Components, Not Classes!

Or
OO Extensions Considered Harmful
(Part II)

Samuel A. Falvo II

2010 November 20

Forth Distributions

- Gforth
- SwiftForth
- Eforth
- F#
- Win32Forth
- PFE
- FICL
- RetroForth

OO Extensions

- FML
- MOPS
- Neon
- SWOOP
- OOF
- Mini-OOF
- “Word Classes”
- And a list of homebrew solutions.

Example Problem

- You wrote a Socket class using Swoop that I want to reuse. My software so far uses a port of OOF for RetroForth. How to solve this problem?

Example Problem

- You wrote a Socket class using Swoop that I want to reuse. My software so far uses a port of OOF for RetroForth. How to solve this problem?
  - Port Swoop to RetroForth and use Socket class as-is. Then I write an OOF "proxy" class, whose objects proxy method calls from OOF to Swoop and vice versa.

Example Problem

- You wrote a Socket class using Swoop that I want to reuse. My software so far uses a port of OOF for RetroForth. How to solve this problem?
  - Rewrite Socket class in OOF for seamless integration with existing code. Requires I have a license to make such changes.

Example Problem

- You wrote a Socket class using Swoop that I want to reuse. My software so far uses a port of OOF for RetroForth. How to solve this problem?
  - I could ditch RetroForth and adopt SwiftForth. This involves rewriting the entire code base, for RetroForth is not an ANSI Forth.

Problems Compound!

- You wrote a Socket class using Swoop that I want to reuse
- Another wrote a GUI toolkit using FML.
- My software targets RetroForth using a specially ported version of OOF.
  - Solutions for when you only have two object systems no longer work when you have three or more. What do you do then?

Forget the object-oriented extensions.

YOU DON'T NEED THEM.

Just write Forth!

Eating My Own Dog Food

- Enterprise applications are *boring*.
- Object oriented programming *invented* to solve (then) complex problems in computer simulations.
- Games are simulations, by definition.

A video game seems an ideal candidate for object-oriented implementation.

HOGWASH.

Equilibrium

- Video game written in SwiftForth for Linux.
- Over 550 word definitions.
- 2343 lines of code across 16 modules.
- 10 modules easily reusable for other games.
- AI not yet written, but will be soon.
- **AI being developed independently of core game logic.**

Equilibrium

- Conceived through Object-Oriented Analysis and Test- and Domain-Driven Development.
- Procedural implementation built on **Relational Algebra.**

NOTHIN' BUT FORTH.

Equilibrium

- Simpler than I *ever* expected to write.
- Most code changes kept very localized.
- Debugging usually trivial, thanks to test-enforced modularity.
- In 2343 lines of code, only 2 multi-day debugging sessions, **neither due to stack imbalances.**

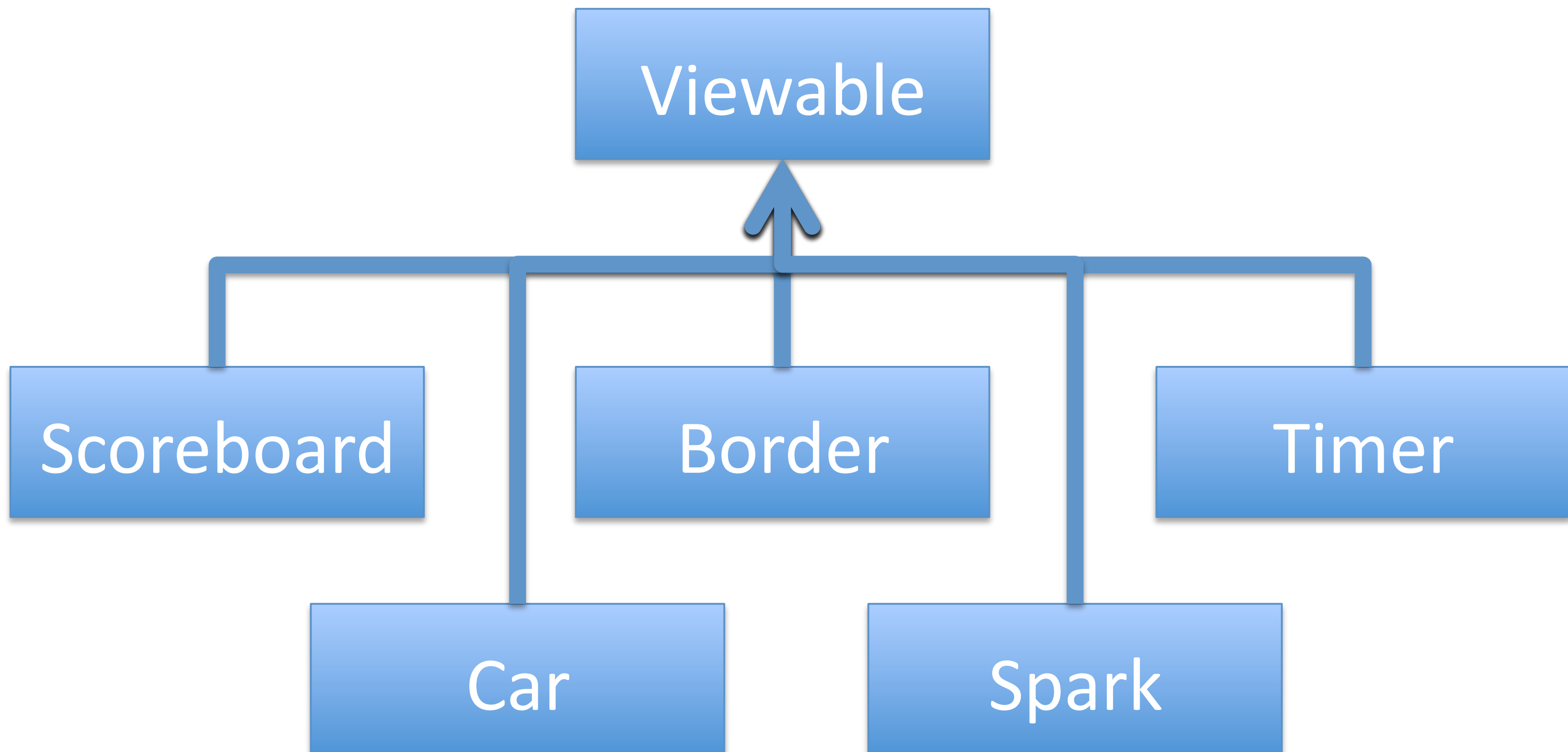
- Ate my own dog food, **and loved it.**

D I S C L A I M E R

- Equilibrium is not intended to illustrate theoretically perfect, clean, or “the right way” to code in Forth.

- It is, however, intended to illustrate the concepts of component-oriented programming over object-oriented programming.

Viewable: Class Hierarchy



Viewable: Component

- Code in viewable.f
- Applies to **all** viewable entities we know of.
- Locally managed storage.
- Fields relevant to **all** kinds of viewables.
- Global over the set of all viewables.

Achieving Polymorphism

- Only each viewable “knows” how to draw itself.
- Polymorphism through **callbacks**.
- Modules support common behaviors though!

Achieving Inheritance

- Two methods to obtain inheritance:
  - Composition of functionality.
  - Dependencies on other modules.
- Inheritance essentially the same as a join.
- Unique object identifiers used as foreign keys to other tables.
- See mobility.f, positionable.f, and elasticity.f for concrete examples of inheritance at work.

On Messages and Methods

- Per Smalltalk, message names intended to have global meaning, scoped by ***type***.
  - at:put: understood by all to be an array setter.
  - at: understood to be an array getter.
- Thus, Smalltalk exhibits *duck-typing*, used to synthesize a natural-language-inspired semantics.

On Messages and Methods

- Per Forth, word names intended to have global meaning, scoped by ***context***.
  - @ universally understood to mean "fetch" for all Forth.
  - +room understood to mean the precondition that enough room exists (for something) in the context of a single problem solution (see DItI).
- Thus, Forth exhibits *natural-language semantics,* used to synthesize data types.

On Messages and Methods

- ***Names and their XTs*** **correspond to object messages.**
- ***Definitions*** **correspond to class methods.**

On Messages and Methods

- Modules define *sets* to which objects may belong.
  - The set of all Viewable objects
  - The set of all Elastic objects
  - The set of all Punctual objects
  - The set of all Reversible objects

On Messages and Methods

- Abstract data types are defined by the *set* of objects in its alphabet and the operations you can perform on them.
- Thus, modules define interfaces.
- Not all methods need be polymorphic.
- Closer to Squeak Smalltalk's "Traits" than either classes or pure interfaces.
- Use vocabularies to prevent namespace collisions.

On Messages and Methods

- Module words aware of the *set* of objects.
- Thus, words can operate on more than one object.
  - Implemented well, obviates the need for multiple dispatch.
  - See collided? and pPreserved.

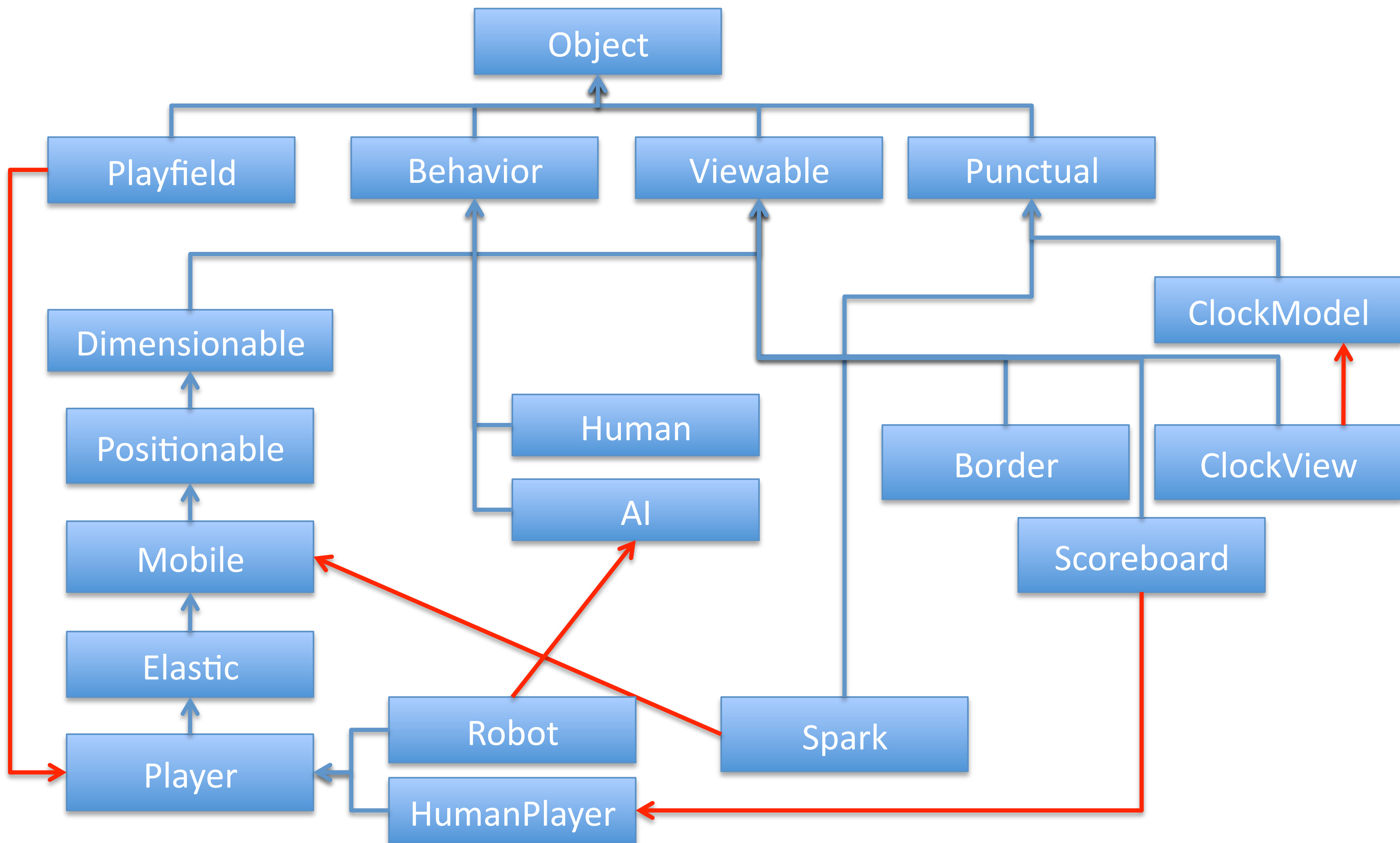
Coding Patterns Behind COP

- Declarative, Imperative, then Inquisitive
- Aggressive Handling
- Partial Continuation
- **Ascetic Programming** – *disown your objects!*
- Factor Indices
- Demultiplex by Request

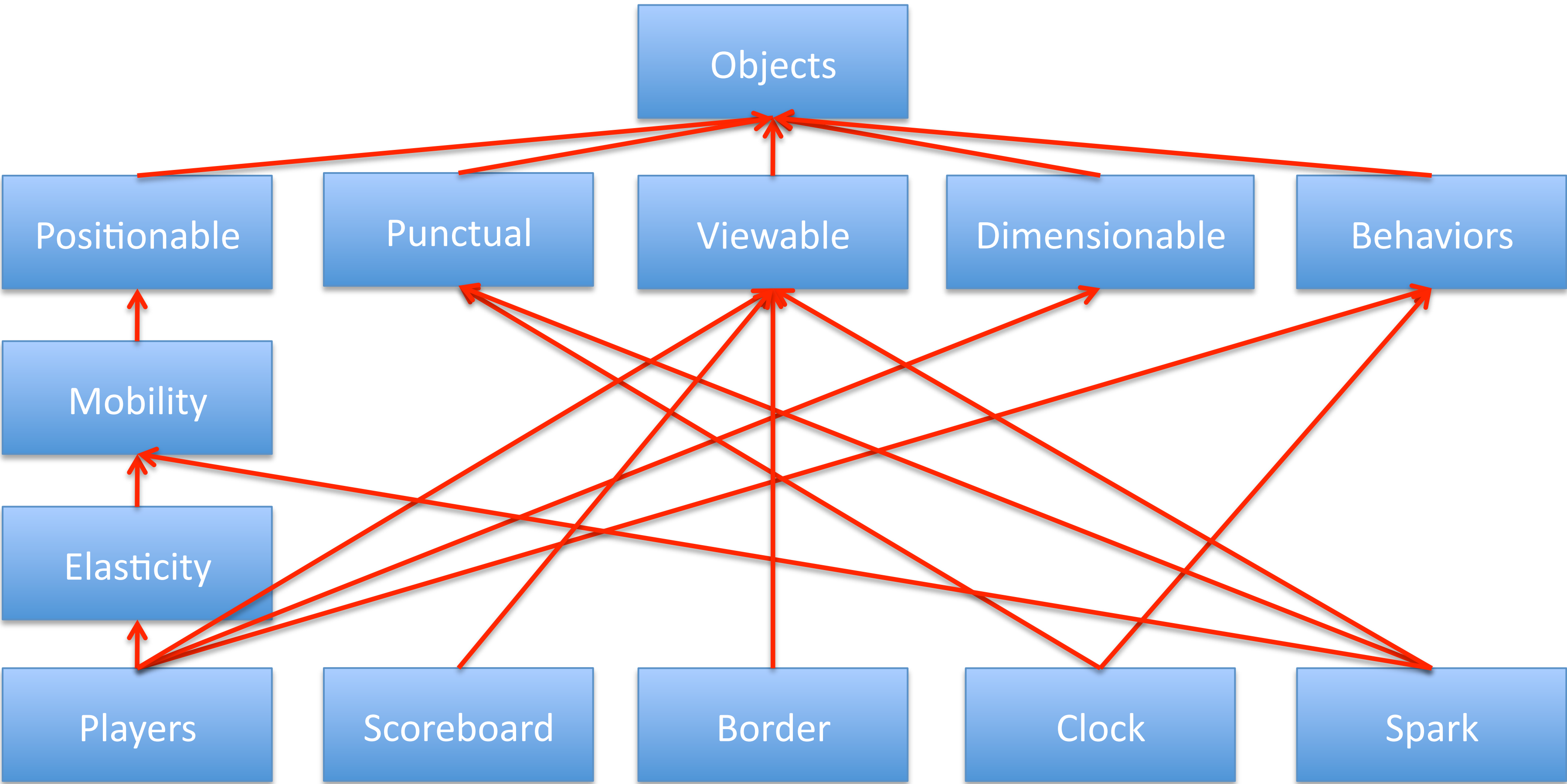
Coding Patterns Behind COP

- Modules employ relational algebra.
- Forth lacks garbage collection.
- Thus, use CRUD-y interfaces.
  - Create
  - Read
  - Update
  - Destroy

Game Class Hierarchy (Abridged)



Game Component Hierarchy



Review: Features of COP

- Modules manage their own storage requirements.
- Polymorphism through callbacks.
- Use relational algebra to express object and/or type relationships.
- Modules define **sets** of objects
- Words define messages you can send to an object of that set.
- CRUD-y interfaces.
- Single- and multiple-dispatch are the same in most cases.
- Very wide module/type hierarchies are the norm.
- Objects constructed piece by piece, not cast through a mold.
- Types not equal to classes.
- Objects can change types as necessary.
- Components tend to be highly orthogonal, domain-specific concepts.
- **Works on microprocessors not optimized for OO, like 6809, 65816, Z-80.**

The Disadvantage of COP

- Naïve module implementations have noticeable runtime performance issues.
  - Game uses O(n) table scans for nearly everything.
  - Modules sometimes use other modules, making some operation complexities effectively $O(n^2)$.
  - On 2.8GHz machine, difference between 8 and 100 on-screen objects is 1.5 *milli*-seconds.
- Plenty of opportunity for optimization if needed.

Wait a minute . . .

- Objects are systematized forms of premature optimizations.
  - Object references and pertinent state exist in same physical record.
  - Oracle calls arrays of such records an Index-Optimized Table.
- Experience with relational DBs suggests other optimizations exist which preserves benefits of COP. IOTs **not** always needed!

Optimization Strategies

- Rolling database columns
  - Move frequently accessed rows to the front.
  - Probabilistic O(1) performance in lots of cases.
  - Thrashing results in $O(n^2)$ worst-case performance.
- Hashing
  - Example: Bits 6-11 of foreign key field used to select bucket (identity hash).
  - Improves look-up time by factor of N, where N is number of buckets supported.
- Trees / Tries / SkipLists / etc…

Conclusion

- Equilibrium demonstrates validity of COP concept
  - TDD and DDD without the use of objects.
  - OOA without the use of object-oriented syntax extensions.
- COP may finally enable a marketplace of reusable Forth modules, a la CPAN for Perl.

Conclusion

- BitBucket.org repository for the game:
  - http://www.bitbucket.org/kc5tja/equilibrium

- Falvotech Blog
  - http://www.falvotech.com/blog2/blog.fs

THANK YOU!

Q & A

EXTRA SLIDES

On Messages and Methods

- In Smalltalk, what does **reverse** mean?
  - For arrays, it means to swap end-for-end each element.
  - For a sprite, it means to reverse direction of movement.
- This can sometimes be confusing! You end up having to remember context after all!

On Messages and Methods

- *Interfaces* solve this problem.
  - Global semantics understood at *level of interfaces*, not individual messages.

On Messages and Methods

- In Component-Oriented Forth, what does **reverse** mean?
  - For arrays, it means to swap end-for-end each element.
  - For a sprite, it means to reverse direction of movement.
- This is every-day experience for us. We're baffled at why other languages don't support *more* of this "hyper-static global" behavior.