object models

Daniel Jackson
November 10, 2009
topics for today

a problem
• conceptual design of a photo organizer

a new paradigm
• computation over relational structures
• today, the abstract design level: object modelling
• determines, in particular, model part of MVC (see last lecture)

object modelling
• snapshot semantics
• basic notation: domain/range, multiplicity, classification
• some classic patterns
the problem
Problem

Design a photo cataloguing application

- Lightroom, iView MediaPro, iPhoto, Aperture, Picasa, etc
what kind of problem is this?

mostly about **conceptual design**

- what are the key concepts?
- how are they related to one another?
- what kinds of structures?

**good conceptual design leads to**

- straightforward path to implementation
- simplicity and flexibility in final product
why a new model?

why not use datatype productions?
  · tree-like structures only: no sharing
  · immutable types only

why not state machines?
  · our catalog is a state machine
  · but the problem lies in the structure of the state
  · our state machine notation assumed simple states

a new approach: object models
  · structure is a labelled graph
  · put another way: sets of objects + relations
the relational paradigm

computation is about
\- actions, states, transitions
\- functions, expressions, values
\- and now: **updates and queries on relations**

why is this useful?
\- conceptual modeling
\- relational databases
\- object-oriented programming*
\- semantic web, document object models, etc

basic OM notation
snapshots

a snapshot or object diagram
• shows a single instance of a structure
• nodes show objects
• arrows show relationships

example for photo organizer
• two kinds of nodes: collections & photos
• two kinds of arrows: for two containments

a relationship: C0 is subcollection of C1
a relationship: P0 in collection C2

more snapshots

how can we summarize this infinite set?
an object model

each box

• denotes a (maybe empty) set of objects
  - Photo: set of image files stored in the catalog
  - Collection: set of collections for classifying images

each arc

• denotes a relation, ie. set of links between objects
  - photos: Collection -> Photo
    c->p in photos means c includes p
  - subs: Collection -> Collection
    c1->c2 in subs means c1 is a subcollection of c2

note: objects have no internal structure!

• all structure is in the relations
enriching the notation

what's wrong with these snapshots?
  • how would we rule them out?

key idea: multiplicity
  • measure the in-degree and out-degree of each relation
multiplicity

multiplicity markings
· on ends of relation arc
· show relative counts

interpretation
· $R$ maps $m$ A’s to each B
· $R$ maps each A to $n$ B’s

marking/meaning
  + one or more
  * zero or more
  ! exactly one
  ? at most one
  omitted marking equivalent to *

© Daniel Jackson 2009
kinds of function

standard kinds of function

- easily expressed with multiplicities

1. \( A \xrightarrow{R} ? \rightarrow B \)  
   R is a function

2. \( A \xrightarrow{R} ! \rightarrow B \)  
   R is a total function

3. \( ? \xrightarrow{R} ! \rightarrow B \)  
   R is an injection

4. \( + \xrightarrow{R} ! \rightarrow B \)  
   R is a surjection

5. \( ! \xrightarrow{R} ! \rightarrow B \)  
   R is a bijection
we've added **naming**

- always an important and subtle issue
- is the multiplicity constraint desirable? necessary?
classifying objects

suppose we want to classify photos

• by file location: online, offline, missing
• by selection: selected, focus

![Diagram showing classification of photos by file location and selection. The diagram includes nodes for Photo, Selected, and Focus, with branches for Online, Offline, and Missing.]
classification syntax

can build a taxonomy of objects
• introduce subsets
• indicate which are disjoint
• and which exhaust the superset

\[
\begin{align*}
A & \subset A \\
B \cap C &= \emptyset \\
B \cup C &= A
\end{align*}
\]
relations on subsets

when placing a relation

\[\begin{align*}
\text{\textbullet} \text{ can place on subset} \\
\text{\textbullet} \text{ loose multiplicity is a hint}
\end{align*}\]
composite

a classic pattern
• hierarchical containment
• file systems, org charts, network domains, etc

you’ve seen this with datatypes
• technical differences though
• OM allows cycles (but often rule out)
• OM can say just one root
hotel locking
example: hotel locking

modelling physical, distributed state

state in OM need not represent

• a centralized store
• data stored in a computer
hotel locking

recodeable locks (since 1980)
• new guest gets a different key
• lock is ‘recoded’ to new key
• last guest can no longer enter

how does it work?
• locks are standalone, not wired
a recodable locking scheme

card has two keys
if first matches lock, recode with second

if second matches, just open
exercise

draw an object model
  • showing the essential state of hotel locking
  • state includes front desk, locks, keys held by guests

review
  • did you exploit multiplicities? keys are all about uniqueness
  • did you include only the sets and relations that are needed?
  • are your sets really sets, or are some of them ‘singleton placeholders’?
  • do all your sets and relations have a clear interpretation?
  • where are the various parts of the state stored physically?
  • which relations are modifiable?
a solution

- guest may occupy more than one room
- family members may have identical cards

Some subtleties

- $g \rightarrow r$ in $\text{occupies}$: guest $g$ has checked in for room $r$ but has not yet checked out
- $k$ in $\text{Issued}$: key $k$ has already been issued by front desk on some card: used to ensure that locks are always recoded with fresh keys
common errors

be wary of top-level singleton
\- Desk and Hotel not needed

relations represent state, not actions
\- so issues is suspect

need enough information in state to support application
\- has is not enough: need to know which key is first, second

scope of classification
\- classification of keys into first and second, is by card, not global
\- so need relation, not subsets to indicate the distinction
colour palettes
example: colour palettes

modelling the state of an application

• how colours are organized

esential idea

• elements are coloured
• can assign colour from palette
• gives consistent appearance

keynote

powerpoint
palette object models

three subtly different approaches

• think what happens when palette is modified
• hard vs. soft links: as in Unix

“Every problem in computer science can be solved by introducing another level of indirection”
-- David Wheeler
completing the organizer
issues to resolve

can collections hold photos and subcollections?
\textbullet{} decision: yes, so not Composite pattern

how are “all photos” in catalog represented?
\textbullet{} decision: introduce non-visible root collection

unique collection names?
\textbullet{} decision: file system style, so siblings have distinct names

do parents hold children’s photos?
\textbullet{} in logic: all \( c: \text{Collection} \mid c.\text{subs}.\text{photos} \text{ in } c.\text{photos} \) ?
\textbullet{} decision: use two relations instead

\( c.\text{inserted} \): the photos explicitly inserted into collection \( c \)
\( c.\text{photos} \): the photos in collection \( c \) implicitly and explicitly

invariant relates these: \( c.\text{photos} = c.\text{inserted} + c.\text{subs}.\text{photos} \)
final object model

additional constraints

• all collections reachable from root (implies acyclic)
  
  \[ \text{Collection in Root.} \ast \text{subs} \]

• implicit photos are inserted photos plus photos in subcollections
  
  \[ \text{all } c : \text{Collection} \mid c.\text{photos} = c.\text{inserted} + c.\text{subs}.\text{photos} \]

• names unique within parent
  
  \[ \text{all } c : \text{Collection} \mid \text{no } c_1, c_2 : c.\text{subs} \mid c_1 \neq c_2 \text{ and } c_1.\text{name} = c_2.\text{name} \]
modeling hints
hints

how to pick sets

› be as abstract as possible (thus Name, not String; SSN, not Number)
› but values to be compared must have same type (so Date, not Birthday)
› beware of singletons -- often a sign of code thinking

how to pick relations

› represent state, not actions (so atFloor: Elevator->Floor, not arrives)
› direction is semantic; doesn't constrain ‘navigation’

choosing names

› choose names that make interpretation clear
› include a glossary explaining what relations and sets mean
summary
principles

data before function

- before thinking about system function, think about data

an object model is an invariant

- meaning is set of structured states
- declared sets + subset relationships + relations between sets + multiplicities
- augment diagram with textual constraints (in Alloy, as above, or just English)

model objects are immutable

- all state kept in subsets and relations
- model objects have no ‘contents’
- important to keep coding options open