Embedded Domain Specific Languages in Clojure

Dimitry Gashinsky (@digash)

Pico Quantitative Trading LLC.

18 August 2010

Outline

Introduction

Main Part

Questions

Who am I? vanity

Used to be by day

Coder for financial, security and technology industries.

Used to be at night

Lisp enthusiast and computational philosopher.

Now in twilight

VP of engineering for Pico Quantitative Trading, which includes all of the above.

My First Lisp and DSL exposure.

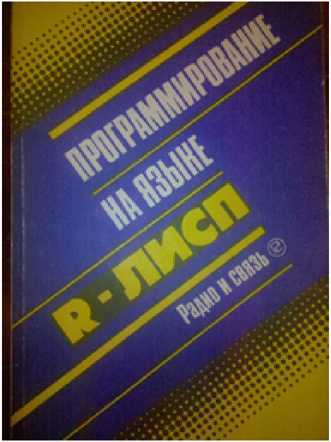


Figure: R-LISP Book (1991)

REDUCE

To symbolically solve

$$\int_0^y cos(2x)\,dx. \tag{1}$$

Start REDUCE REPL

```
 Reduce (Free CSL version), 18-Aug-10 ...

 1: int(cos(2x),x,y,2y);

 sin(4*y) - sin(2*y)
---------------------
          2
```

MAGIC !

Definition of embedded (a.k.a. internal[1]) DSL

Enumerating some of the important attributes

- targeted towards specific problem
- limited in the scope as oppose to general purpose language
- usually not turing complete
- self-documenting

[1] [[http://martinfowler.com/articles/languageWorkbench.html][Language Workbenches: The Killer-App for Domain Specific Languages?]]

Definition of embedded DSL continue

Enumerating more of the important attributes

- bottom-up design[2]
- parasitic in its nature
  - feeds on host abstract syntax tree (AST)
  - hides in the host's syntax
  - reproduces more quickly and in greater numbers than its hosts

[2] Programming Bottom-Up

Motivation for DSL

When in need of wisdom use a good book[2]

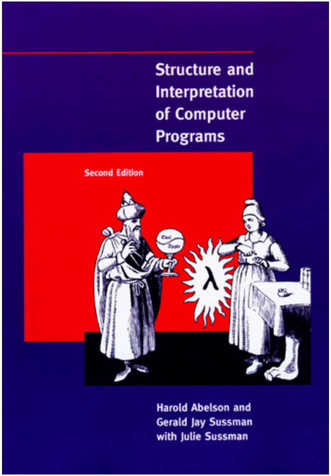


Figure: Structure and Interpretation of Computer Programs

Metalinguistic Abstraction (a.k.a. DSL)

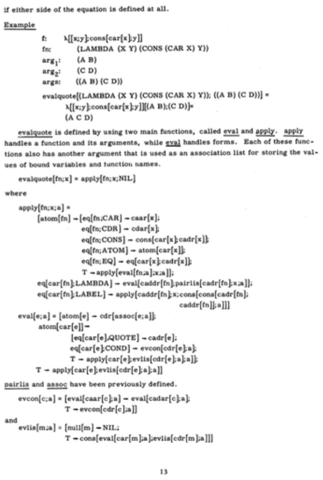
Establishing new languages

- a powerful strategy for controlling complexity
- particularly important to computer programming, because we can implement these languages

Implementing languages is not that scary encouragement

"The evaluator [or compiler], which determines the meaning of expressions in a programming language, is just another program."

Maxwell's Equations of Software! – Alan Kay inspiration



if either side of the equation is defined at all.

Example

f: λ[[x;y];cons[car[x];y]]

fn: (LAMBDA (X Y) (CONS (CAR X) Y))

arg_1: (A B)

arg_2: (C D)

args: ((A B) (C D))

```
evalquote[(LAMBDA (X Y) (CONS (CAR X) Y)); ((A B) (C D))] =
        λ[[x;y];cons[car[x];y]][(A B);(C D)]=
        (A C D)
```

evalquote is defined by using two main functions, called eval and apply. apply handles a function and its arguments, while eval handles forms. Each of these functions also has another argument that is used as an association list for storing the values of bound variables and function names.

```
evalquote[fn;x] = apply[fn;x;NIL]
```

where

```
apply[fn;x;a] =
     [atom[fn] → [eq[fn;CAR] → caar[x];
                  eq[fn;CDR] → cdar[x];
                  eq[fn;CONS] → cons[car[x];cadr[x]];
                  eq[fn;ATOM] → atom[car[x]];
                  eq[fn;EQ] → eq[car[x];cadr[x]];
                  T → apply[eval[fn;a];x;a]];
     eq[car[fn];LAMBDA] → eval[caddr[fn];pairlis[cadr[fn];x;a]];
     eq[car[fn];LABEL] → apply[caddr[fn];x;cons[cons[cadr[fn];
                                                    caddr[fn]];a]]]
eval[e;a] = [atom[e] → cdr[assoc[e;a]];
     atom[car[e]]→
              [eq[car[e],QUOTE] → cadr[e];
              eq[car[e];COND] → evcon[cdr[e];a];
              T → apply[car[e];evlis[cdr[e];a];a]];
     T → apply[car[e];evlis[cdr[e];a];a]]
```

pairlis and assoc have been previously defined.

```
evcon[c;a] = [eval[caar[c];a] → eval[cadar[c];a];
              T → evcon[cdr[c];a]]
```

and

```
evlis[m;a] = [null[m] → NIL;
              T → cons[eval[car[m];a];evlis[cdr[m];a]]]
```

13

Figure: LISP 1.5 Programmer Manual page 13

Why Rich should have all the fn?

“We come to see ourselves as designers of languages, rather than only users of languages designed by others.”

“. . . computer science itself becomes no more (and no less) than the discipline of constructing appropriate descriptive languages.”

Clojure is an excellent host

Short list of features useful for DSLs:

- `fn #()`
- unquote and unquote-splicing
- code as data
- macrology

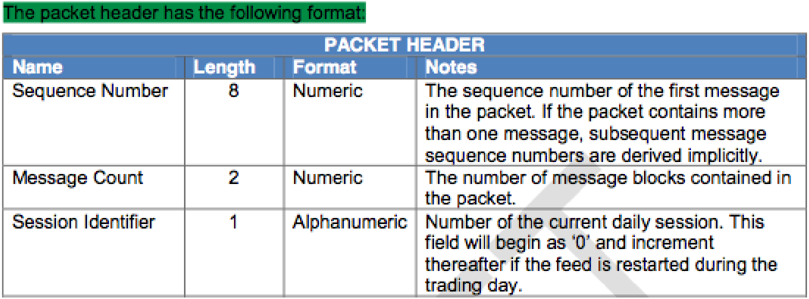
Some of DSLs that we wrote

configuration language

market data message specification

mmap file buffer to records mapping

Market Data Message Specification



The packet header has the following format:

PACKET HEADER			
Name	Length	Format	Notes
Sequence Number	8	Numeric	The sequence number of the first message in the packet. If the packet contains more than one message, subsequent message sequence numbers are derived implicitly.
Message Count	2	Numeric	The number of message blocks contained in the packet.
Session Identifier	1	Alphanumeric	Number of the current daily session. This field will begin as '0' and increment thereafter if the feed is restarted during the trading day.

Figure: One small piece of huge spec

Market Data Message Description Language

```
(defmessage PacketHeader
  [SequenceNumber    8 :numeric
   "The sequence number of the first message in
   the packet. If the packet contains more than
   one message, subsequent message sequence numbers
   are derived implicitly."]
  [MessageCount      2 :numeric
   "The number of message blocks contained in
   the packet."]
  [SessionIdentifier 1 :alphanumeric
   "Number of the current daily session. This field
   will begin as 0 and increment thereafter if the
   feed is restarted during the trading day."])
```

Some of the generated code for the message spec

```
(defrecord PacketHeader [buffer offset]
  Sizable
  (length [this] 16)
  PacketHeaderAccessor
  (getSequenceNumber
   [this]
   (parse-numeric buffer (+ (int offset) (int 0)) 8))
  (lengthSequenceNumber [this] 8)
  (offsetSequenceNumber [this] 0)
  (positionSequenceNumber [this] (+ offset 0))
;; ...
  (lengthMessageCount [this] 2)
  (getSessionIdentifier
   [this]           ; 8 + 2 = 10 bytes offset
   (parse-alphanumeric buffer (+ (int offset) (int 10)) 1))
   (offsetSessionIdentifier [this] 10)
   (positionSessionIdentifier [this] (+ offset 10)))
```

Implementation of defmessage

```
(defmacro defmessage [name & fields]
  (emit-message name fields))
```

Implementation of defmessage continue

```
(defn- emit-message [name fields]
  (let [reader-name (symbol (str name "Accessor"))
        buffer-symbol 'buffer
        offset-symbol 'offset]
    `(do
       (definterface ~reader-name
         ~@(emit-message-signatures fields))
       (defrecord ~name [^ByteBuffer ~buffer-symbol
                         ^int ~offset-symbol]
         Sizable (length [~'this] ~(emit-length fields))
         ~reader-name
         ~@(emit-message-methods fields buffer-symbol
                                 offset-symbol))
       (deftype ~(symbol (str name "Parser")) []
         Parsable   ; just wrap it
         (parse [~'this ~buffer-symbol ~offset-symbol]
                (new ~name ~buffer-symbol ~offset-symbol)))
```

Implementation of defmessage continue

```
(defn- emit-type [format length]
  ({:numeric ({1 Short
               2 Integer
               4 Long
               8 BigInteger} length)
    :alpha String
    :alphanumeric Long
    :mixed parser/byte-array-type} format))
```

Implementation of defmessage continue

```
(defn- emit-message-method-name [p n]
  (symbol (str p (name n))))

(defn- emit-message-signatures [fields]
  (mapcat
     (fn [[n l f c]]
       [`(~(with-meta (emit-message-method-name "get" n)
             {:tag (emit-type f l)}) [] ~c)
;; ...
        `(~(with-meta (emit-message-method-name "pos" n)
             {:tag Integer}) []
             "Position of field in the buffer.")])
     fields))
```

Memory Map Description Language

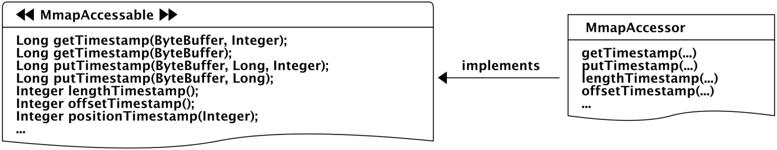
```
(defmmap MarketDataMmap
  [Timestamp        8 :integer]
  [TradeTime        8 :integer]
  [QuoteCondition   4 :character]
  [TradeCondition   4 :character]
  [Bid              4 :decimal]
  [Ask              4 :decimal]
  [Last             4 :decimal]
  [BidSize          4 :integer]
  [AskSize          4 :integer]
  [LastSize         4 :integer]
  [TotalAmount      8 :integer]
  [Volume           4 :integer]
  [Currency         3 :character]
  [Latency          8 :integer]
  [Symbol          12 :string])
```

Macroexpanded defmmap

```
--:---  defmmap-macroe--:---  defmmap-macroex--:---  defmmap-macroex--:---  defmmap-macroexp--:---  defmmap-macroexpand--:---  defmmap-macroexpand--:---  defmmap-macroexpand--:---  defmmap-macroexpand.clj  --:---  defmmap-macroexpand.clj
```

Figure: defmmap macroexpanded

Get a Java API too



◀◀ MmapAccessable ▶▶

```
Long getTimestamp(ByteBuffer, Integer);
Long getTimestamp(ByteBuffer);
Long putTimestamp(ByteBuffer, Long, Integer);
Long putTimestamp(ByteBuffer, Long);
Integer lengthTimestamp();
Integer offsetTimestamp();
Integer positionTimestamp(Integer);
...
```

MmapAccessor

```
getTimestamp(...)
putTimestamp(...)
lengthTimestamp(...)
offsetTimestamp(...)
...
```

Memory Map Accessable Java API

```
public interface MarketDataMmapAccessable {
    public Long getTimestamp(ByteBuffer, Integer);
    public Long getTimestamp(ByteBuffer);
    public Long putTimestamp(ByteBuffer, Long, Integer);
    public Long putTimestamp(ByteBuffer, Long);
    public Integer lengthTimestamp();
    public Integer offsetTimestamp();
    public Integer positionTimestamp(Integer);
    public Long getQuoteTime(ByteBuffer, Integer);
    public Long getQuoteTime(ByteBuffer);
    public Long putQuoteTime(ByteBuffer, Long, Integer);
    public Long putQuoteTime(ByteBuffer, Long);
    public Integer lengthQuoteTime();
    public Integer offsetQuoteTime();
    public Integer positionQuoteTime(Integer);
    public Long getTradeTime(ByteBuffer, Integer);
    public Long getTradeTime(ByteBuffer);
```

Memory Map Accessor Java API

```
public final class MarketDataMmapAccessor implements MarketD
    public MarketDataMmapAccessor(Object, Object);
    public MarketDataMmapAccessor();
    public Object length();
    public Integer positionSymbol(Integer);
    public Integer offsetSymbol();
    public Integer lengthSymbol();
    public String putSymbol(ByteBuffer, String);
    public String putSymbol(ByteBuffer, String, Integer);
    public String getSymbol(ByteBuffer);
    public String getSymbol(ByteBuffer, Integer);
    public Integer positionInternalLatency(Integer);
    public Integer offsetInternalLatency();
    public Integer lengthInternalLatency();
    public Long putInternalLatency(ByteBuffer, Long);
    public Long putInternalLatency(ByteBuffer, Long, Integer
    public Long getInternalLatency(ByteBuffer);
```

Questions

Questions?

mailto:dimitry@gashinsky.com

twitter:@digash

xmpp:i@digash.com

http://blog.digash.com