Modular Stateful Traits in Pharo

Pablo Tesone

IMT - Lille-Douai / RMod INRIA
Traits

- We have duplicated behaviour.
- Present in different hierarchies.
- Inheritance is not enough
Traits (2)

- Reuse of behaviour
- Share behaviour outside the hierarchy of classes.
- Reduce the duplication of code.
- Manual resolution of conflicts.
- Not affecting the subclassing.
Some Implementation Details

• Traits are flattened.

• Classes have copy of the methods

• Handled by the traits implementation… not by the programmer.

• Transparent to the Runtime
But not perfect...

- Only can interact with the instance through messages.
- Needs of glue code (usually accessors).
- Cannot define instance variables (slots)
Stateful Traits

- Reuse of slots.
- Improving the Initialisation
- We reduce the glue code.
- Adding new operations to handle conflicts.
**Stateful Traits (2)**

- Flattened Methods.
- Flattened Slots.
- Everything is compiled correctly.
- Methods defined in a trait accesses slots defined in it directly.

<table>
<thead>
<tr>
<th>Employee</th>
<th>External Resource</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>creation</td>
<td>creation</td>
<td>creation</td>
</tr>
<tr>
<td>modification</td>
<td>modification</td>
<td>modification</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>initializeTrait</td>
<td>initializeTrait</td>
<td>initializeTrait</td>
</tr>
<tr>
<td>logModification</td>
<td>logModification</td>
<td>logModification</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Diagram:

```
   TAudit
   creation
   modification
   initializeTrait
   logModification

<<uses>>
Employee
... creation
... modification
... initializeTrait
... logModification
...<

<<uses>>
External Resource
... creation
... modification
... initializeTrait
... logModification
...<

<<uses>>
Sale
... creation
... modification
... initializeTrait
... logModification
...<
```
Initialisation of instances

- Traited classes have another initialisation method.

- Perform the initialisation of all the traits.

- Generated if missed by the traits mechanism.
Initialisation of instances

Employee >> initialize

super initialize.
self initializeTrait

Employee >> initializeTrait

super initialize.
self initializeTrait

Sale >> initialize

super initialize.
self initializeTrait

TWithRol
rol
initializeTrait
rol:

TAudit
creation
modification
initializeTrait
logModification

Employee
...initialize
initializeTrait
initializeTAudit
initializeTWithRol

External Resource
...initialize
initializeTrait
initialize

Sale
...initialize
initializeTrait
initialize
New Trait Composition

Operations

- Alias Slot
- Remove Slot
- Merge Slots

Adding Slots brings new conflicts.
That's not new!!! Kill him!!
Also Fix other problems

Pharo 6 Traits

- Monolithic Impl.
- Not Extensible
- Limited Polymorphic w/classes
Modular Implementation

- The kernel does not know nothing about traits.
- They are loaded after.
- Reducing Coditional Code.
Modular Implementation

Pharo 6

Behavior >> includesBehavior: aClass

self isTrait ifTrue: [ ^false ].
^self == aClass or:[self inheritsFrom: aClass]

Pharo 7

Behavior >> includesBehavior: aClass

^self == aClass or:[self inheritsFrom: aClass]

Trait >> includesBehavior: aClass

^false
Solved by Polymorphism
Polymorphism w/Classes

- The traits are traited classes.....

- But you cannot instantiate them.

- We only provide the different behaviour.

- Simplifies the tools
Extensible Algebra

- We need the algebra to solve conflicts.
- New operations to handle slots:
  - Rename
  - Remove
  - Alias
- Also the algebra easily extensible.
• The class building process has been modified. Allowing external extensions.

• All the creation of classes with traits are extension methods.
Does it scale?

• The methods and slots are flattened

• The slots are compiled in the target class.

• The initialisation code is chained.

• During runtime, there is no difference.

• 100 % Backward compatible.

• No VM Changes or needed support.
Bonus: simplification of MetaObject - Protocol

- Same MOP with classes.
- Classes, Traits (whatever) are only source of slots and methods.
- Better API for selectors, localSelectors and slots and localSlots.
- Simplifying the tools.
- System Traits deprecated (TClass, TBehavior, …)
- Still in progress…. so many tools to update.
For the same price.... (?)

Kernel Reduction

<table>
<thead>
<tr>
<th>Lines</th>
<th>22,606</th>
<th>15 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>2,897</td>
<td>21 %</td>
</tr>
<tr>
<td>Bootstrap Process</td>
<td>5 min</td>
<td>30 %</td>
</tr>
<tr>
<td>Overall Pharo Build</td>
<td>4 min</td>
<td>20 %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lines</th>
<th>6,557</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affected Packages</td>
<td>89</td>
</tr>
</tbody>
</table>

Tool Simplification
Conclusion

- Stateful Traits!

- Modular Solution (faster bootstrap, you can avoid it).

- Extensible

- Simpler MOP / Polymorphism with classes.

- Same performance!

- 100% Backward Compatible

THANKS!!!