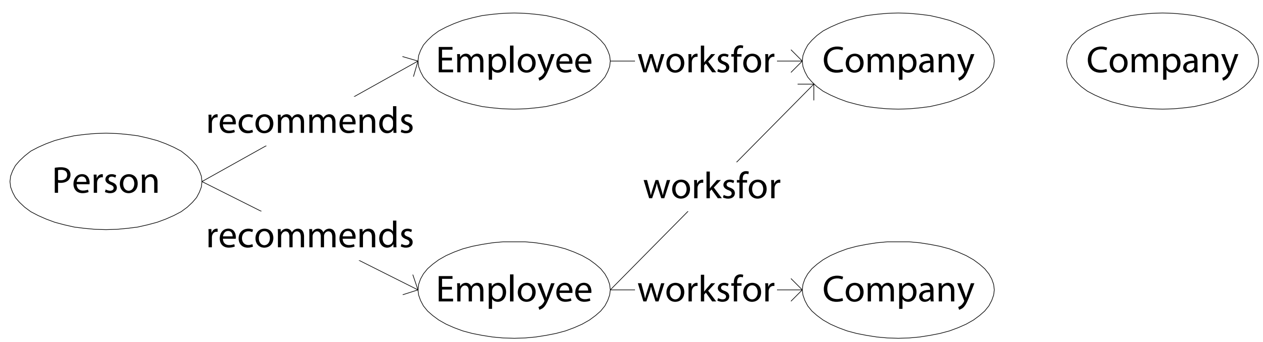


bad

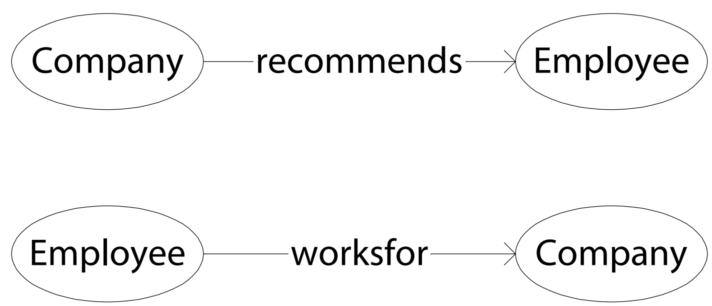


sample snapshots: employment db

good



bad



mutability

what it's about

· very useful to say what can change

· rather a subtle notion

  not often useful to say a set or relation doesn't change

  this prohibits new objects from coming into existence!

two useful kinds of constraint

· 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

file system

· File, Dir, Link are all static (a file can't become a directory)
· to is right-static (what a given link points to is fixed)

employment db

· Employee is not static (can get a job)
· Agency is static (govt agency can't become a Company)

fixed sets

very occasionally

· might want to describe a set that doesn't change
    then it's *fixed*

· shown with vertical stripes on both sides of box

examples

· in file system, *Root* is fixed: can't change which object is the root

· in card game program, *Suit* would be fixed

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

no notion of "root" object from which navigations start

· PL notions: subclasses vs. interfaces, methods, etc.

OMs are tricky!

· often embody careful judgments

· family example:

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

code OMs

same syntax, but read more into it

· sets are classes or interfaces
· subset is extends or implements
· relations are references

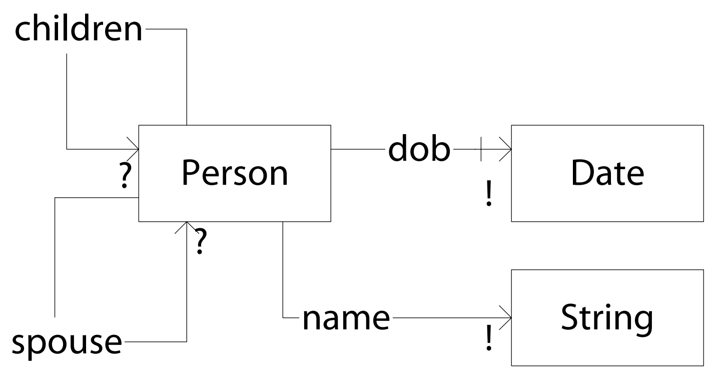
but many choices

· about how state is represented
· affect performance, ease of coding, flexibility

code OM for family (1)

representation choices

- wife, husband: as one field *spouse*
- parents: in transpose, as vector field *children*
- Married, Man, Woman: as boolean fields of Person
- top-most class has instance variable that holds Person at root of family tree



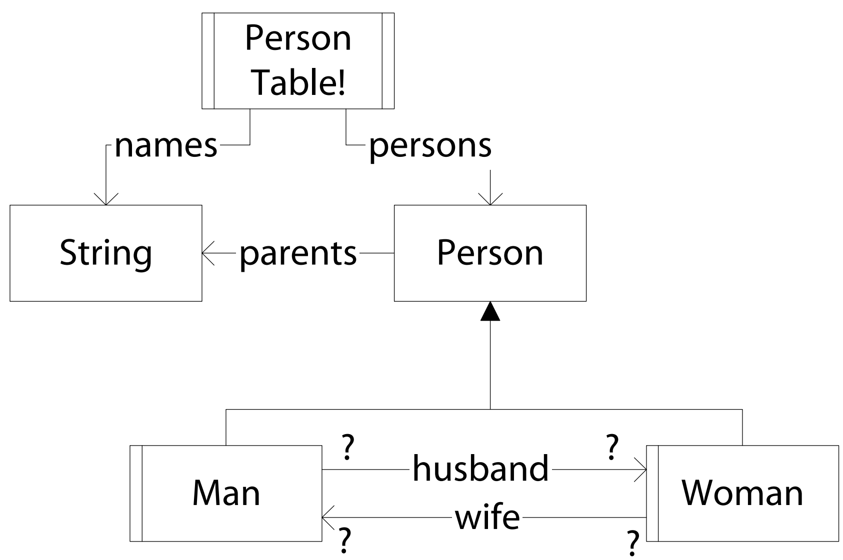
code OM for family (2)

representation choices

· name: as PersonTable

· dob/Date: as dd/mm/yy int fields

· parents: as array[string] field



from design OM to code OM

ways to represent a relation

· directly or in transpose (ie. reversed), as a field
· as a separate table object

ways to represent a set

· as a concrete or abstract class, or as an interface
· as a boolean field
· as a separate set object

other changes

· adding redundancy for extra paths

consequence of mutability

· a static set can become a *subtype*
· a set whose relations are all right-static can become *immutable*

polymorphism in code OMs

polymorphism

· some classes are polymorphic

· can be used in different ways

· eg, hashtable can be used for different relations

· add clarity to OM by representing with separate boxes

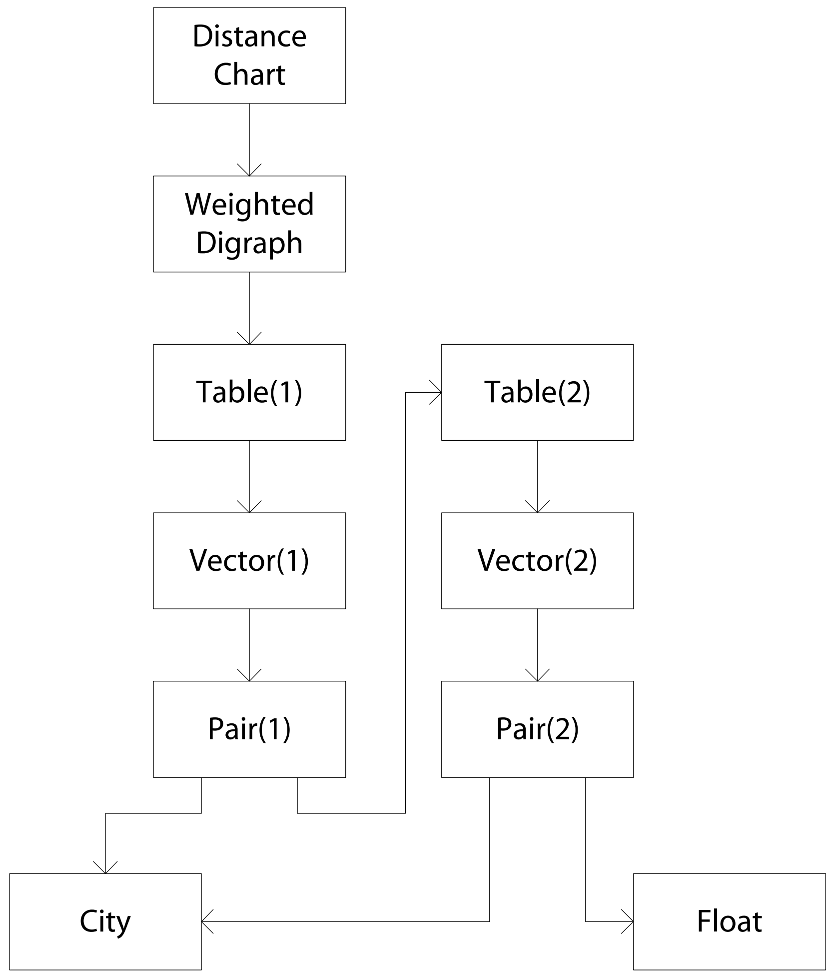
example

· in PS2/3

Table(1): from City to Table(2)

Table(2): from City to Float

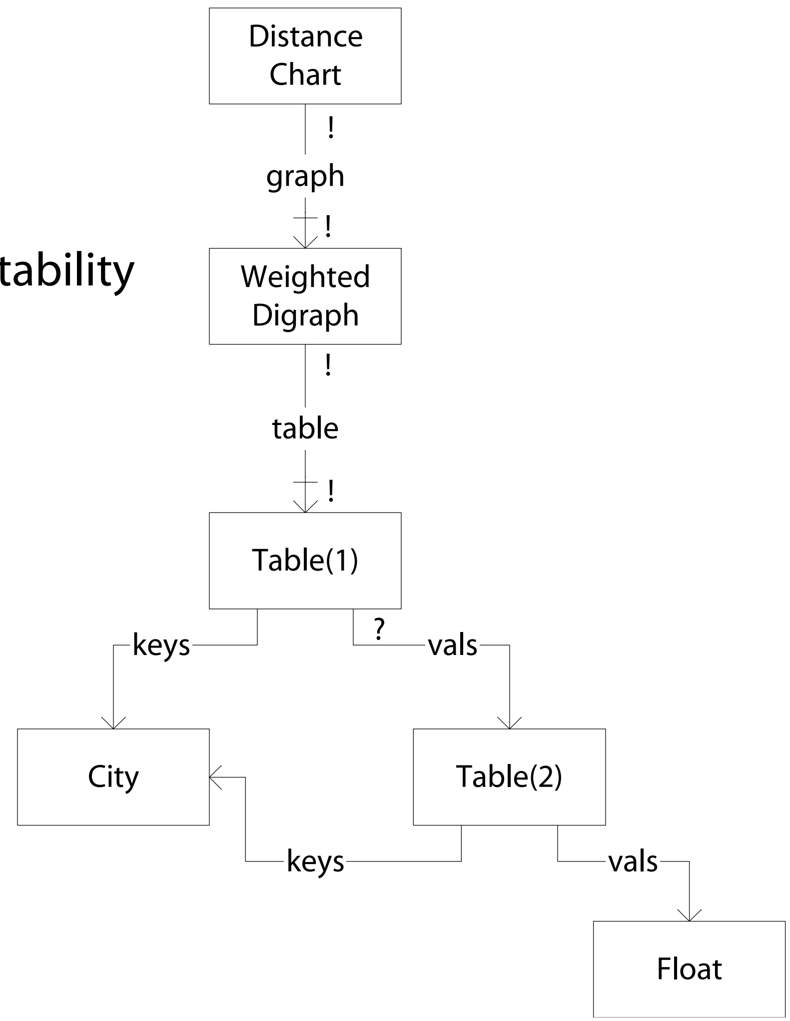
OM for PS2/3


Distance Chart
Weighted Digraph
Table(1)
Table(2)
Vector(1)
Vector(2)
Pair(1)
Pair(2)
City
Float

another OM for PS 2/3

notes

- same rep
- this OM elides some rep details
- also shows multiplicity and mutability



summary

OM gives

· an invariant on the state space

which states are permissible

state is like a venn diagram with relations

· 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