Composition Mechanisms in OO Languages: Traits and ClassBoxes

Roel Wuyts
ULB Séminaire

25/03/04
4 talks

“Logic Meta Programming and Language Symbiosis”

“A Data-centric Approach to Composing Embedded, Real-time Software Components”

“Unanticipated Integration of Development Tools and the StarBrowser”

“Composition Mechanisms in OO Languages: Traits and ClassBoxes”
Roadmap

- Context
- Traits
  - Problem, model, validation
- Classboxes
  - Problem, model, validation
- Validation
- Conclusion
Traits

- Modularity of methods within classes
  - Solve shortcomings with inheritance

- Compose methods in groups
- Compose groups to form classes
- Advantage: level between single methods and complete classes
Multiple inheritance problem

```
return Circle::draw();
return SyncDrawing::draw();
return Polygon::draw();
return SyncDrawing::draw();
```

```
acquireLock();
result := super.draw();
releaseLock();
return result;
```
Mixins

Circle
draw ==

return radius() == other.radius() && center() == other.center();

Circle + MBorder

draw ==

return super == other && borderSize() == other.borderSize();

... + MColor

rgb ==

return super == other && rgb() == other.rgb();

BorderedColoredCircle

==
Inheritance Problems

- **Single:** not expressive enough
- **Multiple:**
  - Complex solution → Hard to understand
  - Explicit class references → Fragile
  - Sometimes requires code duplication
- **Mixins:**
  - Implicit conflict resolution → Surprises!
  - Composite entity is not in full control
    - Dispersal of glue code
    - Fragile hierarchies
What we want:

```
acquireLock();
result := super.draw();
releaseLock();
return result;
```
What are Traits?

- Traits are parameterized behaviors
  - Traits provide a set of methods (● — )
  - Traits require a set of methods ( — )
  - Traits are purely behavioral (no state)

<table>
<thead>
<tr>
<th>TCircle</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>● area</td>
<td>radius</td>
</tr>
<tr>
<td>● bounds</td>
<td>radius:</td>
</tr>
<tr>
<td>● diameter</td>
<td>center</td>
</tr>
<tr>
<td>● hash</td>
<td>center:</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
How are Traits Used?

 Traits are the building blocks of classes

\[ \text{Class} = \text{superclass} + \text{state} + \text{traits} + \text{glue} \]

Traits do not replace single inheritance

They provide modularity *within* classes
Composition Rules

Class methods take precedence over trait methods
Explicit Conflict Resolution

- Override the conflict with a glue method
  - Aliases to access to the conflicting methods
- Avoid the conflict
  - Exclude conflicting method from one trait
Flattening Property (2 views on code)

Structured view

```
Object
  ColoredCircle
    draw
    TCircle
      draw
      radius
    TColor
      hue
      rgb
```

Flat view

```
Object
  ColoredCircle
    draw
    radius
    hue
    rgb
```

equivalent
Validation

- Implemented in Smalltalk
  - Smart method dictionary manipulation
  - Development environment extended

- Set-theoretic formalization to prove the flattening property

- Applications
  - Refactored the collection hierarchy
  - Applied on metaclass composition

- Broad impact
Classboxes

- Modularity of (groups of) classes
  - address packaging problems

- Compose classes and methods
- Import classes
- Extend existing classes locally

Advantage: Module system for OO language handling class extensions
Class extensions

- method defined in a package whose class is defined elsewhere
- can be an extension or a redefinition
- exists in CLOS, Smalltalk, Ruby, ...
What we want...

Base

Socket
ping(host)

HTTPSocket
getHttp(url)

HTML Support

HTMLParser

HTMLEntity

HTMLBody

HTMLAnchor

Dead Link Checker

Socket
ping(host)

Linkchecker
check(url)

DeadLinksVisitor
visitBody(body)

visitAnchor(anchos)

HTMLVisitor

HTMLBody
acceptVisitor(visitor)

HTMLEntity
acceptVisitor(visitor)

HTMLAnchor
acceptVisitor(visitor)

...
Classbox Model Definition

- The Classbox Model is a module system supporting local rebinding

- A Classbox
  - Is a unit of scoping (acts as a namespace)
  - Can define classes, methods, variables
  - Can import class definitions

- Classes and methods always belong to exactly one classbox
Local rebinding

Extensions act as if they were global, but they are local to the classbox

Two different implementations!
Local rebinding (ctd)

Parser creates HTML nodes

Parser creates “visitable” instances
Validation

- Implemented in Smalltalk
  - new method lookup algorithm
  - extended development environment

- Classboxes
  - Set-theoretic formalization to prove the local rebinding property
  - applied on web application framework
Conclusion

- Composition at different levels
  - Traits: methods in classes
  - Classboxes: classes in modules
- Automatic conflict detection
- Explicit (manual) conflict resolution
  - overriding, cancellation, aliasing


Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, Andrew P. Black, *Traits: Composable Units of Behavior*, Proceedings of ECOOP 2003

Stephane Ducasse, Nathanael Schärli, Oscar Nierstrasz, Roel Wuyts and Andrew Black, *Traits: A Mechanism for fine-grained Reuse*, Submittet to TOPLAS.
Meta conclusion

Composition Mechanisms

Reflection Symbiosis

Development Environments

Software Components

DeComp
Resulting Class Hierarchy

Result consists of 3 parts

- Abstract root class Collection
  - Only contains the methods supported by all collection classes
- Abstract classes with the public functionality for different kind of collections
- Layer of concrete collection classes
  - They inherit the public functionality from one of the functionality classes
  - They use a trait that adds a specific implementation
Resulting Trait Hierarchies

- Two trait hierarchies
  - Functional Traits
  - Implementation Traits
- Very fine-grained
  - Most traits consist of multiple subtraits