Redesigning with Traits

The Nile Stream trait-based Library

Damien Cassou[1] Stéphane Ducasse[1] Roel Wuyts[2]

[1]LISTIC, University of Savoie, France

[2]IMEC, Leuven and Université Libre de Bruxelles

European Smalltalk User Group, 2007

Outline

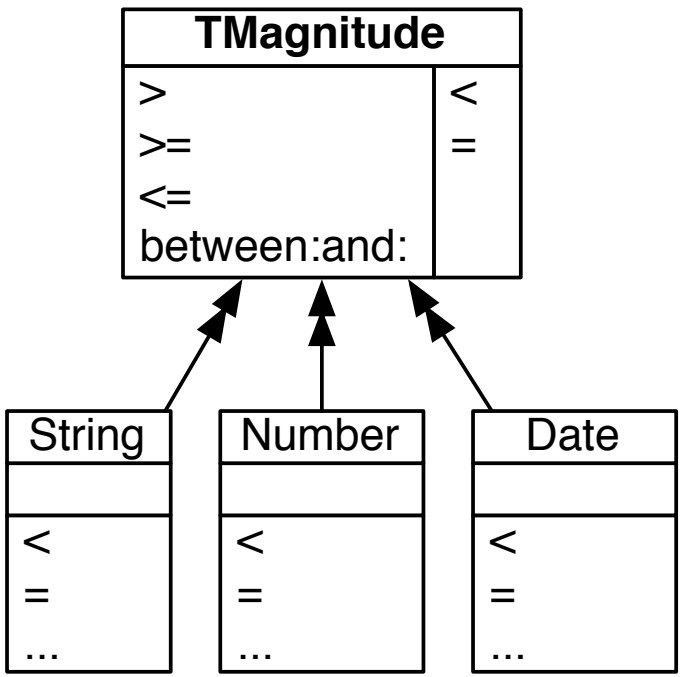
Outline

Traits Introduction

Traits

- group of reusable methods
- implemented and required methods
- conflict management
- implemented in Perl 6, Fortress, Scala, Smalltalk

Trait sample



- TMagnitude offers four methods and requires two
- A class uses traits and implements required methods

Problems

Observation

Never used to design framemorks/libraries from scratch

Questions

- ideal trait granularity?
- how much code reuse?
- trait or class?
- limits, constraints, problems?

Designing a New Trait-based Stream Library

Solution

- development of an adapted project
- choosing the stream library

Needs

- compatibility with ANSI Smalltalk
- compatibility with the current Squeak implementation
- speed efficient

Streams

Streams

Streams are used to read and write data sequentially in collections, files and network sockets.

Reading

- next
- next:
- peek

Writing

- nextPut:
- nextPutAll:
- next:put:

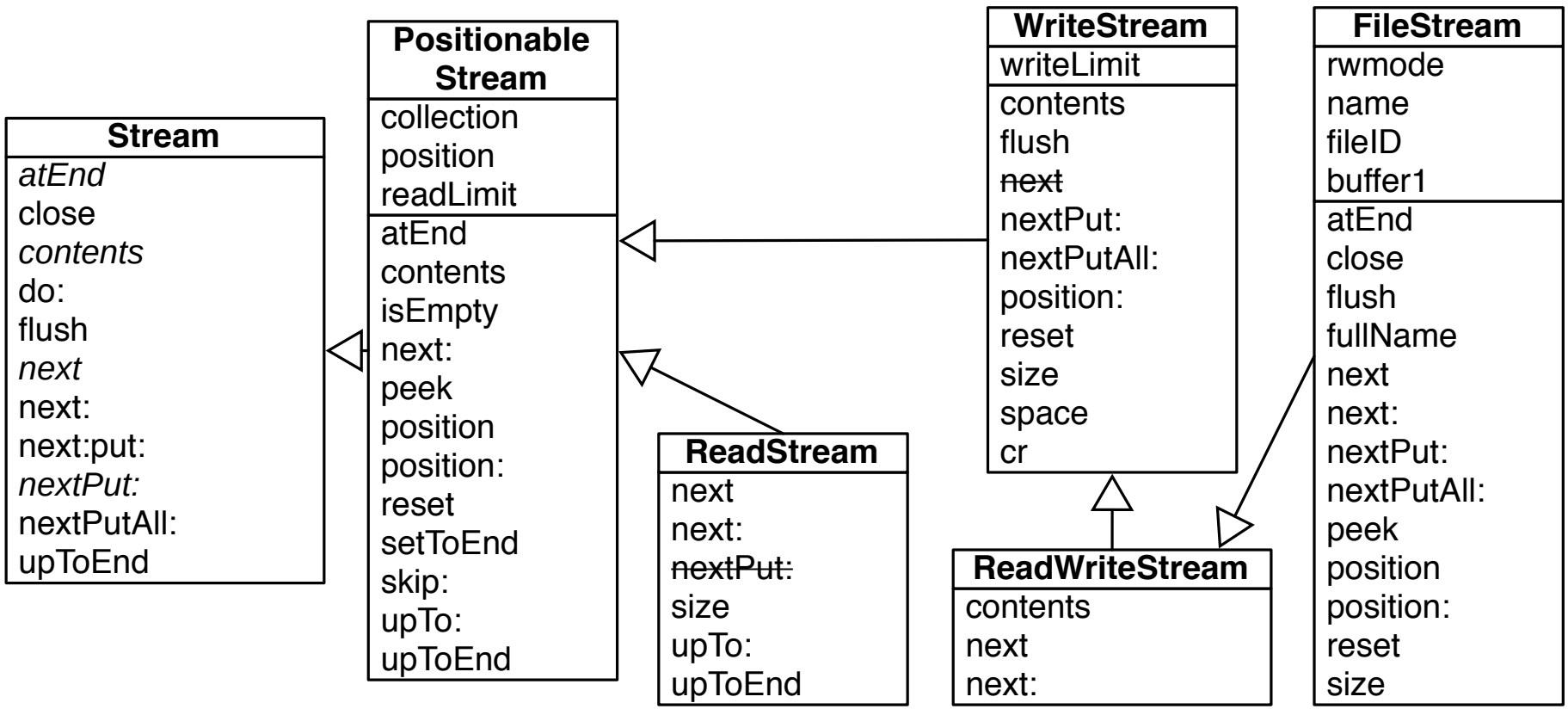
Positioning

- position:
- reset
- setToEnd

Why streams ?

- naturally modeled using multiple inheritance
- existing implementations (some of them trait-based)
- complex library

Squeak Implementation

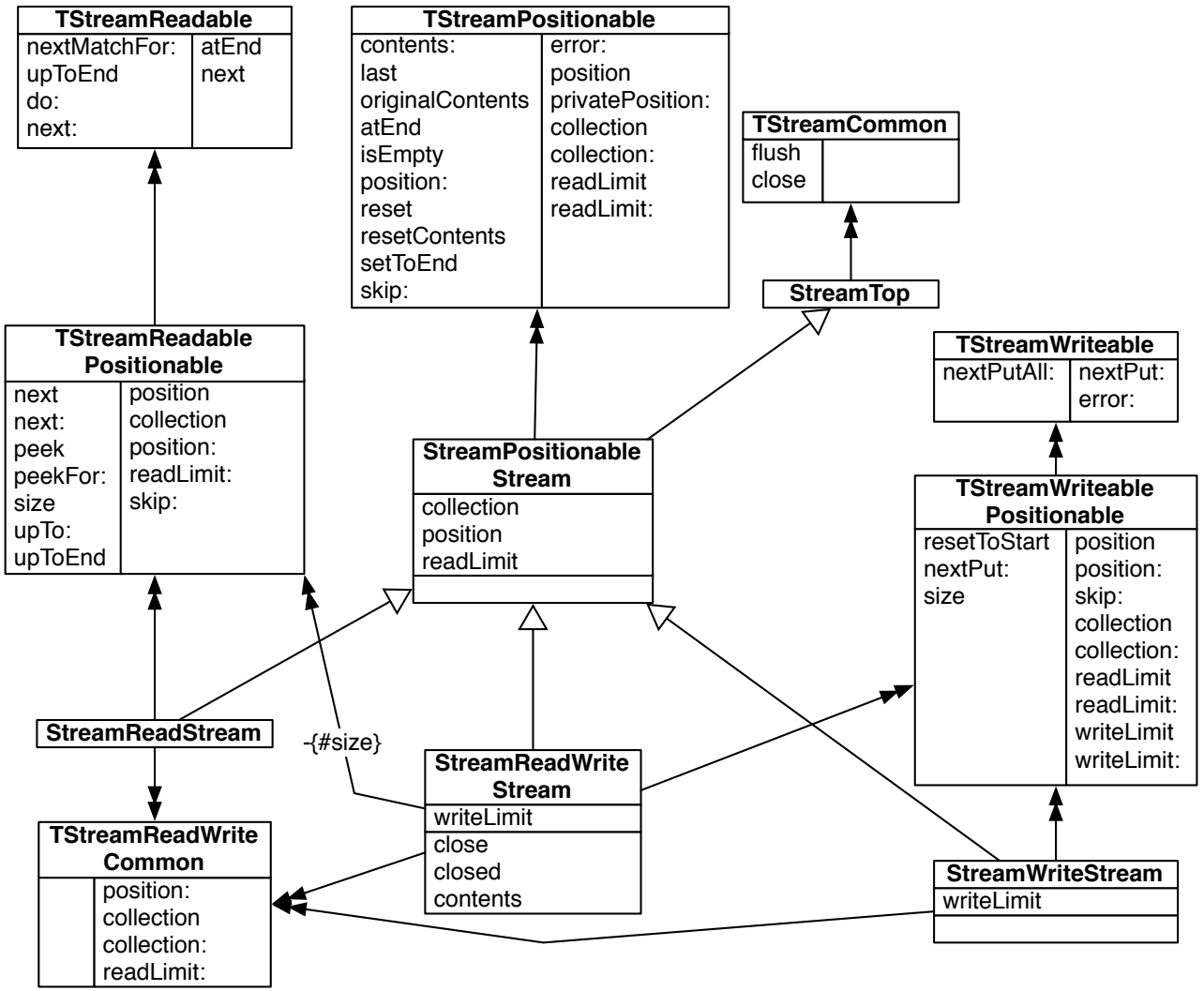


- everything in the Stream class
- methods disabled
- reimplemented methods

- methods reactivated
- files inherit from collections

Schärli's Refactoring

OOPSLA 2003

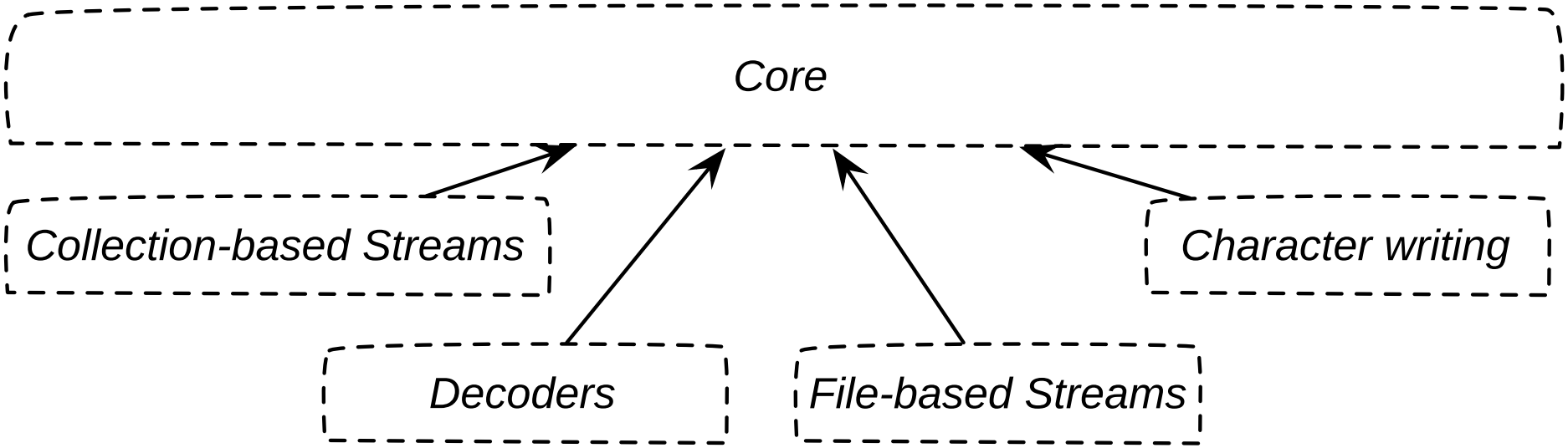


Only a refactoring

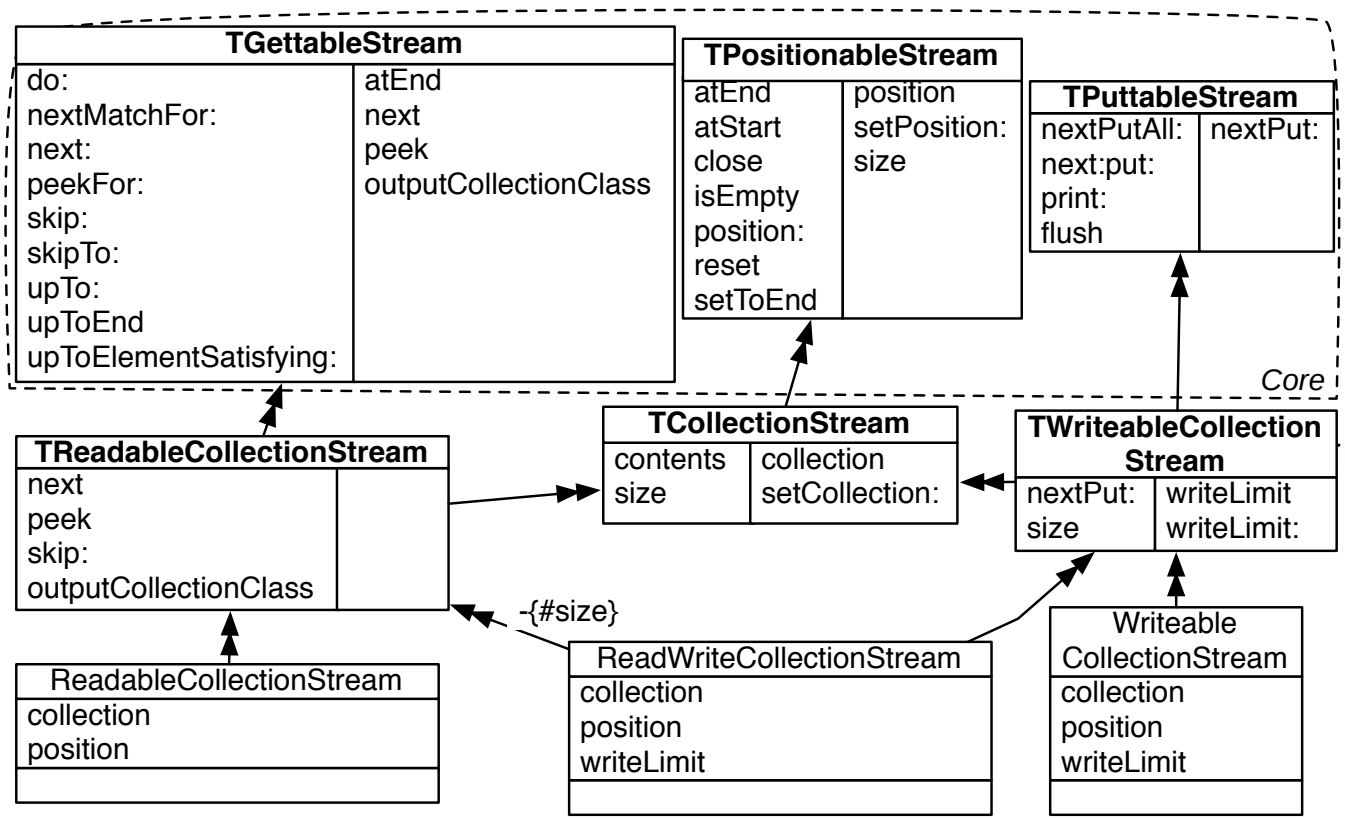
- no positioning for files
- peek depends on position and collection
- ...

Outline

Nile overview

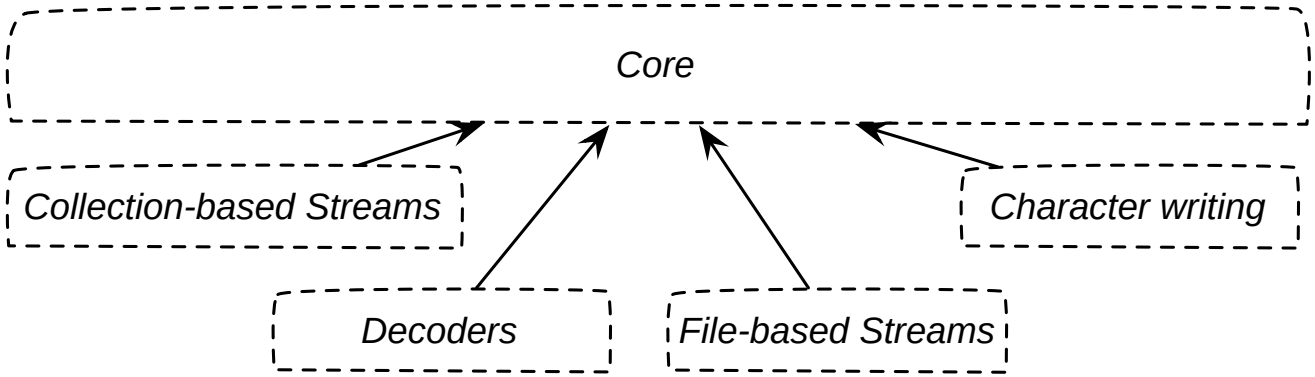


Collection-based Streams



- a trait per basic functionality
- traits for collections and one class for each original Squeak class

Other Implemented Libraries



Other Implemented Libraries

Files	file-based streams
Random	random number generator
SharedQueue	queue concurrent access
Decoder	pipe system to chain streams (compression, multiplexing. . .)

and many others. . .

New Design

Good Code Reuse

- good code reuse : 22 classes use TGettableStream
- important gain : TCharacterWriting requires implementing one method to gain 8

Better Implementation

- 18% fewer methods
- 15% less byte-code
- not forced to use unwanted behavior

Better Performance

- faster execution
- traits don’t prevent optimization

Trait-related Problems Discovered

- interface pollution
  - traits require accessors
  - sometimes an extra getter or setter
- IDE support for traits needs improvement
  - difficult to visualize and navigate

Outline

Further Work

- Having an equivalent implementation?
  - much more entities (11 traits+classes compared to 4 classes in Squeak)
  - simplify using only read/write access?
  - lot's of ugly code in the original implementation

Conclusion

- Context
  - traits needed to be tested in a new project
  - a stream library is a good project example
- Realized work
  - existing implementation was analyzed
  - new design was created
  - efficient implementation
  - problem analysis
- really improves the design
- needs further work