Formalizing Design Patterns: A Comprehensive Contract for Composite

Jason O. Hallstrom
School of Computing, CS Division
Clemson University

Neelam Soundarajan
Computer Science and Engineering
Ohio State University

This work is supported in part by the National Science Foundation
(CNS-0745846, DUE-0633506)
Responsibilities and Rewards

When using a pattern in an given application, designers are interested in two sets of properties

- **Responsibilities**
  The implementation requirements that must be satisfied to apply the pattern correctly

- **Rewards**
  The system properties that result by virtue of satisfying the implementation requirements

A comprehensive pattern formalism must capture both
The Formalization Challenge

The main challenge in formalizing patterns is striking the right balance between two competing objectives:

- **Precision**
  Implementation requirements and behavioral guarantees must be clear and unambiguous.

- **Flexibility**
  Pattern specifications must be customizable as appropriate to particular applications.

A comprehensive pattern formalism must satisfy both.
Pattern Contracts

Our approach to addressing these requirements relies on a multi-level contract framework

- **Pattern Contract**
  Captures the requirements and guarantees associated with *all* instances of a given pattern

- **Pattern Subcontract**
  Refines a pattern contract (or subcontract) to yield the specification of a sub-pattern or pattern implementation

Abstraction concepts are a key source of contract flexibility
Contract Structure

**Contract**

**Pattern Level**
- State abstraction concepts
- Constraints
- Interaction abstraction concepts
- Constraints
- Pattern instantiation conditions
- Pattern invariant

**Role Level**
- Enrollment / disenrollment conditions
- State requirements
- Behavioral requirements
  - Method state conditions
  - Method trace conditions
- Non-interference requirements

**Subcontract**

**Pattern Level**
- Concept definitions

**Role Level**
- Role maps
  - State maps
  - Method maps
Example: Composite Pattern (1/3)

pattern contract Composite {

  state abstraction concepts:
  Modified(Composite_α, Composite_β, Component_γ)
  Consistent (Component_δ, Component_ε)

  constraints:
  (↑α =↑β) ∧ ¬((↑δ =Leaf) ∧ (↑ε =Leaf)) ∧
  ∀c1,c1* ⊨ Composite, c2 ⊨ Component ::
  ((Consistent(c1,c2) ∧ ¬Modified(c1,c1*,c2))
  → Consistent(c1*,c2))

interaction abstraction concepts:
...omitted...

pattern invariant:
∀c1,c2 ⊨ Component :
(c1 ∈ players) ∧ (c2 ∈ players) ∧
(↑↑c1 =Component) ∧ (c2 ∈ c1.children):
((c2.parent = c1) ∧ Consistent(c1,c2))
Example: Composite Pattern (2/3)

```
role contract Component [1, abstract] {
  Component parent;
  void operation();
    pre: true
    post: (parent = #parent) ∧
          Consistent(parent, this)

  others:
    post: (parent = #parent) ∧
          (Consistent(parent, #this) → (Consistent(parent, this))
}
```

```
role contract Leaf [*] : Component {
  void operation();
    ...inherited from Component...

  others:
    ...inherited from Component...
}
```
Example: Composite Pattern (3/3)

1. role contract Composite [+]: Component {
   2. Set<Component> children;
   3. void add(Component c);
      pre: c \notin children
      post: (children= (#childrenU\{c\}))\land
         (c.parent=this)\land
         \forall oc \in Component :
         (oc \notin #children):
         \neg Modified(this, #this, oc)\land
         (|\tau.c.operation| = 1)
   4. void remove(Component c);
      pre: c \in children
      post: (children= (#children-\{c\}))\land
         \forall oc \in Component :
         (oc \in #children):
         \neg Modified(this, #this, oc)
   5. ...other child management methods omitted...

23. void operation();
24.   pre: ...inherited from Component...
25.   post: ...inherited from Component...\land
26.      (children= #children)\land
27.      \forall c \in Component :
28.         (c \in children):
29.            (Modified(this, #this, c)
30.            \implies (|\tau.c.operation| = 1))
31. others:
32.    ...inherited from Component...
33.    (children=children)\land
34.    \forall c \in Component :
35.       (c \in #children):
36.          \neg Modified(this, #this, c)
37. }
Questions?

Formalizing Design Patterns:
A Comprehensive Contract for Composite

Jason O. Hallstrom
School of Computing, CS Division
Clemson University

Neelam Soundarajan
Computer Science and Engineering
Ohio State University

This work is supported in part by the National Science Foundation
(CNS-0745846, DUE-0633506)