analysis patterns

Daniel Jackson
MIT Lab for Computer Science
6898: Advanced Topics in Software Design
March 4, 2002

what are analysis patterns?

background

- Martin Fowler, 1996
- based on experience applying object modelling to large corporate information systems
- focus on models themselves, not process

according to MF

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

plan

look more deeply into one pattern
- “Referring to objects”

apply Alloy
- express constraints formally
- investigate dynamic aspects too

consider applications of pattern
- what problems does this fit?

example: naming

motivation

- systems need to deal with entities using names
- humans use names to refer to entities
- how names/entities are related can be tricky
- (anything object oriented here?)

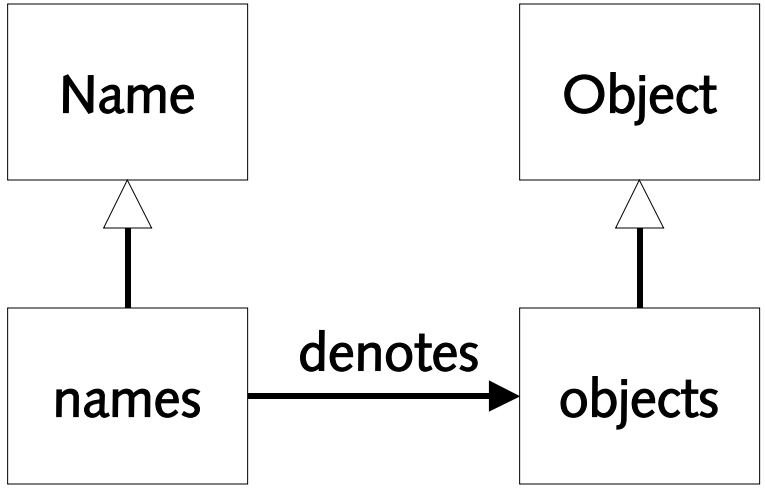
patterns

- Referring to Objects
- Identification Scheme
- Object Merge

objects & names

```
sig Object {}
sig Name {}

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



static constraints

```
fun UniqueIdentifiers (s: Scheme) {
    func (s.denotes)
    }

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

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

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

play time…

matching constraints to problems

constraints

- UniqueIdentifiers
- NoAliases
- AllDenote
- AllNamed

problems

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

dynamic constraints

questions

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

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 iff
           n->o in s'.denotes
    }
```

a generic constraint

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

says

- for the names in ns and objects in os
- naming is fixed

symmetrical in s and s’

- surprising?

varieties of dynamic constraint

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

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

fun NSticky (s, s': Scheme) {
    FixedFor (s,s',s.names & s'.names, Object)
    }

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

play time

identification scheme

```
sig SchemeName {}

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

constraints

- naming constraints across schemes?
- how can schemes change?

a sample operation

code refinement

- classification scheme
- each name refers to set of objects
- find need to refine classification by introducing new names

```
fun SameNames (s: Scheme): Object -> Object {
    result = {o,o': s.objects | s.denotes.o = s.denotes.o'}
    }

fun Refines (s, s': Scheme) {
    s.objects in s'.objects
    some SameNames (s) - SameNames (s')
    NSticky (s, s')
    AllDenote (s) AllDenote (s')
    AllNamed (s) AllNamed (s') }
```

talking points

“conceptual patterns only useful to software engineers if they can see how to implement them”

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

“analysis & design techniques may be rigorous but they don’t have to be”

“I try to develop very conceptual models that focus entirely on the problem, yet my techniques are object oriented, and hence reflect a design approach”