

Traits at Work

S. Ducasse, D. Cassou, C. Delaunay, D. Pollet
INRIA

Roadmap

- Reuse in single inheritance world
- Traits
- Applications

The evil root problem

- <http://onsmalltalk.com/sandstonedb-simple-activerecord-style-persistence-in-squeak>
- To use SandstoneDb, ***just subclass*** SDAciveRecord
- Well I already have a domain root

Can't we?

Plug behavior without having
to change superclass?

Reuse the same behavior?



Use delegation

Use delegation

© Cartoonbank.com



*"Instead of 'It sucks' you could say,
'It doesn't speak to me.'"*

Copy and paste

Copy and paste



Classes... hmmm

Classes are schizophrenic!

units of creation vs. units of reuse

Traits: Units of composable *behavior* (no state)

multiple *implementation* inheritance

composer is in *control*

resolve conflicts via ignore / alias

E.g.: Magnitude, or Ruby's Comparable
given the total order relation,
provide the comparison operators $\leq$, $<$, $>$,
 $\geq \dots$

Units of *composable* *behavior*

multiple *implementation* inheritance

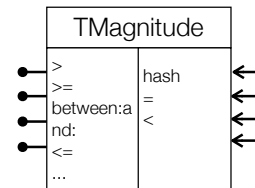
composer is in *control*

resolve conflicts via ignore / alias

backward compatible

Traits are parameterized behaviors:

- **provide** a set of methods
- **require** a set of methods
- purely behavioral (no state)



Class

=

Superclass

+ State

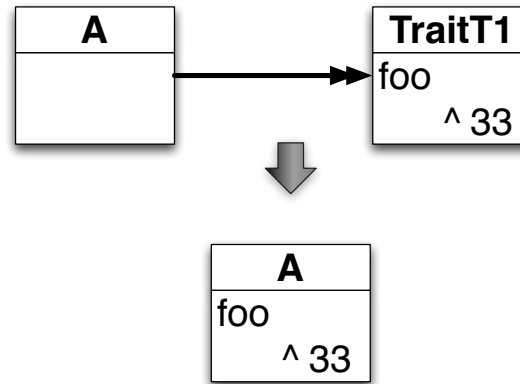
+ Traits

+ Methods

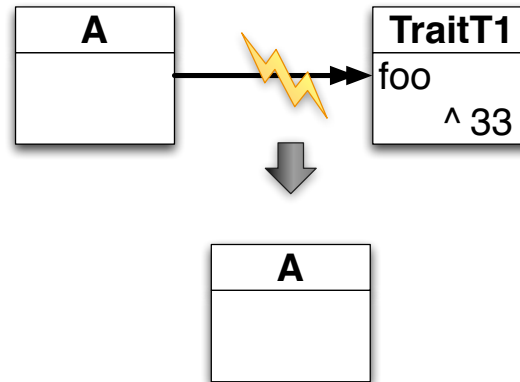
Traits do ***NOT*** exist at runtime

- Traits are like macros
- Method defined in class take precedence over trait methods

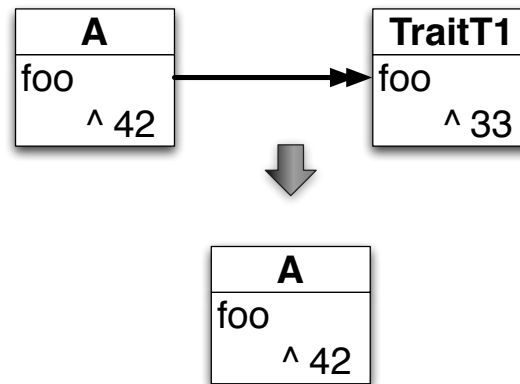
Using T1



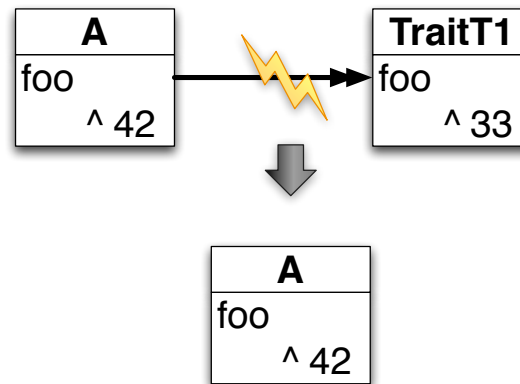
Not using anymore T1

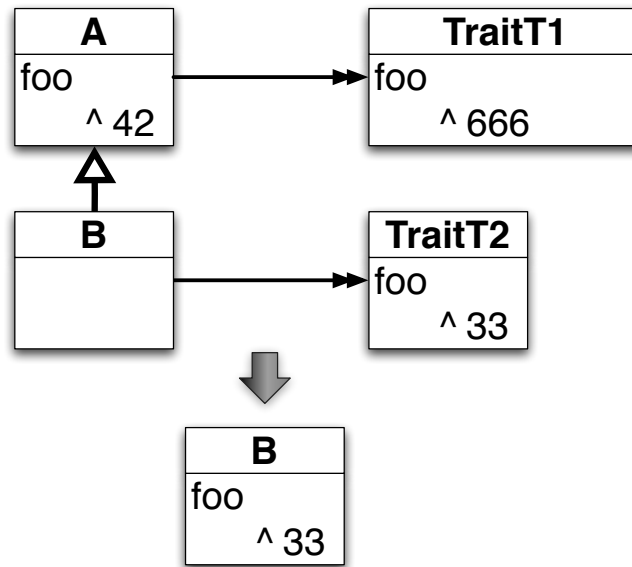


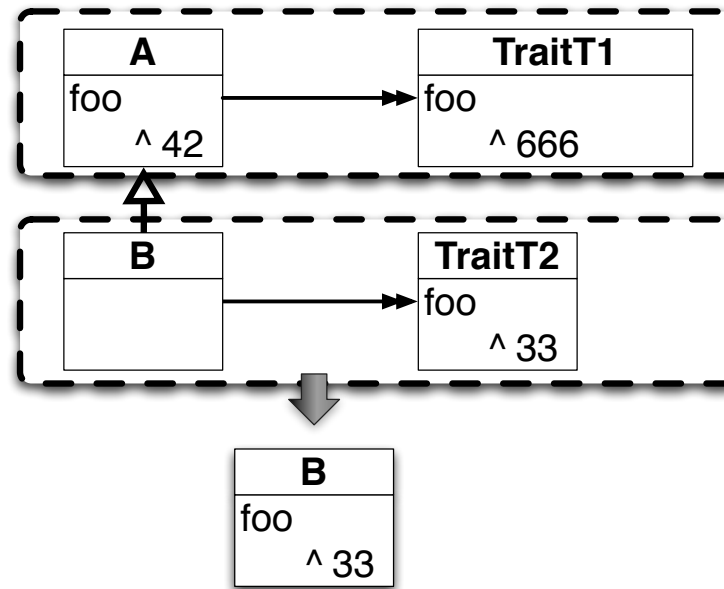
Composer has power

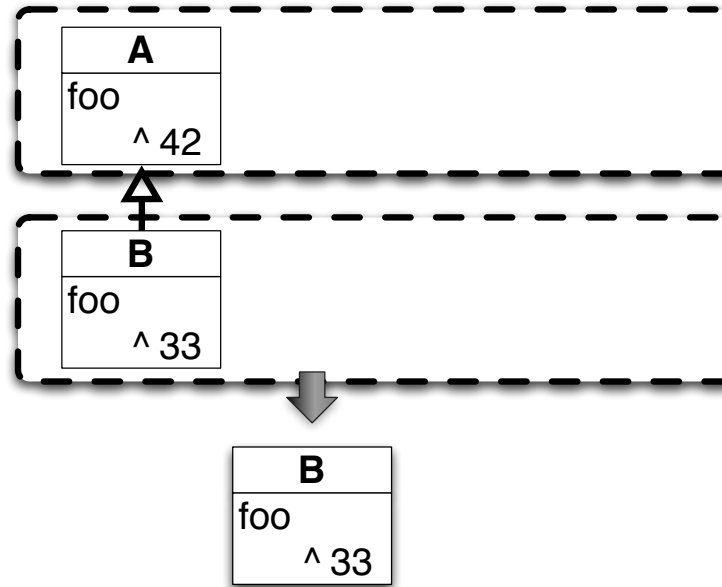


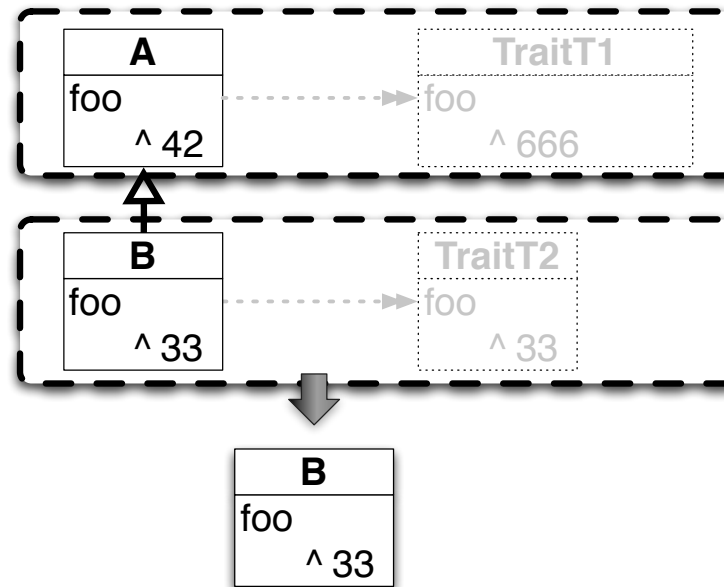
Composer has power



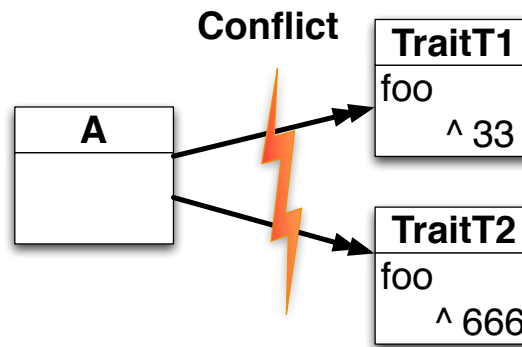




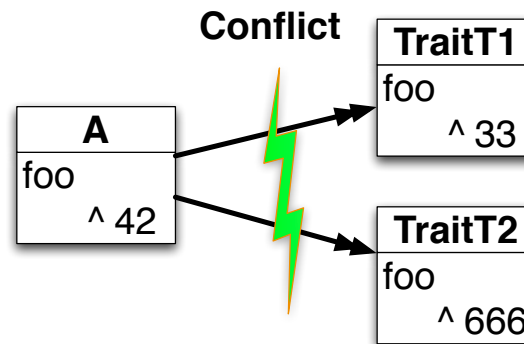




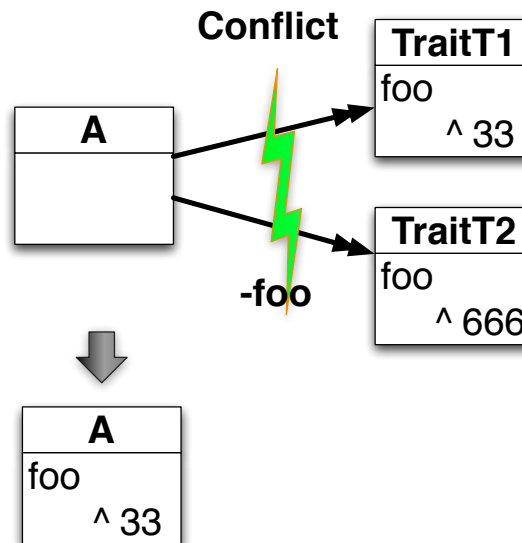
Conflicts



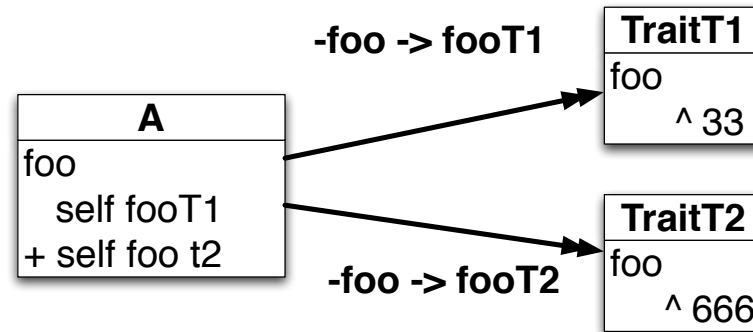
Resolved: “Overrides”



Resolved: Ignore



Access to ignored methods



Applications

- Building tests out of common traits
- Nile
- Polymorph
- Miro



Test Traits

Photo from <http://www.flickr.com/photos/keithm/> (Creative Commons BY-NC-SA 3.0)

Common Protocols

accessing size capacity at: at:put:

testing isEmpty occurrencesOf:
includes: contains:

adding, add: addAll:
removing remove: removeAll: remove:ifAbsent:

enumerating do: collect: select: reject:
inject:into: detect: detect:ifNone:

converting asBag asSet asArray asOrderedCollection
asSortedCollection asSortedCollection:

creating with: withAll:

Existing Tests

“Test-by-Use™”

No systematic testing

features

limit conditions

: (

Duplicated test methods

Tests for ad hoc behavior

Test Traits

One test trait per protocol

- requires accessors to a fixture

- provides systematic domain-level tests

Test classes

- compose test traits

- define the fixture

- define additional specific tests

Test Class =

- Superclass (TestCase)
- + fixture
- + test traits
- + glue methods

TPutTest >> testAtPut

```
self nonEmpty at: self anIndex put: self aValue.  
self assert:  
  (self nonEmpty at: self anIndex) == self aValue.
```

TPutTest >> testAtPutOutOfBounds

```
self  
  should: [self empty at: self anIndex put: self aValue]  
  raise: Error.
```

TPutTest >> testAtPutTwoValues

```
self nonEmpty at: self anIndex put: self aValue.  
self nonEmpty at: self anIndex put: self anotherValue.  
self assert:  
  (self nonEmpty at: self anIndex) == self anotherValue.
```

For TPutTest, the fixture must provide:

`empty nonEmpty`: instances of the collection

`anIndex`: integer or dictionary key or...

`aValue anotherValue`: legal for the collection

Each test class:

controls which test traits to compose (and how)

provides ad-hoc tests

groups all test code for a domain class

⋮)

Results

27 test traits

150 tests written

29 fixture req.

test runner reports:

765 runs

Test trait	Users	Methods ⁵	Ad-hoc	Effective
	dir. / inh.	req. → prov.	meth. → tests	unit tests
TAddForUniquenessTest	1 / 1	3 → 4	3 → 0	4
TAddTest	3 / 4	3 → 7	10 → 1	28
TCloneTest	9 / 11	2 → 3	18 → 0	33
TCopyPreservingIdentityTest	1 / 1	1 → 1	1 → 0	1
TCopyTest	6 / 7	2 → 5	12 → 0	35
TCreationWithTest	3 / 3	1 → 7	3 → 0	21
TDictionaryAccessingTest	1 / 2	3 → 13	3 → 0	26
TDictionaryAddingTest	1 / 2	3 → 4	3 → 0	8
TDictionaryComparingTest	1 / 2	0 → 1	0 → 0	2
TDictionaryCopyingTest	1 / 2	3 → 2	3 → 0	4
TDictionaryEnumeratingTest	1 / 2	3 → 9	3 → 0	18
TDictionaryImplementationTest	1 / 2	0 → 8	0 → 1	16
TDictionaryPrintingTest	1 / 2	3 → 2 +1	3 → 0	4
TDictionaryRemovingTest	1 / 2	3 → 4 +1	3 → 0	8
TEmptySequenceableTest	2 / 2	3 → 6 +3	7 → 0	12
TEmptyTest	6 / 13	2 → 8	22 → 0	104
TGrowableTest	1 / 1	5 → 3	5 → 1	3
TIdentityAddTest	1 / 1	2 → 1 +1	3 → 0	1
TIncludesTest	8 / 10	5 → 6	41 → 0	60
TIndexAccessingTest	3 / 3	1 → 13	3 → 2	39
TIterateSequenceableTest	2 / 2	3 → 3	6 → 0	6
TIterateTest	2 / 9	7 → 20 +2	49 → 6	180
TPutTest	3 / 4	4 → 3	12 → 1	12
TRemoveForMultiplenessTest	2 / 3	1 → 1	2 → 1	3
TRemoveTest	2 / 4	2 → 4	6 → 0	16
TSizeTest	2 / 9	2 → 2	14 → 0	18
TStructuralEqualityTest	2 / 1	2 → 4	2 → 0	4

Results

One test written, ~**4.7** run

average on a wide subset of the collections classes

still 1.8 / 1 when counting all methods

■
■)

Balances to strike:

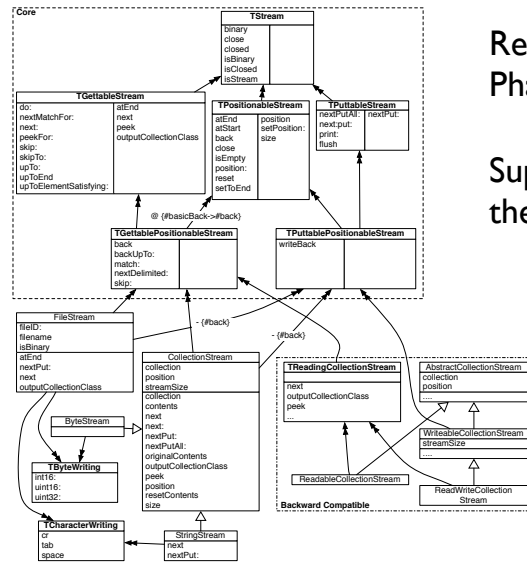
tests & fixtures: explicit vs. generic

inheritance, pre-composed traits...

Nile

Reimplementing streams in
Pharo and Squeak

Supports old and new styles with
the same traits recomposed



The great story of Nile

First ReadStream/ReadWriteStream/WriteStream
rewritten based on traits
but Squeakers not happy with old design anyway

Second (in **2 hours**)

ReadStream/ReadWriteStream/WriteStream rewritten
based on traits

- + one single class Stream based on the **same** traits
- + backward compatible



Example

Object subclass: #NSFileStream

uses: NSTGettablePositionableStream + NSTPuttablePositionableStream ...

Object subclass: #NSCollectionStream

uses: NSTGettablePositionableStream + NSTPuttablePositionableStream ...

NSCollectionStream

next

"Reads the next object in the stream and returns it."

<primitive: 65>

^ self atEnd

ifFalse: [collection at: (position := position + 1)]

NSFileStream>>next: amount

"Reads the next amount objects in the stream and returns a collection containing them in the same order."

| count buff |

buff := self outputCollectionClass new: amount.

count := self primRead: self getFileID into: buff startingAt: 1 count: amount.

^ count = amount

ifTrue: [buff]

ifFalse: [buff copyFrom: 1 to: count].

NSFileStream>>next

"Reads the next object in the stream and returns it. Please ensure that the stream is not at its end before calling #next.
Behavior is not defined if the stream is at its end."

^ (self next: 1) first



Traits

Implemented in Squeak/Pharo Smalltalk

Fully backwards compatible

No performance penalty for method lookup

Refactored Streams

Collection tests

In Scala (but looks more like mixins)

Replace classes in Fortress (SUN Microsystems)

Introduced in Perl6, Slate, DrScheme, AmbientTalk,

May be in Javascript!

