Common Lisp ecosystem and the software distribution model

Daniel Kochmański

TurtleWare

July 3, 2016
1936 – Alozno Church invents Lambda Calculus
1960 – John McCarthy presents paper about LISP
1973 – MIT Lisp Machine Project
1984 – AI winter (and the unfortunate marriage with Lisp)
1994 – Various dialects unification with Common Lisp
2000 – Renaissance of the community
Figure: “John McCarthy presents Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I” – Painting by Ferdinand Bol, 1662
Figure: John McCarthy (1927-2011)
**Figure:** A Knight machine preserved in the MIT Museum
### Figure: Lisp dialects timeline

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lisp 1.5</td>
<td>Lisp 1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maclisp</td>
<td></td>
<td>Maclisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interlisp</td>
<td></td>
<td>Interlisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZetaLisp</td>
<td></td>
<td>Lisp Machine Lisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheme</td>
<td></td>
<td>Scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIL</td>
<td>NIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common Lisp</td>
<td></td>
<td>Common Lisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoLISP</td>
<td></td>
<td>AutoLISP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISLISP</td>
<td></td>
<td>ISLISP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EuLisp</td>
<td></td>
<td>EuLisp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Racket</td>
<td></td>
<td>Racket</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arc</td>
<td></td>
<td>Arc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clojure</td>
<td></td>
<td>Clojure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFE</td>
<td></td>
<td>LFE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hy</td>
<td></td>
<td>Hy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Homoiconicity – code is a data, everything is an expression
• Macros – growing the language
• Higher-order and anonymous functions
• Flexible type system
• Garbage collection
• Read-Eval-Print-Loop – interactive programming
• The whole language is always available
• CLOS – generic function-style OOP
(defun compose (function &rest more-functions)
  (declare (optimize (speed 3) (safety 1) (debug 1)))
  (reduce
    (lambda (f g)
      (let ((f (ensure-function f))
          (g (ensure-function g)))
        (lambda (&rest arguments)
          (declare (dynamic-extent arguments))
          (funcall f (apply g arguments))))
    more-functions
    :initial-value function))
(defmacro while (test &body body)
  (let ((ret (gensym)))
    '('block nil
      (do ((,ret nil (progn ,@body)))
          ((not ,test) ,ret)))))
Figure: Common Lisp types hierarchy
- Wide range of implementations
- Active FOSS community
- Growing ecosystem
Figure: Common Lisp implementations graph
(defun eval. (e a)
  (cond
   ((atom e) (assoc. e a))
   ((atom (car e))
     (cond
      ((eq (car e) 'quote) (cad e))
      ((eq (car e) 'atom) (atom (eval. (cad e) a)))
      ((eq (car e) 'eq) (eq (eval. (cad e) a)
                          (eval. (caddr e) a)))
      ((eq (car e) 'car) (car (eval. (cad e) a)))
      ((eq (car e) 'cdr) (cdr (eval. (cad e) a)))
      ((eq (car e) 'cons) (cons (eval. (cad e) a)
                              (eval. (caddr e) a)))
      ((eq (car e) 'cond) (evcon. (cdr e) a))
      ('t (eval. (cons (assoc. (car e) a)
                     (cdr e))))
      ((eq (caar e) 'label)
       (eval. (cons (caddr e) (cdr e))
               (cons (list. (cad e) (car e)) a)))
      ((eq (caar e) 'lambda)
       (eval. (caddr e)
               (append. (pair. (cad e) (evlis. (cdr e) a))
                        a))))
  )
)

Yo dawg, I heard you like lisp so I put a lisp in your lisp so you can eval while you eval.

**Figure:** Typical Lisp programmer
Incremental compilation

CL-USER> (defun yyy () (xxx))
Yyy
CL-USER> (defun xxx () "Hello world")
XXX
CL-USER> (yyy)
"Hello world"
CL-USER> (defun xxx () "Bye world")
XXX
CL-USER> (yyy)
"Bye world"
FASt Load

CL-USER> (compile-file "xxx.lisp")
"/home/jack/xxx.fas"

CL-USER> (load "xxx")
T

CL-USER> (xxx)
"Hello world"
save-lisp-and-die and build-system

CL-USER> (sb-ext:save-lisp-and-die "xxx")

[jack@pandora ~]$ ./xxx
"Hello world"

CL-USER> (compiler::builder
    :program "xxx"
    :lisp-files ("file-1.lisp"
                   "file-2.lisp")
    :main-name "main")

[jack@pandora ~]$ ./xxx
"Hello world"
Deployment

The following facilities are wrappers around save-lisp-and-die or build-system (via ASDF which is covered later):

- cl-launch
- clbuild
- clon
- roswell
- uiop
Manual system definition example

(defparameter *source-files* 
  '("packages" 
    "utilities" 
    "classes" 
    "application"))

(mapcar #'(lambda (f) 
  (format t "Loading file ~A~%" f) 
  (load f)) 
  *source-files*)
defsystem, mk-defsystem and sbt-defsystem
ASDF – Another System Definition Facility
ISDF – Inferred System Description Facility
(defsystem #:metering
  :name "Metering" :version "3.2"
  :description "Portable Code Profiling Tool"
  :author "Mark Kantrowitz <mkant@cs.cmu.edu>"
  :maintainer "Daniel Kochmański <daniel@turtleware.eu>"
  :components (:(cl-source-file.cl "metering"))
  :in-order-to ((test-op (test-op #:metering/test))))

(defsystem #:metering/test
  :depends-on (#:metering #:fiveam)
  :components (:(file "metering-test"))
  :perform (test-op (o s)
    (funcall (intern (string '#:run!) #:metering/test
       #:metering-suite))))
Dependency graph example

Figure: McCLIM dependencies
State of the art

A few software distribution solutions out there:

- common-lisp-controller
- asdf-install
- Quicklisp
- NPM (JavaScript)
- RubyGems (Ruby)
- CPAN (Perl)
- aptitude
- guix
- pkgsrc
- portage
- NiX
Usage example

CL-USER> (ql:quickload 'clim-examples)
T

CL-USER> (clim-demo:demodemo)
; magic happens
Pros and cons

- Easy to use
- Well maintained
- Allows custom repositories
- Reliable
- Integrated with the language

- Poor documentation
- Single trust authority (not safe!)
Figure: CLinch Demo
Figure: Practical Common Lisp – Peter Seibel
Summary

- Lisp has quite a history
- Its features are slowly adapted in other languages
- Common Lisp has various implementations
- Distributing binaries isn’t a viable option for the CL developers (many binary-incompatible implementations)
- Quicklisp is in a similar spirit as the NPM and the RubyGems
Literature

- Revenge of the Nerds – Paul Graham [http://paulgraham.com/icad.html]
Attributions

- “Lisp” painting –
  http://classicprogrammerpaintings.com/post/142817850864/john-mccarthy-presents-recursive-functions-of

- John McCarthy photo (CC BY-SA 2.0) by null0 at
  http://www.flickr.com/photos/null0/272015955/

- Lisp Machine photo (CC BY-SA 3.0) – no machine-readable author provided. Jszigetvari assumed (based on copyright claims)

- Lisp dialects (CC BY-SA 3.0) –
  https://en.wikipedia.org/wiki/Lisp
Attributions

- Common Lisp types hierarchy (CC BY-NC-SA 3.0) by Greg Pfeil, http://sellout.github.io/2012/03/03/common-lisp-type-hierarchy/

- Common Lisp implementations graph (CC-BY-4.0) by Daniel Kochmański, https://common-lisp.net/project/ecl/posts/ECL-Quarterly-Volume-IV.html


About me

I build device prototypes and do FOSS consultancy in my own company TurtleWare. I specialize in the embedded systems, Linux kernel and userspace development, C/C++ and Common Lisp programming and compiler design.

This presentation is available at http://turtleware.eu/static/talks/pkgsrcCon-2016-lisp.pdf