how to design a photo catalog

Daniel Jackson
April 9, 2007
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

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
  • another view: 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 modelling
- 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

example for photo organizer
  • in this case, two sets
    Photo (shown in beige)
    Collection (in grey)
  • and two relations
    photos: Collection -> Photo
    subs: Collection -> Collection

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

how can we summarize this infinite set?
an object model

each box
• denotes a (maybe empty) set of objects

each arc
• denotes a relation, i.e., set of links between objects

note
• objects have no internal structure!
• all structure is in the relations

exercise
• draw a snapshot that the OM rules out
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 2008
we’ve added **naming**

\1. always an important and subtle issue
\2. is the multiplicity constraint desirable? necessary?
classify objects

suppose we to classify photos

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

oval means singleton set
classification syntax

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

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

when placing a relation
• can place on subset
• loose multiplicity is a hint
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

some subtleties

• guest may occupy more than one room
• family members may have identical cards
colour palettes
example: colour palettes

modelling the state of an application
- how colours are organized

essential idea
- elements are coloured
- can assign colour from palette
- gives consistent appearance
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

“all photos”

• how’s the set of all photos in catalog represented?
• decision: introduce non-visible root collection

do parents hold children’s photos?

• can express the question in logic:
  \[
  \text{all } c: \text{Collection} \mid c\.\text{subs}.\text{photos} \in c\.\text{photos}
  \]
• consider removing a photo: does it disappear from parent collection too?
• decision: use two relations instead

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

  Collection in Root.*subs

• implicit photos are inserted photos plus photos in subcollections

  all c: Collection | c.photos = c.inserted + c.subs.photos

• names unique within parent

  all c: Collection | no c1, c2: c.subs | c1 != c2 and c1.name = c2.name
modelling 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

semantics
\cdot capture essential problem structure
\cdot model denotes set of states: it's an invariant (aha!)

syntax
\cdot box is set, arrow is relation
\cdot multiplicities, classification

expressiveness
\cdot diagrams have limited expressiveness
\cdot must augment with textual constraints
\cdot examples above are in Alloy; you can write in English
lecture exercises

simple file system
• construct an OM of a simple Unix-like file system
• sets should include directories, files, names
• find out the difference between soft and hard links and show both

formulas and CNF
• construct an OM for a boolean formula (with and, or, not)
• construct an OM for a formula in conjunctive normal form
• how do these compare to the datatype declarations we saw before?

world wide web
• construct an OM representing the basic structure of the web
• sets might be files, servers, URLs, links