how to design a photo catalog

Daniel Jackson
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

Screenshot of Adobe Photoshop Lightroom removed due to copyright restrictions.
In the Library view, you can select images to add or remove.
The left-hand sidebar includes Collections that you can define.

design a photo cataloging application

\* Lightroom, iView MediaPro, iPhoto, Aperture, Picasa, etc

© Daniel Jackson 2008
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

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
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, ie. 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 *
kinds of function

standard kinds of function
• easily expressed with multiplicities

- A \(\xrightarrow{R} ?\) B: R is a function
- A \(\xrightarrow{R} !\) B: R is a total function
- A \(\xrightarrow{? R} !\) B: R is an injection
- A \(\xrightarrow{+ R} !\) B: R is a surjection
- A \(\xrightarrow{!} !\) B: R is a bijection
we've added *naming*

- always an important and subtle issue
- is the multiplicity constraint desirable? necessary?
suppose we to classify photos

\begin{itemize}
  \item by file location: online, offline, missing
  \item by selection: selected, focus
\end{itemize}
Classification syntax can build a taxonomy of objects:

- Introduce subsets
- Indicate which are disjoint
- And which exhaust the superset

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

when placing a relation

- can place on subset
- loose multiplicity is a hint
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

recondable 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 holds Card

occupies Room

key ?

fst, snd !

k in Issued: key k has already been issued by front desk on some card: used to ensure that locks are always recoded with fresh keys

Guest g in occupies: guest g has checked in for room r but has not yet checked out

some subtleties

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

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

Screenshots of color schemes in the Keynote and PowerPoint presentation programs removed due to copyright restrictions.
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

© Daniel Jackson 2008
completing the organizer
issues to resolve

can collections hold photos and subcollections?
  · decision: yes, so not Composite pattern

how are “all photos” in catalog represented?
  · decision: introduce non-visible root collection

unique collection names?
  · decision: file system style, so siblings have distinct names

do parents hold children’s photos?
  · in logic: all c: Collection | c.subs.photos in c.photos ?
  · decision: use two relations instead
    c.inserted: the photos explicitly inserted into collection c
    c.photos: the photos in collection c implicitly and explicitly
    invariant relates these: c.photos = c.inserted + c.subs.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
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
\begin{itemize}
\item before thinking about system function, think about data
\end{itemize}

an object model is an invariant
\begin{itemize}
\item meaning is set of structured states
\item declared sets + subset relationships + relations between sets + multiplicities
\item augment diagram with textual constraints (in Alloy, as above, or just English)
\end{itemize}

model objects are immutable
\begin{itemize}
\item all state kept in subsets and relations
\item model objects have no ‘contents’
\item important to keep coding options open
\end{itemize}