Back 2 Basics with Goo

Jonathan Bachrach
MIT AI Lab
Goo

- Goals
- Examples
- Relation
- Definition
- State
- Future
Goo Hello World

.puts out "hello world"
Goo Goals

- Simple
- Productive
- Powerful
- Extensible
- Dynamic
- Efficient
- Real-time
- Teaching and research vehicle
- Electronic music is domain to keep it honest
Simplicity

- 10K lines 10 page manual
- Hard limit – pressure makes pearls
Best of All Worlds

- Want scripting and delivery language rolled into one
- Tools work better
- No artificial boundaries and cliffs
- Never been done effectively

- Electronic music forces realism
Goo Ancestors

• Language Design is Difficult
  – Leverage proven ideas
  – Make progress in selective directions

• Ancestors
  – Scheme
  – Cecil
  – Dylan
Goo <=> Scheme

- Concise naming
- Procedural macros
- Objects all the way

- Long-winded naming
- Rewrite rule only
- Only records
Goo <=> Cecil

- Prefix syntax
- Scheme inspired special forms
- Infix syntax
- Smalltalk inspired special forms
Goo <=> Dylan

- Prefix syntax
- Procedural macros
- Rationalized collection protocol / hierarchy
- Always open
- Predicate types

- Infix syntax
- Rewrite-rule only …
- Conflated collection protocol / hierarchy
- Sealing
- Fixed set of types
Object Orientation

• Assume you know OO basics

• Motivations:
  – Abstraction
  – Reuse
  – Extensibility
Goo: OO & MM

(dcc <point> (<any>))
(dp point-x (<point> => <int>) 0)
(dp point-y (<point> => <int>) 0)

(dv p1 (new <point>))

(dm + (p1|<point> p2|<point> => <point>)
  (new <point>
    point-x (+ (point-x p1) (point-x p2))
    point-y (+ (point-y p1) (point-y p2)))))
Language Design: User Goals -- The “ilities”

- Learnability
- Understandability
- Writability
- Modifiability
- Runnability
- Interoperability
Learnability

- Simple
- Small
- Regular
- Gentle learning curve

- Perlis: “Symmetry is a complexity reducing concept...; seek it everywhere.”
Goo: Learnability

• Simple and Small:
  – 18 special forms: if, seq, set, fun, def, let, loc, esc, fin, dv, dm, dg, new, dc, dp, ds, ct, quote
  – 7 macros: try, rep, mif, and, or, cond, case

• Gentle Learning Curve:
  – Graceful transition from functional to object-oriented programming

  – Perlis: “Purely applicative languages are poorly applicable.”
Goo: Special Forms

IF    (IF ,test ,then ,else)
SEQ   (SEQ ,@forms)
SET   (SET ,name ,form) | (SET (,name ,@args) ,form)
DEF   (DEF ,var ,init)
FUN   (FUN ,sig ,@body)
LOC   (LOC ((,name ,sig ,@body) ...) .@body)
ESC   (ESC ,name ,@body)
FIN   (FIN ,protected-form ,@cleanup-forms)
DV    (DV ,var ,form)
DM    (DM ,name ,sig ,@body)
DG    (DG ,name ,sig)
DC    (DC ,name (,@parents))
DP    (DP ,getter (,class => ,type) [,init])
NEW   (NEW (,@parents) ,@prop-inits)

sig       (,@vars) | (,@vars => ,var)
var       ,name | (,name ,type)
prop-init ,name ,value
Understandability

• Natural notation
• Simple to predict behavior
• Modular
• Models application domain
• Concise
Goo: Understandability

• Describable by a small interpreter
  – Size of interpreter is a measure of complexity of language

• Regular syntax
  – Debatable whether prefix is natural, but it’s simple, regular and easy to implement
Writability

- Expressive features and abstraction mechanisms
- Concise notation
- Domain-specific features and support
- No error-prone features
- Internal correctness checks (e.g., typechecking) to avoid errors
Goo: Error Proneness

• No out of language errors
  – At worst all errors will be caught in language at runtime
  – At best potential errors such as “no applicable methods” will be caught statically earlier and in batch

• Unbiased dispatching and inheritance
  – Example: Method selection not based on lexicographical order as in CLOS
Design Principle Two: Planned Serendipity

• Serendipity:
  – M-W: the faculty or phenomenon of finding valuable or agreeable things not sought for

• Orthogonality
  – Collection of few independent powerful features combindable without restriction

• Consistency
Goo: Serendipity

- Objects all the way down
- Slots accessed only through calls to generic’s
- Simple orthogonal special forms
- Expression oriented
- Example:
  - Exception handling can be built out of a few special forms: `esc`, `fin`, `loc`, ...
Modifiability

- Minimal redundancy
- Hooks for extensibility included automatically
- Users equal partner in language design
- No features that make it hard to change code later
Goo: Extensible Syntax

• Syntactic Abstraction
• Procedural macros
• WSYWIG
  – Pattern matching
  – Code generation
• Example:
  (ds (unless ,test ,@body)
       `(if (not ,test) (seq ,@body)))
Goo: Multimethods

- Can add methods outside original class definition:
  - (dm jb-print (x|<node>) ...)
  - (dm jb-print (x|<str>) ...)
Goo: Generic Accessors

• All slot access goes through generic function calls
• Can easily redefine these generic’s without affecting client code
Runnability

• Features for programmers to control efficiency
• Analyzable by compilers and other tools
Goo: Optional Types

• All bindings and parameters can take optional types
• Rapid prototype without types
• Add types for documentation and efficiency

• Example:
  (dm format (s msg args|...) ...)
  (dm format (s|<stream> msg|<str> args|...) ...)

Goo: Pay as You Go

• Don’t charge for features not used
• Pay more for features used in more complicated ways
• Examples:
  – Dispatch
    • Just function call if method unambiguous from argument types
    • Otherwise require dynamic method lookup
  – Goo’s bind-exit called “esc”
    • Local exits are set + goto
    • Non local exits must create a frame and stack alloc an exit closure
The Rub

- Support for evolutionary programming creates a serious challenge for implementers
- Straightforward implementations would exact a tremendous performance penalty
Implementation Strategy

• Simple dynamic compilation
• Maintains both
  – optimization and
  – interactivity
Initial Loose Compilation

- Very quick compilation
- Generate minimal dependencies
  - only names and macros
Dynamic Whole Program Compilation

• Assume complete information
• Perform aggressive type flow analysis
  – Chooses, clones and inlines methods
• Compilation can be triggered manually, through dependencies, or through feedback
Dependency Tracking

• Assumptions are tracked
• Changed assumptions trigger recompilation
• Based on Fun-O-Dylan approach
  – Dependencies logged on bindings
  – Record dependent and compilation stage
Simple Code Generator

• Focus is on high-level optimizations
• Potentially gen-code direct from AST with approximated peep-hole optimizations
Save Image

• Save executable copy of image to disk
  – Maintains optimizations and dependencies
  – Uses dump/undump approach of emacs

• Avoid hassles of
  – File formats
  – Databases
  – etc
Status

- Fully bootstrapped
- Module system
- Dynamic C-based code-gen
- Dependency tracking
- Flow-typist by summer’s end
Research Directions

- Language Design
  - Dynamic parameterized types
  - Dynamic Interfaces
  - Series
  - Macros

- Language Implementation
  - Dynamic compilation
  - Analysis/optimizations
  - Visualization
  - Real-time