Lessons Learned From Building General Mission Analysis Tool With MacPorts

Daniel Beatty

June 1, 2019

Abstract

Contents

Introduction

0.1 Concurrent Engineering Before

In the beginning of concurrent engineering, a new concept of telecommuting took baby steps towards manufacturing things collaborately over distance.

Another thing that benefits from the distributed object approach is tool, simulator, and in-loop design and testing. These all benefit from an architecture that contain:

- Agents
- Common Agent Language(s)
- Facilitators and translators
- Brokers
- Schedulers

These approaches supported development of jet engines and made it possible to get around proprietary issues. This concept also supports cube satelite to ask for services during its design and test.

We propose a new model that can corridinate of data center cloud computing, garage start ups, and industrial facilities. Each agent makes requests to local facilitators. Facilitators talk with each other, enough to advertise services. One service request can ask current star patterns and observations, like SDSS or other more recent acquistions. Another request may ask GMAT to determine orbit maneuvers.

Some services might live literally in the cloud. Others might live in a benefactor's garage. More should exist where we construct large parts. These service agents require architectures so that they may carry out their basic functions.

0.2 Package Managers

This work looks at how to build NASA's General Mission Analysis Tool (GMAT) in MacOS environment. In this case, we configure for MacOS Mo-

jave. Most UNIX environments depend on system development tools. Apple prefers application exist as self-contained entities, except for system library frameworks. Apple also prefers developer tools that incorporate signatures from the developer responsible.

In many UNIX environments, developer communities have built package managers. The Apple developer community also supports package managers for scientific and developer work. Some of these management systems include MacPorts, Fink, and Homebrew. MacPorts was designed to facilitate open source projects beneficial for Apple developer, education, and scientific communities.

Linux administrators utilize and prefer this method to provide their systems' capabilities. One can identify a distribution by its package distribution system. The Application Package Tool (APT) supports the Debian/Ubunto Linux Distributions. Yum favors Red Hat variants. The package manager called Yet Another Support Tool (YaST) supports SuSE

Later, other package distribution managers emerged to support families of projects. For example, Homebrew emerged to support Ruby on Rails and the grid/cloud projects produced under this banner. Homebrew and MacPorts share characteristics in common in terms of the Qt tools used to build cross platform tools like GMAT. Also, Homebrew supports both Mac and Linux platforms.

This work shares lessons learned from supporting these distribution managers. GMAT is a capability to simulate the performance of an orbiting platform. Provisioning these capabilities and integrating components served as a corner stone of concurrent engineering, especially within grid technology. Now, we need GMAT agents and

0.3 Concurrent Engineering from the Garage to Post-Cloud

Part I

Basic Build Dependencies

Chapter 1

What is a Package Manager?

With package managers, declaring dependencies matters. There are many misconceptions amongst the layman and seasoned engineers.

The make and cmake systems are build systems, not dependency managers. All build systems entail the ability to select system libraries and frameworks.

Some more sophisticated build systems that entail dependency management include Jenkins and XCode Build Server. Microsoft also produces tools for Visual Studio in server capacities. These build systems are designed to work with version control systems. The premise in these cases is to build part if not whole systems based on changes to (VCS) repository. These systems produce packages of developed code as products for distribution via the server. These services also corridinate build services over a cluster thereby enhancing build efficiency.

It is not uncommon for package management services to use distributed build systems/services to provide the packages for their "library of packages." It is also common to see package management services such as RPM, YaST, APT, MacPorts, or Homebrew to work with distributed build systems to ensure a known system during build.

The inherent goal of such a package system is to provide configurations that work together. Often, software packages depend on a common set of frameworks of libraries. In some cases, a major operating system places these frameworks in known loacations. The characteristics for a particular version of operating system then provide a baseline. This baseline allows a developer to engineer small codebase contribution weilding common elements from other providers. Most UNIX systems entail primary system libaries and a diverse cast of libraries provided by environment where they operate.

This work domain allows agencies and companies to specify service providing appliances for specific purposes. For example, Red Hat and SuSE Linux have been a mainstay of basic server operations for years. Scientific Linux inherits from its CERN and Fermi National Accelerator Laboratory (FNAL) contributions and focuses its packages on particle physics. NASA spent fair

amount of effort with virtual machine and container based cloud technologies to rapidly provision services for specific mission objectives.

Chapter 2

Common Issues on Apple

Homebrew and MacPorts run into similar issues in MacOS, typical past Mavericks. The most significant issue entails overriding primary system libraries. The LLVM compilers on Apple expect certain frameworks as dominant. This includes libraries for image and video.

2.1 System Dependencies

These libraries include dependencies upon dependencies: An error message such as 'Solving some "Symbol not found" problems for libJPEG, libTIFF, libGL, libPng'. The script listed in listing 2.5. Alternatives also exist as shown in [1]

Listing 2.1: Resetting libraries to stock for LLVM-MacOS.

```
#!/bin/sh
ln -sf /System/Library/Frameworks/OpenGL.framework/Versions/A/Libraries/libGL.dylib /opt/local/lib/libGL.d

ln -sf
/System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/ImageIO.framework/Versions/
        /opt/local/lib/libpng.dylib

ln -sf
/System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/ImageIO.framework/Versions/
        /opt/local/lib/libtiff.dylib

ln -sf /System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/ImageIO.framework/Ve
        /opt/local/lib/libjpeg.dylib
```

2.2 Installed Ports

The F2C found in MacPorts does not agree with NASA's GMAT either. Thus, this is a dependency that must be satisfied.

Another error that comes frequently is that cmake doesn't find wxWidgets, a cross-platform GUI library utilized by Python, Perl, Ruby, and its

[1] https://www.wxwidgets.org/blog/2014/07/easier-way-to-avoid-dependencies-on/

native C++. MacPorts as of this writing is at version 3.1.2 (current release). Therefore, we alter the CMakeList file at line 25 with this:

Listing 2.2: Where is cSPICE, starting settings in GMAT CMake

```
INCLUDE_DIRECTORIES(/opt/local/include)
set(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} /opt/local/lib)
set(CSPICE_INCLUDE_DIR /Users/danielbeatty/projects/BeanStalk/cubsat/Dependencies/CSPICE/cspice/i
set(CSPICE_DIR /Users/danielbeatty/projects/BeanStalk/cubsat/Dependencies/CSPICE/cspice)
```

Also, near line 123 is a portion searching for wxWidgets. It doesn't seem to make sense why cmake on GMAT doesn't find wxWidgets, even with settings point at where MacPorts puts it shown in 2.3.

Also, I put some debugging lines in the CMakeList file to hint at why wxWidgets and F2C were not so easily found. These are included here in listing 2.4

Listing 2.3: Where is wxWidgets, starting settings in GMAT CMake

```
/opt/local/Library/Frameworks/wxWidgets.framework/Versions/wxWidgets/3.1/bin)
set(ENV{PATH} "${SYS_PATH}:${wxWidgets_ROOT_DIR}")
set(wxWidgets_INCLUDE_DIRS
        /opt/local/Library/Frameworks/wxWidgets.framework/Versions/wxWidgets/3.1/include)
set(wxWidgets_LIBRARIES
        /opt/local/Library/Frameworks/wxWidgets.framework/Versions/wxWidgets/3.1/lib)
include_directories("${wxWidgets_ROOT_DIR}/include") # header files

MESSAGE(STATUS "Place of wxWidgets_LIBRARIES = " ${wxWidgets_LIBRARIES} )
MESSAGE(STATUS "Place of wxWidgets_INCLUDE_DIRS = " ${wxWidgets_INCLUDE_DIRS} )
MESSAGE(STATUS "Place of wxWidgets_ROOT_DIR = " ${wxWidgets_ROOT_DIR} )
```

Listing 2.4: Where is F2C, starting settings in GMAT CMake

```
# By default use the CSPICE version of F2C
SET(F2C_DEPENDS_DIR ${CSPICE_INCLUDE_DIR})
MESSAGE(STATUS "F2C Depends on CSPICE.  Is this the F2C required?" ${F2C_DEPENDS_DIR})

# Search for F2C folder in default paths
# User can change this via CMake GUI or command line
FIND_PATH(F2C_DIR
  NAMES f2c.h
  PATHS
    ${F2C_DEPENDS_DIR}
  DOC "Path to F2C directory (containing f2c.h)"
  NO_DEFAULT_PATH
  )

  MESSAGE(STATUS "F2C Directory variable set for " ${F2C_DIR})

# Make sure F2C exists
if(F2C_DIR)
  MESSAGE(STATUS "Found F2C: " ${F2C_DIR})
else()
  MESSAGE(WARNING "F2C NOT FOUND. Set F2C_DIR to path to f2c.h and re-configure.")
endif()
```

Remove MacPorts

```
 https://guide.macports.org/chunked/installing.macports.uninstalling.html
port -fp uninstall installed
rm -rf \
        /opt/local \
        /Applications/DarwinPorts \
```

```
        /Applications/MacPorts \
        /Library/LaunchDaemons/org.macports.* \
        /Library/Receipts/DarwinPorts*.pkg \
        /Library/Receipts/MacPorts*.pkg \
        /Library/StartupItems/DarwinPortsStartup \
        /Library/Tcl/darwinports1.0 \
        /Library/Tcl/macports1.0 \
        ~/.macports
```

Listing 2.5: Resetting libraries to stock for LLVM-MacOS.

```
#!/bin/sh
ln -sf /System/Library/Frameworks/OpenGL.framework/Versions/A/Libraries/libGL.dylib /usr/local/lib/libGL.d

ln -sf
/System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/ImageIO.framework/Versions/
        /usr/local/lib/libpng.dylib

ln -sf
/System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/ImageIO.framework/Versions/
        /usr/local/lib/libtiff.dylib

ln -sf /System/Library/Frameworks/ApplicationServices.framework/Versions/A/Frameworks/ImageIO.framework/Ve
        /usr/local/lib/libjpeg.dylib
```

2.3 Homebrew issues

The first step in putting the system in a Homebrew environment is to install Homebrew. Next, we install formulae that serve as dependencies for GMAT. There are a few missing from Homebrew. Notably, F2C does not appear in Homebrew's repository.

```
#!/bin/sh

/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Home


brew install cmake
brew install sqlite
brew install boost
brew install wxmac
brew install wxMaxima
brew install wxpython
brew install cairo
brew install xerces-c

#port -f install f2c libf2c +x86_64 -d
brew install p7zip
brew install curl

brew install pcre
brew install pcre2
brew install pcre++
# brew install ocaml-pcre
```

```
# port -f install cl-ppcre
```

Among some of the reasons to consider Homebrew include Ruby on Rails and SwiftObjects. If one considers the some of the Legacy approaches to GMAT, then methods that Service Oriented Architecture provides makes an opportunity to incorporate Concurrent Engineering.

To make Homebrew a better fit for GMAT, we need to establish package management records, called *f*ormulae, for the repository, called the *H*omebrew tap.

Amongst items not included is F2C. We can obtain this formula as shown in listing 2.6. This establishes the needed libraries. So, this is where CSPICE, F2C, and general libraries are identified to the *C*MakeList.txt file shown in listing **??**.

Listing 2.6: Setting up F2C in Homebrew Environment

```
brew tap FranklinChen/tap
brew install --HEAD franklinchen/tap/f2c
```

Listing 2.7: Where is cSPICE, starting settings in GMAT CMake

```
INCLUDE_DIRECTORIES(/opt/local/include)
set(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} /opt/local/lib)
set(CSPICE_INCLUDE_DIR /Users/danielbeatty/projects/BeanStalk/cubsat/Dependencies/CSPICE/cspice/i
set(CSPICE_DIR /Users/danielbeatty/projects/BeanStalk/cubsat/Dependencies/CSPICE/cspice)
```

After doing this, the builder discovers that Xerces is not up to par under Homebrew. So, we replace it with the source code from [2]. Now, GMAT configures correctly under CMake for Xcode, and Xcode builds all libraries and executables correctly. The big thing that produces errors is the install script as it tries to copy dependencies into the stand alone application directory.

[2] http://xerces.apache.org/xerces-c

Part II

Memoir Supported Part II

Chapter 3

Another Part, Another Chapter

Part III

Appendices go here

Appendix A

Why SOA Died in Concurrent Engineering

1 2 3

[1]https://en.wikipedia.org/wiki/Service-oriented_architecture

[2]http://jpmorgenthal.com/2009/06/19/the-reason-soa-isnt-delivering-sustainable-software/

[3]https://www.zdnet.com/article/bray-soa-too-complex-just-vendor-bs/

Appendix B

Because I am appended