March 4, 2002
6898: Advanced Topics in Software Design
MIT Lab for Computer Science
Daniel Jackson
What are analysis patterns?

simple models obvious only in retrospect
modeling = Business Process Reengineering
type models provide 'language of the business'
useful across business areas (health, finance, manufacturing)
ideas from one context useful in another
according to MF

focus on models themselves, not process
corporate information systems
based on experience applying object modeling to large

Martin Fowler, 1996

background
Plan

- What problems does this fit?
- Consider applications of pattern
- Investigate dynamic aspects too
- Express constraints formally
- Apply Alloy

"Referring to objects"

Look more deeply into one pattern
example: naming
objects & names

sig Object {}
sig Name {}
sig Scheme {objects: set Object, names: set Name, denotes: names -> objects}

denotes
names

Object

Name

objects & names

{ objects: names -> objects
  names: set Name,
  objects: set Object
  } Scheme

{} Name
{} Object

objects & names
static constraints

fun AllDenote (s: Scheme) {
    names in dom (s.denotes)
}

fun AllNamed (s: Scheme) {
    objects in ran (s.denotes)
}

fun NoAliases (s: Scheme) {
    inj (s.denotes)
}

fun Uniquedentifiers (s: Scheme) {
    func (s.denotes)
}
Play time…
Matching constraints to problems

- roadway/number
- Java object/heap address
- medical procedure/health plan treatment code
- MIT class/class number
- aircraft/flight number
- person/social security number
- machine/mac, machine/IP, machine/domain name
- problems
- AllNamed
- AllDenote
- NoAliases
- UniqueIdentifiers

Constraint matching constraints to problems
Questions

- Can names be recycled?
- If a name exists at two times, are its objects the same?
- If an object exists at two times, are its names the same?
- Can the name/object mapping change?

Dynamic Constraints
sample dynamic constraints

fun RetainNames(s, s': Scheme) { all o: s.objects & s'.objects | s.denotes.o = s'.denotes.o }

fun NamesSticky(s, s': Scheme) { all n: s.names & s'.names | all o: s.objects & s'.objects | n->o in s.denotes if fn->o in s'.denotes }

run NamesSticky(s, s': Scheme)

run RetainNames(s, s': Scheme)
A generic constraint

\begin{minipage}{\textwidth}
\begin{verbatim}
fun FixedFor (s, s': Scheme, ns: set Name, os: set Object) {
  let r = ns->os | s.denotes & r = s'.denotes & r

  \text{symmetrical in } s \text{ and } s' \text{ naming is fixed}

  for the names in ns and objects in os

  says

  \{
    let r = ns->os | s.denotes \& r = s'.denotes \& r
  \}

  fun FixedFor (s, s': Scheme, ns: set Name, os: set Object)
\end{verbatim}
\end{minipage}
varieties of dynamic constraint

fun Sticky (s, s' : Scheme) {
  FixedFor (s, s' : Name, Name, s.objects & s' objects)
}

fun NSticky (s, s' : Scheme) {
  FixedFor (s, s' : Name, Name, s' objects & s objects)
}

fun NSticky (s, s' : Scheme) {
  FixeddFor (s, s' : Name, Object, s' objects & s objects)
}

fun NSticky (s, s' : Scheme) {
  FixeddFor (s, s' : Name, Object, s' objects & s objects)
}

fun NamingFixed (s, s' : Scheme) {
  FixedFor (s, s' : Object, Name, s' objects & s objects)
}

fun NamingFixed (s, s' : Scheme) {
  FixedFor (s, s' : Object, Name, s' objects & s objects)
}

fun NamingFixed (s, s' : Scheme) {
  FixedFor (s, s' : Object, Name, s' objects & s objects)
}

fun NamingFixed (s, s' : Scheme) {
  FixedFor (s, s' : Object, Name, s' objects & s objects)
}

varieties of dynamic constraint
how can schemes change?

naming constraints across schemes?

corstraints

{ schemes: names -> schemes
  names: set Name,
  schemes: set Scheme,
} sig World

} sig SchemeName

} sig Scheme
A sample operation

\begin{align*}
\text{find need to refine classification by introducing new names}\\
\text{each name refers to set of objects} \\
\text{classification scheme} \\
\text{code refinement}
\end{align*}

\begin{align*}
\{ \text{AllNamed (s) AllNamed (s')} \\
\text{AllDenote (s) AllDenote (s')} \\
\text{NSticky (s) s'} \\
\text{same SameNames (s) - SameNames (s')} \\
\text{s.objects in s.objects} \\
\text{run Renames (s, s': Scheme)}
\}
\}

\begin{align*}
\{ \text{result} = 0 \mid \text{s.objects = s'.denotes.0} \} \\
\text{run SameNames (s: Scheme): Object -> Object}
\end{align*}
talking points

“...reflect a design approach...”

the problem, yet my techniques are object oriented, and hence

I try to develop very conceptual models that focus entirely on

“have to be”

analyses & design techniques may be rigorous but they don’t

“models are not right or wrong; they are more or less useful”

can see how to implement them

“conceptual patterns only useful to software engineers if they

"