Introduction to Medley Interlisp

January 3, 2021

by Blake McBride

Table of Contents

1 Introduction

Back in the '70s and '80s Lisp workstations were an important development. A portable version of one of those environments was created at that time called Medley. The source code to that original system has been made open-source, ported, and modernized. This introduction is about that system.

Medley originally supported Interlisp (an older dialect of Lisp) but grew to support Common Lisp as well. Medley, therefore, is a software development environment that embodies Interlisp and Common Lisp.

The Medley development environment is very different from the file-based model used today. The value of this system is to provide the ability to learn about, explore, and possibly utilize these benefits in future projects.

The majority of the system is written in Lisp and rides on a portable virtual machine written in portable C. The system uses an X11 window to display the bitmap in which the Lisp code manages. The system is very portable and has been ported to Linux, Macintosh, and other systems.

The purpose of this document is to provide access to this system to someone who is already familiar with current Common Lisp development environments in as brief a form as possible.

1.1 History

Lisp was formulated by John McCarthy in 1958 at MIT. Lisp quickly became the language of choice for the burgeoning field of Artificial Intelligence (AI) research. Lisp quickly spread in the academic community. Lisp also spread in the commercial world to support research projects done at those institutions.

Due in part to the youth of the industry, institutions often implemented their own version of Lisp, and those versions almost always differed from other Lisp implementations in ways that suited each particular institution. The two most common implementations were MacLisp and Interlisp. MacLisp came from MIT (the East coast), and Interlisp (which was derived from BBN Lisp) came from the Xerox Palo Alto Research Center in California (the West coast). Both systems bled into the commercial world.

Back in the 1960's through the early 80's, much research in AI was funded by DARPA which was a government agency. DARPA became tired of supporting projects written in incompatible Lisp implementations. They therefore issued an edict demanding a single, portable version of Lisp. Thus became Common Lisp.

While having a single version of Lisp had the benefit of making software more portable, it also had the negative effect of somewhat stifling the exploration in the language.

MacLisp was written in assembler code for particular machines. Since those machines don't exist anymore, MacLisp itself has disappeared. Interlisp was also largely written in assembler and those systems too have disappeared. However, there was at least one version of Interlisp written in C. That is, our system, Medley. Because it was written in C (to some, a portable assembler), it is still available today.

From its inception up until very recently, Medley was a commercial product and was unavailable freely or in source-code form. Recently, the system has been made open-source.

Due to the efforts of some of the original authors of Interlisp and others in the community, Medley is being made available to the general public.

2 Architecture

The Medley system operates in a virtual machine. Lisp code is either interpreted or compiled into (essentially) byte-code for this virtual machine. Use of a virtual machine in this way, while more popular today with environments such as Java and C#, was quite a bit more unique and innovative when Medley was developed. As it does with other languages, use of a virtual machine made Medley portable.

The Medley system is comprised of two major pieces. The first part is the virtual machine. This is called "maiko". The remainder of the system is written in a combination of Interlisp and Common Lisp.

The Medley system is image-based (like Smalltalk) rather than file-based (like most other systems such as C, Java, C#, Python, etc.). In file-based systems, source code resides in disk files. Interpreters interpret this code and compilers convert these files into machine code (placed in other files) for execution. Development amounts to the editing of these source files.

In an image-based systems you are working on a live, running system. As code is written, the running system evolves. At various points, the user can save an "image" of that running system. What that is is a single, binary, copy of the entire running system. It basically saves the state of the running machine at the point the image is saved. Later, the system may be restarted with a particular image and the system resumes from the exact point from which it was saved.

With an image-based system you can still write your source code to disk files for various purposes but that is not the principal mode of development. One of the beauties of image-based development is that when you want to see or change something, the entire system is at your immediate disposal. You do not need to search for the source files. Also, changes that you make take effect immediately. There is no need to rebuild and redeploy the system.

As opposed to having two language sub-systems (Interlisp and Common Lisp), Medley integrates the two such that both or either can be used. Interlisp functions as the "IL" package of Common Lisp.

3 Building & Running

Medley is open-source and may be obtained from GitHub. It is currently portable to Linux and Apple Mac systems. The core is written in portable C but the system depends on the X11 system for its display.

3.1 Building The System

3.1.1 Obtaining The System

The Medley system comes in two parts. The first is the C-based virtual machine named "maiko". It may be obtained from `https://github.com/Interlisp/maiko`.

The remainder of the system may be obtained from `https://github.com/Interlisp/medley`.

Once the system has been cloned, you should have two directories named `maiko` and medley.

3.1.2 Linux

On Linux, the system may be built with the following commands:

```
$ cd maiko/bin
```

At this point you will need to edit the file named `makefile-linux.x86_64-x` if you are using GCC rather than CLANG. Comment out the line (with a #) that defines `CC` for `clang` and uncomment the line (delete the #) for the line that defines `CC` for `gcc`.

```
$ ./makeright x
```

This creates the virtual machine in the `maiko/linux.x86_64` directory.

3.1.3 Mac

Building for the Mac requires an X-Server. This can be freely obtained at `https://www.xquartz.org/`. Download and install it.

After that, Medley build just like it does on Linux:

```
$ cd maiko/bin
$ ./makeright x
```

3.2 Running The System

The system can be run by typing:

```
$ cd medley
$ ./run-medley
```

Or, if you wish to start Medley up with a particular image, use:

```
$ cd medley
$ ./run-medley <image-file-name>
```

Once the system comes up, give it a few seconds to initialize.

The first time the system is run it loads the system image that comes with the system. When you exit the system the state of your machine is saved in a file named `~/lisp.virtualmem`. Subsequent system startups load the `~/lisp.virtualmem` image by default.

3.3 Exiting The System

The system may be exited from the Interlisp prompt by typing:

```
(LOGOUT)
```

Or from the Common Lisp prompt with:

```
(IL:LOGOUT)
```

When you logout of the system, Medley automatically creates a binary dump of your system located in your home directory named "lisp.virtualmem". The next time you run the system, if you don't specify a specific image to run, Medley restores that image so that you can continue right where you left off.

4 Medley

4.1 Orientation

Once the system has been started, you will see some windows with title bars. Navigation within the system is a little unusual.

To control a window, right-click on the window's title bar.

To bring up a system menu, right-click anywhere outside a window.

Windows titled "Exec" are read-eval-print loops. The title bar will also tell you whether you are running LISP (for Common Lisp), XCL (for Xerox Common Lisp), INTERLISP (for current Interlisp), or OLD-INTERLISP-T (for an older version of Interlisp).

When the system first comes up you will notice a window labeled "Exec (XCL)". That is a Xerox Common Lisp read-eval-print loop.

(Hint: When the system has completed its initialization process, a fresh prompt will appear.)

4.2 Different Lisp Environments

The system comes with four version of Lisp as follows:

`Interlisp`
: the latest Interlisp implementation

`Old Interlisp`
: an older Interlisp implementation

`Common Lisp`
: moving towards CLtl2 and then ANSI

`Xerox Common Lisp`
: Common Lisp with a number of Xerox enhancements

Even though each of the possible Exec's give you a default environment, all of the various Lisp systems are interchangeably available from any of the Exec windows via package specifications. For example: an Interlisp function names "ABC" may be run from Common Lisp via `(IL:ABC ...)`. Likewise, a Common Lisp function "DEF" may be executed from Interlisp via: `(CL:DEF ...)`.

4.3 Images And Files

The Medley system includes a virtual Machine (VM) that runs the Lisp programs. Medley can run Lisp code interactively or compiled into the byte-code for the Medley VM. This is a bit similar to the Java Virtual Machine or the .NET CLR but much more similar to the Smalltalk or Squeak environment.

In traditional systems, a developer edits source code files and then compiles those files into something the machine can execute — be it a machine executable file or byte-code for a VM. Medley does not work that way. Medley is an image-based system similar to Smalltalk or Squeak.

When Medley is started up, you are in the development and runtime environment. All of your development is done here and all of your programs are run here. You are essentially editing runnable programs in memory. When you exit the system, Medley creates a backup of this memory onto what is called a SYSOUT image. When the system is started up again, this image is read into memory and the system picks up right where it left off. This is what is called an image-based development environment.

In Medley, programs are developed, edited, debugged, and run all from within Medley. The state of this development is saved in your image file. There is also a way to save your system to more traditional disk files. This is called the "File Package".

Medley utilizes the following file types:

`lisp.virtualmem`
: This is located in your home directory and is an image of the last time Medley was run. If you start Medley without specifying an image to run, this image is used.

`XXXX.SYSOUT`
: This is an image that was explicitly saved by the developer. This may be loaded by specifying its name when starting Medley.

`XXXX (no file name extension)`
: Source files created with the "File Package".

`XXXX.LCOM`
: Compiled versions of the XXXX files

`XXXX.DFASL`
: Another compiled form of XXXX files

`XXXX.TEDIT`
: Text (like a word processor) in a Medley-specific format

4.4 Project Directory

Medley has something called its "connected directory". This is just another way to say "working directory". All file reads and writes occur (if no path is specified) in the connected directory. When Medley first starts up, its connected directory is your home directory. This can be changes by executing one of the following commands:

```
cd MyProject
```

or

```
(CNDIR "MyProject")
```

You will see the connected directory displayed at the top of the Medley window. Once the connected directory has been changed, all future image saves or file loads or saves will occur in this directory. Additionally, this directory will be saved in the image so when you re-load the image, you will already be in the desired directory.

One thing to note, however, is that the default image "lisp.virtualmem" will still be saved in your home directory. This assures that if you start Medley up without specifying an image, you will return to the last environment you were in.

4.5 Saving And Running Images

An image of the running system is automatically created every time you logout. This file is named "lisp.virtualmem" and is placed in your home directory. This way, when you restart the system you'll be right where you left off.

You can also manually save an image to a specific file at any point by typing the following from an Interlisp prompt:

```
(SYSOUT "my-file")
```

This will save your running image to a file named "my-file.SYSOUT" that can be loaded again by passing that file name to the `./run-medley` command.

4.6 Creating And Editing Functions

Objects (function, variables, etc.) can be credited and edited as follows:

```
ED(myfunc)
```

First, if it is a new object, the system will ask what type of object it is. After that the user will be presented with a GUI structure editor where the object may be defined or edited.

When the window is closed, via right-clicking on the title bar, the object will be saved.

4.7 Saving & Loading Source Code To/From Disk Files

When you edit or define a function, a variable, etc., Medley keeps track of the fact that they have been created or edited. These functions and variables are grouped and ultimately get associated with a file on your disk. In places, Interlisp refers to these disk files as "symbolic files".

`FILELST` list of user files the system is aware of

`SYSFILES` list of system files loaded

`(FILES?)` prompts user for which file to associate newly defined functions, variables, etc.

`(MAKEFILE "myfile")`
writes out all of the functions, variables, etc. that have been associated with file "myfile"

`(LOAD "myfile")`
loads a previously saved file

`(CLEANUP)`
interactively save all changed files

5 Interlisp

Interlisp is a dialect of Lisp and as such it is based in the familiar syntax of left-parenthesis, function name, arguments, and right-parenthesis. Besides many of the functions having different names and arguments compared to Common Lisp, Interlisp has many other, more fundamental, differences from Common Lisp. While this section will not go into any of the functional differences between Interlisp and Common Lisp, it will attempt to detail the more fundamental differences between the two. The reference manual may be used for a detail description of the Interlisp functions.

5.1 Case

Interlisp uses mixed case. That is, upper-case letters and lower-case letters are treated as different. This means you can have a variable name `my-var` and a variable named `MY-VAR` that are unique and unrelated to each other. This is similar to most languages such as C, C#, Java, Python, etc.

Most Interlisp primitives are in upper case.

As a side note, the Medley system includes a package called DWIM (Do What I Mean). This system reads in what you type and attempts to automatically correct input errors. At times, in an effort to correct typing errors this system will auto-convert something you type in lowercase into uppercase. Thus it may appear that the case doesn't matter - but it does.

5.2 Variables

Except for Special Variables, variables in Common Lisp are lexically scoped. This means that local variables are only visible within the scope they are defined. This means, among other things, that variables defined in one function are not visible to other functions.

In functions that are running interpretively (as opposed to having been compiled), variables in Interlisp are dynamically scoped. This means that variables are visible within the dynamic environment they are in. For example, let's say we create two functions `FUN1` and `FUN2`. If `FUN1` introduced a local variable and then called `FUN2`, then `FUN2` would have access to the variable since it is in the dynamic environment of being called by `FUN1`. In other words, the variable was in existence when `FUN2` was called. Variables declared in compiled functions are lexically scoped as in Common Lisp.

Common Lisp also supports dynamic variables as well. They are called the Special Variables.

5.3 LISP-2

Like Common Lisp but unlike Scheme, Interlisp is a LISP-2 language. This means, in part, that the namespace for variables is separate from the namespace for functions. For example, in Interlisp and Common Lisp, you can simultaneously have a variable named `ABC` and a function named `ABC` that are unrelated.

5.4 LAMBDA & NLAMBDA

While `lambda`s are the same between Interlisp and Common Lisp, Interlisp adds the notion of `nlambda`. An `nlambda` is a function that doesn't evaluate its arguments. Arguments to `nlambda` function are passed directly into a function without being evaluated.

Interlisp supports spread and no-spread lambda arguments just like Common Lisp. However, Interlisp treats all arguments as *optional* and ignores extra arguments.

5.5 Macros

Interlisp supports macros but unlike Common Lisp, Interlisp symbols may simultaneously have a function definition and a macro definition. If a symbol has both a function definition and a macro definition, the function definition is used by the interpreter and the compiled version is used by the compiler. This allows for extra error checking during development and fast operation during production use.

Interlisp also has a backquote facility similar to Common Lisp's ` and ,` read macros.

Unlike Common Lisp, Interlisp does not have a special function for defining macros. Macros are defined by placing their definition on the property list of the symbol.

6 Common Lisp

The Common Lisp currently supported by Medley is somewhere between CLtl1 and CLtl2. There is an ongoing effort to complete the move to CLtl2.

As mentioned previously, in Medley Common Lisp and Interlisp are fully integrated. From within Common Lisp, Interlisp functions may be accessed through the Common Lisp package nicknamed "IL".

7 Continuing On

This introduction was designed to provide the most general of information – just enough to get you started. Medley comes with extensive documentation. First, Medley includes a full primer at `medley/docs/primer`.

Medley also includes a full, online reference that is available through the Medley System as follows:

```
right-click on the desktop to get to the system menu
select DInfo
```

7.1 Useful Links

The following provides a variety of useful links to obtain Medley and to learn more about it.

- Medley Interlisp Project web site (`https://interlisp.org`)
- Source code to the Medley virtual machine (`https://github.com/Interlisp/maiko`)
- Source code to Medley lisp and runtime (`https://github.com/Interlisp/medley`)