Configure And Build

Åsmund Ødegård
Simula Research Laboratory AS
November, 2006

FEniCS'06

What we will talk about

- A rather short survey of available solutions
  - We only consider Open Source systems
- Some of the solutions applied to dolfin
- The currently selected solution for PyCC

What is the challenge?

- Installing software on peoples computer is the second thoughest challenge (HPL)
- You get 5 minutes attention span!
- Most people unfortunately run Windows, so you have to support that.
- Binary distribution isn't always feasible. Source distribution is hard.

What are really the main problems

- Developers want something powerfull, flexible, but still possible to comprehend.
- (Advanced) users should be able to figure out things when somethings goes wrong
- For plain users your system should simply works.
- Packaging - how should software be delivered to end-users?

On the more technical side

- Audit trails - what cause this flag in the build
- Hierarchical configure and build - namespaces
- Error messages should appear early!
- On the more philosophical side: How much of your system do you really need to inspect?
  - Pessimistic view: I really need to check everything
  - Optimistic view: I expect that my system works

There is no perfect system around!

There is no perfect system around!

Who are the main players in the field?

There is no perfect system around!

Who are the main players in the field?

- The ancient (in)famous Autotools

There is no perfect system around!

Who are the main players in the field?

- The ancient (in)famous Autotools
- CMake of kitware/VTK fame

There is no perfect system around!

Who are the main players in the field?

- The ancient (in)famous Autotools
- CMake of kitware/VTK fame
- SCons, grown out of the perl-based Cons

There is no perfect system around!

Who are the main players in the field?

- The ancient (in)famous Autotools
- CMake of kitware/VTK fame
- SCons, grown out of the perl-based Cons
- BuildSystem from PETSc, Matt's pet

There is no perfect system around!

Who are the main players in the field?

- The ancient (in)famous Autotools
- CMake of kitware/VTK fame
- SCons, grown out of the perl-based Cons
- BuildSystem from PETSc, Matt's pet
- Ant from the Apache world

There is no perfect system around!

Who are the main players in the field?

- The ancient (in)famous Autotools
- CMake of kitware/VTK fame
- SCons, grown out of the perl-based Cons
- BuildSystem from PETSc, Matt's pet
- Ant from the Apache world
- Jam, and variants like FTjam, Boost.jam

Autotools - no intro required!

- Autoconf, Automake, Libtool, and Gettext
- Generated "standard" makefiles
- Use make to actually do the build
- Not so nice syntax. Based on sh and the M4 macro language
- Backtrack on errors can be really hard.
- Produced make-files are really messy.
- The configure script in dolfin is 24627 lines.

CMake

- Developed by Kitware (VTK!)
- Selected by the KDE team recently
- Description files must be written in special CMake syntax.

CMake

```
project(DOLFIN)

set(CMAKE_CXX_FLAGS "-Wall -ansi")

# Configuration parameters

set(PETSC_DIR, /usr/local/lib/petsc)
option(ENABLE_PETSC, "Enable PETSc support")
option(ENABLE_CURSES, "Enable curses support")
option(ENABLE_DEBUG, "Turn on debugging and warnings")
option(ENABLE_BOOST, "Enable BOOST support")

if (ENABLE_DEBUG)
    add_definitions(-DDEBUG=1)
    set(CMAKE_CXX_FLAGS "-g ${CMAKE_CXX_FLAGS}")
endif (ENABLE_DEBUG)
```

CMake

- Developed by Kitware (VTK!)
- Selected by the KDE team recently
- Description files must be written in special CMake syntax.
- Simple but maybe not very powerful syntax

Features of CMake

- Strong on cross-platform development
- Can generate Makefiles, kdevelop projects and VisualStudio projects
- Multiple compilation trees possible from a single sourcetree.
- Mainly a configure tool?
- Quite mature! But lacks in documentation
- Large user community.
- Open Source

SCons - a quite different tool

- While its ancient roots is the Perl based Cons, SCons is implemented in Python
- Python also used as definition language
- Began as ScCons, which won SC build competiton in 2000
- Recently turned down by the KDE project
  - Actually, SCons combined with bksys, a wrapper created for KDE.
  - The bksys guy forked SCons into Waf recently.
- Replaces make!
- Different concept of signatures (MD5 vs. time stamp)

Features of SCons

- Strong on cross-platform development
- Supports for many languages, swig included
- Built-in support for CVS, Bitkeeper, Perforce
  - Unfortunately, no support for SVN and Mercurial
- Still beta, good for build, not configure
- Works with Python ≥ 1.5.2
- Smaller community.
- Open Source. Good documentation.

BuildSystem

- Someone here knows this much better!
- Implemented for PETSc?
- Primarily a configure system, output Makefiles
- Written in Python! Only requires Python to run.
- Supports version control systems (Bitkeeper, what else?)
- Open Source. Not so good documentation
- No community.
- It does its job, but have issues...

Apache Ant is a Java-based build tool

- In theory a replacement for make
- Use XML for descriptions/buildfiles
- Fully buzzword compliant?
- Open source, large community
- Cross plattform development

Jam and all it's variants

- The original Jam is from perforce
- Versions from freetype.org and Boost
- Main strength that it understands C/C++
- Can parse targets for #include to figure out what needs to be compiled
- All the variantes tell me it's not a perfect thing
- I really do not known Jam though. I have heard of enough people struggle with it.

What else have we

I have found numerous other tools mentioned:

What else have we

I have found numerous other tools mentioned:

- tmake/qmake
- MakeXS
- GConfigure
- buildtool
- package-framework
- MakeNG
- nmake
- Maven
- Waf - fork of SCons
- The commericial world?
- Probably other systems I don't know about....

Some experiments with Dolfin

- Arve Knudsen have done some experiments with SCons and CMake.
- Only very a very limited part of dolfin is considered in these experiments.
- No configure-tests are actually carried out
-

Dolfin and CMake

- Write a CMakeLists.txt in the root and eventually in subdirectories
- <show example>
- Run *cmake .* to create Makefiles in batch mode
- Run *cmake -i* to run CMake interactively. All settings defined in the CMakeLists.txt file, as well as some system variables can be modified
- CMakeLists.txt files are quite clean

Dolfin and SCons

- Write a SConstruct file in the root
- Write SConscript files in various subdirectories
- As SCons is not a preprocessor to make, the buildrules must be defined explicitly

- Run by saying *scons* in the root directory.
- A submodule can be compiled with *scons -u* in that directory.
- Central concept: the Environment
- <show an example>

Pros and Cons

- Pros - CMake
  - Clean syntax
  - Large user community
- Cons - CMake
  - Special macro language
  - Not much documentation available for free
  - Rely on make for
  - Looking into the CMake system for VTK doesn't give me great confidence

- Pros - SCons
  - Based on Python
  - Decent documentation
  - Support multiple build-environments (but not true hierarchical)
  - Can run under the regular python debugger (pdb)
- Cons - SCons
  - Lacks proper configure
  - Lot of relatively black magic
  - Build-files can be messy if not implemented with care

Making SCons work for us

- We are currently using SCons for PyCC.
- We have just started prototyping a configure system for SCons
- Our experience is that SCons feels more like a framework for building configure and build system, than a system in it self - at least for larger projects
- Maybe that is what we need, and maybe what SCons has is suitable for our needs

Offloading the work

- It is common that a configure system do all the searching for information about external dependencies.
- In our system, pkg-config will provide information about external dependencies (Include dirs, lib dirs, compile and link flags).
- We provide a pkg-config "generator" that do the searching if no pkg-config is found.
- We still have to test that the information pkg-config provides, is usable!

Simple description of submodules

- PyCC is organized in several submodules
- To ease the burden for the developer, we have implemented a simplified way of specifying sources and dependencies, in

```
Dependencies = ["numpy-1"]
LibSources = ["Conductivity.cpp", "FiberInterpolator.cpp"]
```

- A *swig* directory with a file *module.i* within the directory for the submodule will trigger building of a swig wrapper.
- We create a SCons build environment (*Environment*) for each submodule.
- There are other things as well that can be set in scons.cfg.

SCons and configure

- We are currently trying to prototype a separate configure step for SCons.
- It will use the scons.cfg files to figure out what is needed.
- pkg-config is used to pull in information.
- Using the build-system in SCons, tests can be carried out.
- All necessary information is stored using cPickle, and read when the user needs to build.
- This system is by no means ready for consumption.

Other slides

Cross language reference counting

Åsmund Ødegård, Ola Skavhaug, and others.

Cross language reference counting

Consider some python code where a python object is sent to some wrapped C/C++ function:

```
def somefunc(x):
  y = filter(x)
  c = SomeC++function(y)
  return c
c.doSomething() #seg.faults
```

Now, if the object ‘c’ use a pointer to data living inside y, we get a seg.fault when c.doSomething() is called, due to garbage collection.

A few words on the context

- Some C++ library
- Wrap the library with SWIG
- Use the library from Python
- We want something we can plug in with the SWIG typemap system
- From the C++ side it should be transparent whether an object has connections to Python or not, when it comes to ref. count

The FEniCS server

- As I am in charge of running the server, does anyone have any request?