CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2010
Doc 21 Metadata and Active Object-Models
22 Apr 2010

References

Metadata and Active Object Models, Foote & Yoder, http://hillside.net/plop/plop98/final_submissions/P59.pdf

The User-Defined Product Framework, Johnson & Oakes, http://st-www.cs.illinois.edu/users/johnson/papers/udp/UDP.pdf

Daily Quote

First Law of Programming

Lowering quality lengthens development time

G. Weinberg

Metaprogramming

"Writing of computer programs that write or manipulate other programs (or themselves) as their data"

Wikipedia

Metaprogramming. (2010, April 11). In Wikipedia, The Free Encyclopedia. Retrieved 03:29, April 22, 2010, from http://en.wikipedia.org/w/index.php?title=Metaprogramming&oldid=355399719

Metadata & Active Object-Models

Complexity
Generality
Flexibility
Configurability

Pattern Language

Parameterization
Configuration
Expressions
Scripts
Dialogs
Tables
Specs
Message Routing
Context
Namespaces
Editor
Visual Builder
Dynamic Validation
History

Value Holder/Smart Values
Metaclass
Idempotence
Synthetic Code
Code As Data
Causal Connection
Boostrapping

Property
Smart Variables
Schema/Descriptor
Active Object-Model

Property Pattern

Attributes
Annotations
Dynamic Slots
Property List

Property

How do you allow individual objects to augment their state at runtime

Therefore, provide runtime mechanisms for accessing, altering, adding, and removing properties or attributes at runtime

What is a Property?

Key (Indicator) - name of the property

Value - the value of the property

Descriptor - information about property

display name, type, constraints

default value, accesor functions, etc

Indicates how to downcast

Used by tools

Java Fake Example

```
class Example {
    HashMap<String,Object> properties = new Hashmap<String, Object>();

    public void setProperty(String name, Object value) {
        properties.put(name, value);
    }

    public Object getProperty(String name) {
        return properties.get(name);
    }

    public boolean hasProperty(String name) {
        return properties.containsKey(name);
    }
```

Some Property methods

```
void addProperty(Indicator name, Descriptor aboutProperty, Object value );
void removeProperty(Indicator name);
boolean hasProperty(Indicator name);
void setProperty(Indicator name, Object value);
Object getProperty(Indicator name);

Decriptor getDescriptor(Indicator name);
Descriptor[] getDescriptors();
Object[] propertyList();
```

Java Properties Class

```
Properties defaults = new Properties();
defaults.put("a", "one");
defaults.put("b", 'two");

Properties test = new Properties(defaults);
test.put("c", "three");
test.put("a", "override a default");

test.get("a");
test.get("b");
test.get("d");
```

Consequences

You avoid a proliferation of subclasses

Fields may be added to individual instances

Fields may be added and removed at runtime

You may iterate across the fields

Metainformation is available to facilitate editing and debugging

Properties can graduate to first-class fields as an application evolves.

Consequences

Syntax is more cumbersome in the absence of reflective support

Property access code is more complex that that for real fields

Reflective mechanisms, where they are available, can be slower

Idiomatic implementations, when reflective support is not available, are also slow

Access to heterogeneous collections can be expensive

A field must be added to all objects, while only a few ever use it

The User-Defined Product Framework

The User-Defined Product Framework

Let users

Construct a complex business object from existing components

Define a new kind of component without programming

Insurance managers can invent a new policy rider

Framework developed at ITT Hartford

Used to represent insurance policies

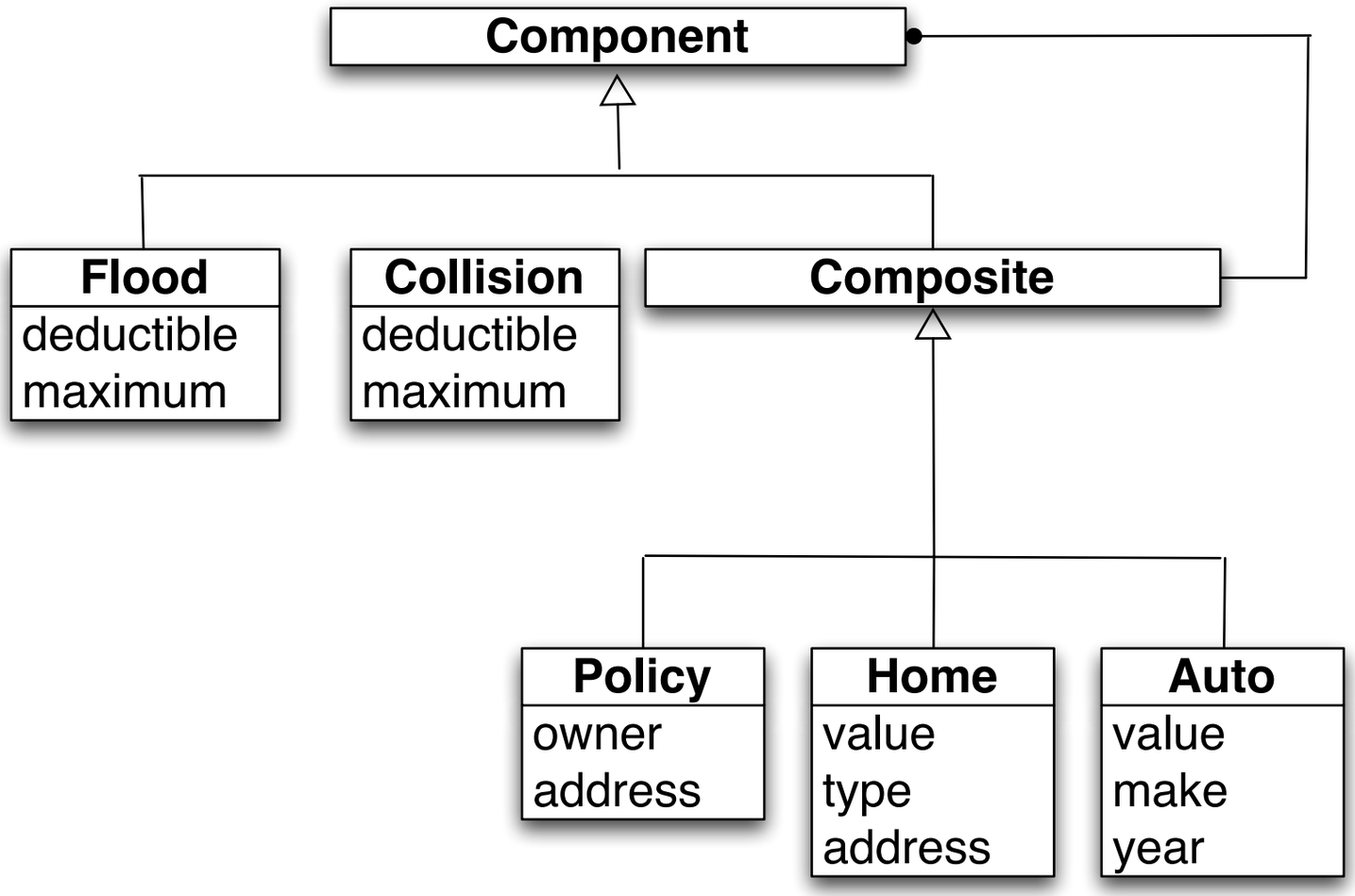
Problem

Which is the best way to combine features, multiple inheritance or composition?

Need 10,000 classes to get all the combinations needed

Use object composition to combine features instead of multiple inheritance.

Solution - Composition



Policy
Auto
Collision
Auto
Home
Flood

Problem

Design is still complex and hard to use

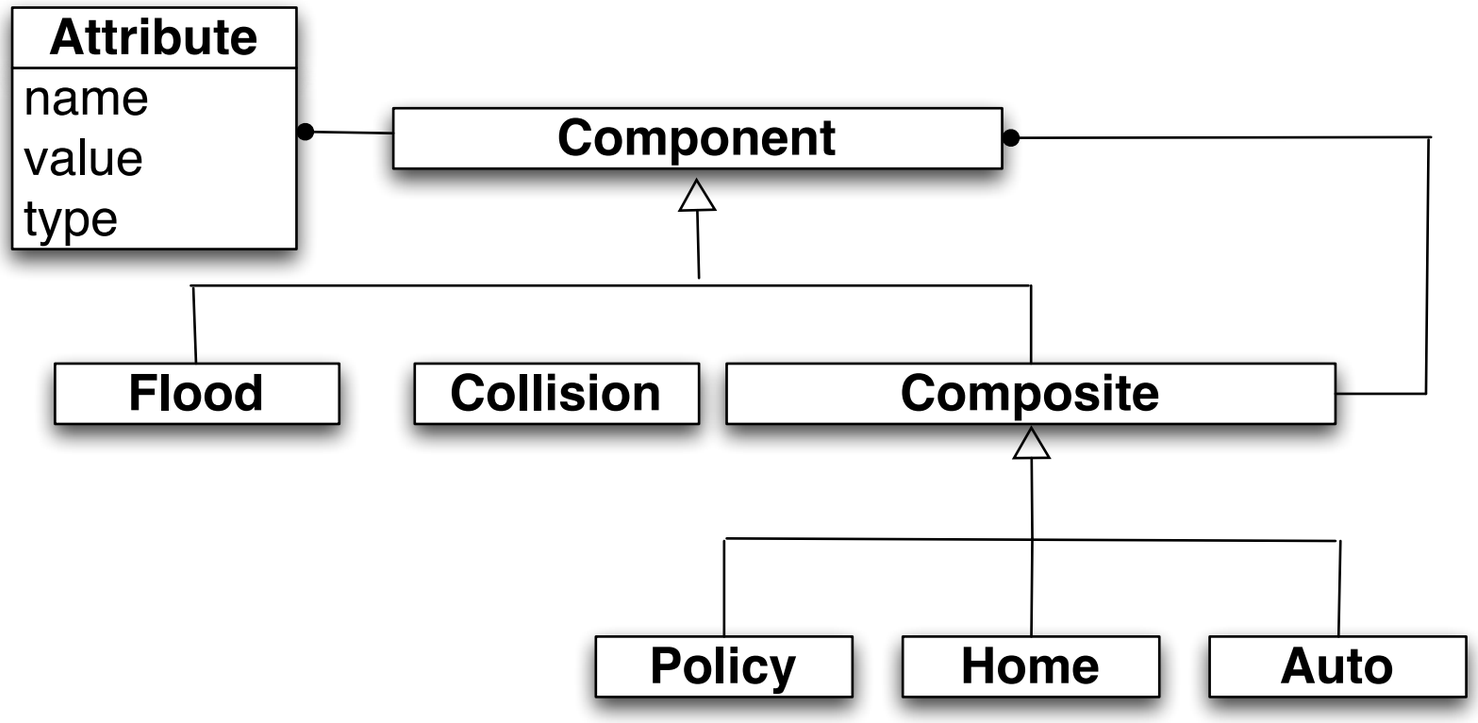
a huge number of Component classes

adding a feature means making a new one

Component has too many subclasses.
How can we keep from having to subclass Component?

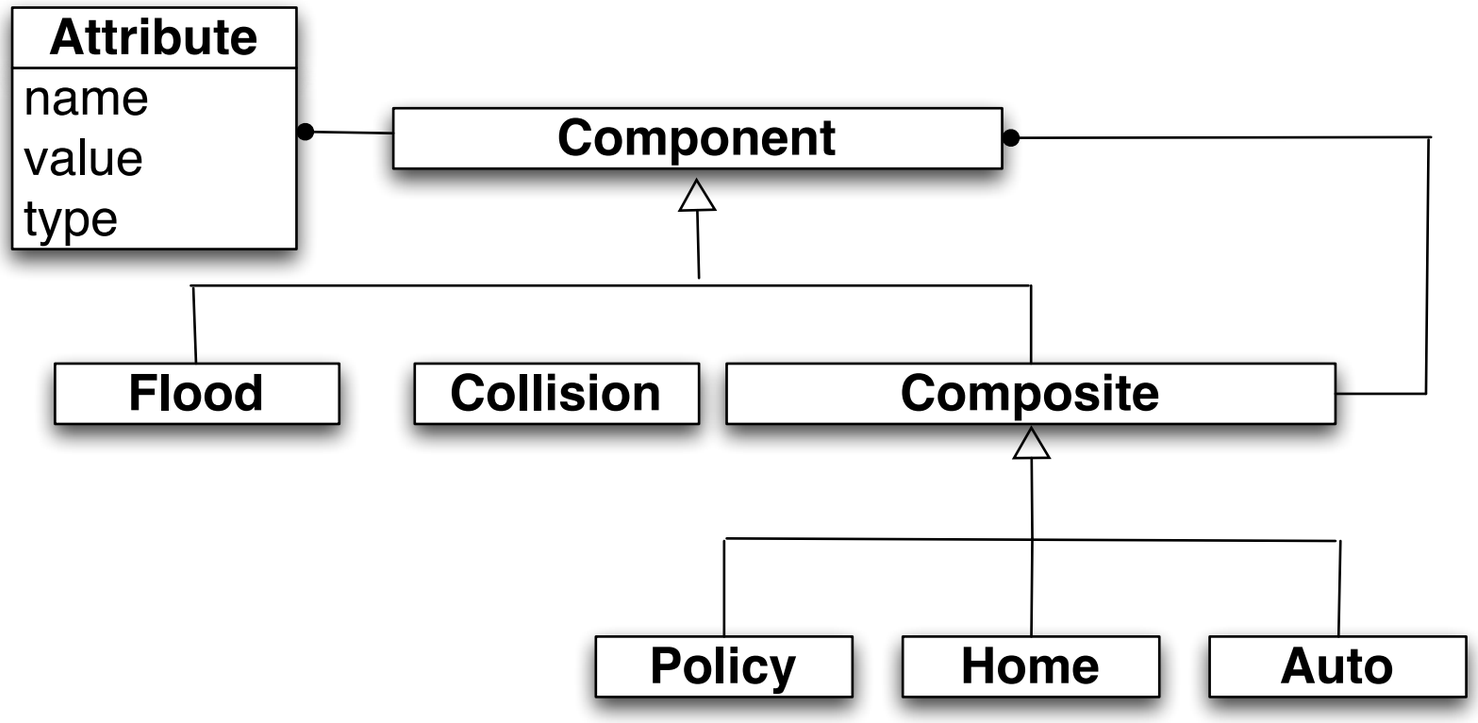
Solution - Properties (Variable State)

Eliminate the need to subclass to add instance variables by storing attributes in a dictionary instead of directly in an instance variable.



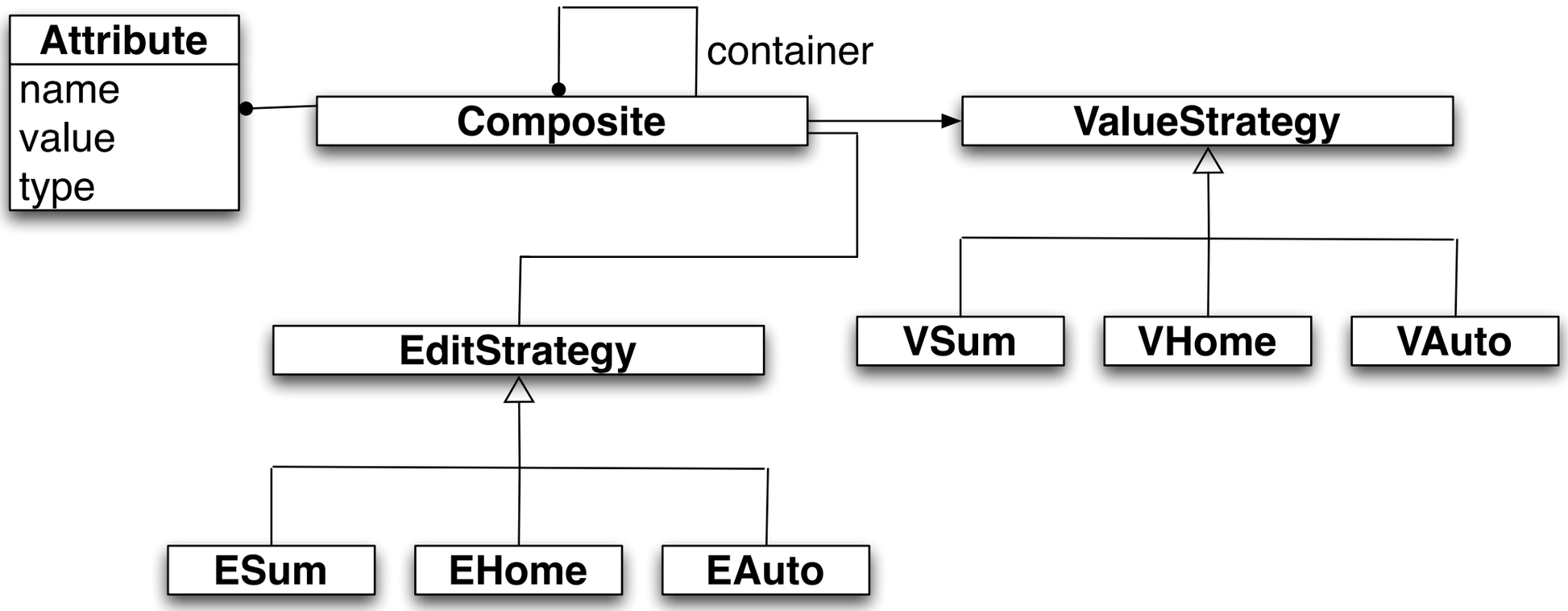
Problem

Still have subclasses for behavoir



Solution - Strategy

Make a Strategy for each method of Component that varies in its subclasses.



Problem

But now instead of lots of component subclasses

We have lots of Strategy subclasses

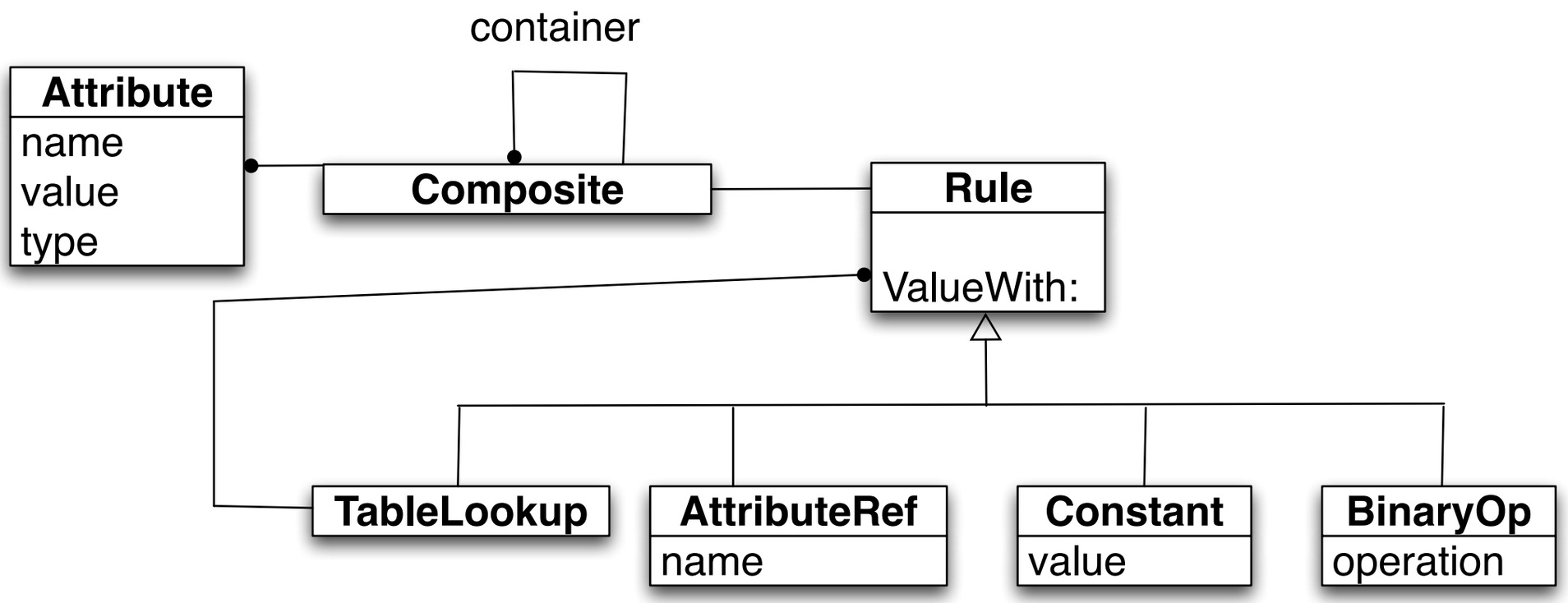
Solution - Interpreter

Create small language for the behaviors of strategies

Value strategies use:

- arithmetic expressions
- table look up
- if statements

Solution - Interpreter



Rules

- read/write attributes
- pre-formula
  - evaluated before component's children
- post-formula
  - evaluated after component's children

Problem

Component subclass replaced with attributes & rules

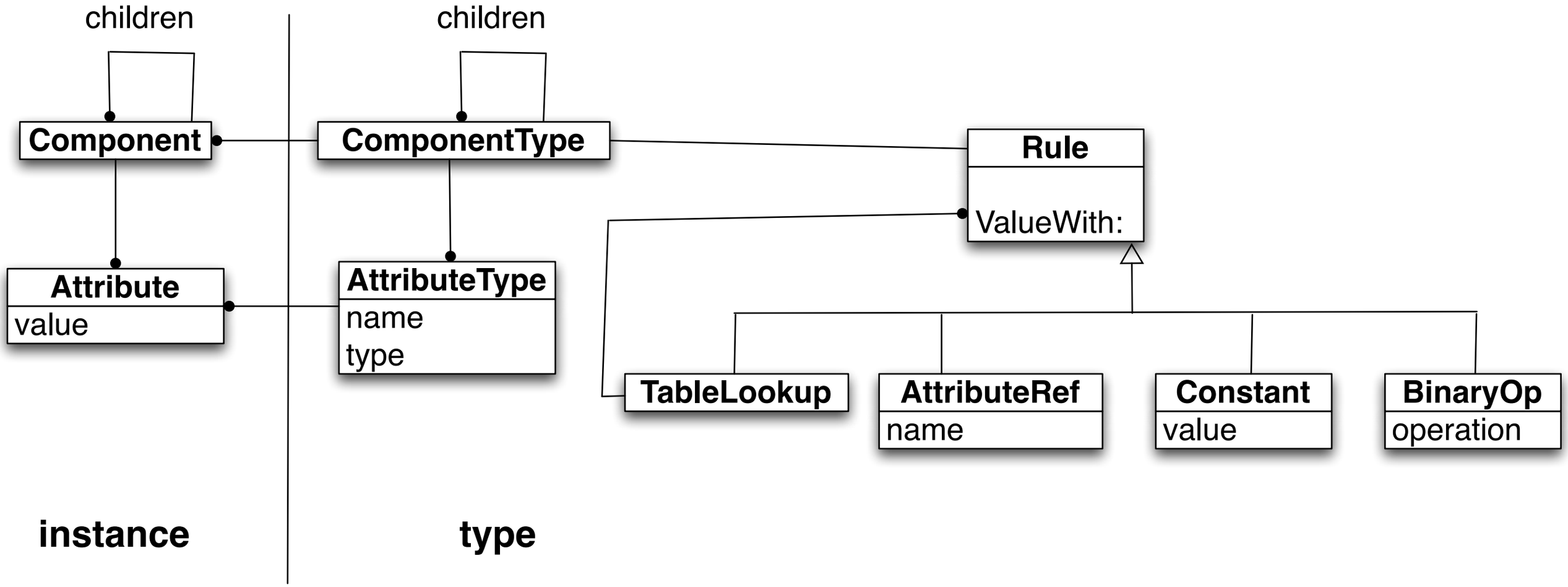
Each "component" instance has own copy of rules - duplication

Without classes to categorize components
    harder to understand code

How can you eliminate duplication in a component system and represent categories of similar components when all components have the same class?

Solution - Type Object

Use the Type Object pattern; i.e. make objects that represent the common features of a category of components, and let each component know its type and access those features by delegating to the type



Talk about editors and database of types

Problem

Sometimes attributes need to have rules

Life insurance over $1,000,000 has special data and rules

Most attributes don't have rules so why add that option to all attributes

Solution - Decorator

AttributeDecorator - adds rule to attribute

Paper also covers several other patterns used

Smart Variable

Issue

Often when a field changes some action is required

Most of the time accessor methods handle this fine

Examples when not

Debugger - watch points
Simulations
Real-time tracking of business

Actions tied to State Change

Dependent Notification
Persistence
Distribution
Caching
Constraint Satisfaction
Synchronization

Schema

Descriptor
Map
Database Scheme
Layout

Schema

How do you avoid hard-wiring the layouts of structures into your code?
How do you describe the layout of a structure, object, or database row?

Therefore, make a schema or map describing your data structures available at runtime

Participants

Schema - collection of descriptors

Descriptor - describe layout of element

May contain attributes

display name, type, default value

Subject - objects being mapped by schema

Grapples - map between symbolic name to actual object

Attributes

Examples

Database Object-Relational mapping

Hibernate, Spring, Active Record in Ruby on Rails

GUI Builders

JavaBeans - Descriptor