MILANO CONFERENCE2013

gbuild:
State of the LibreOffice build system

- Michael Stahl, Red Hat, Inc.

- 2013-09-27

Overview

- The Historic OpenOffice.org Build System
- Goals for a Better Build System
- gbuild Architecture
- Migration
- gbuild New Features (since last year's conference)
- Future Work
- Lessons Learned

The Historic OpenOffice.org Build System (1)

- combination of `build.pl` / `deliver.pl` / `dmake`
- `dmake`:
  - conceptually similar to standard make but different syntax
  - OOo the only project using it
  - according to folklore `dmake` was selected in 90s because it was the only thing that worked on Mac OS
  - it's so obsolete it's licensed GPLv1 (!)
- `build.pl` / `deliver.pl`
  - homegrown Perl scripts...

The Historic OpenOffice.org Build System (2)

- `build.pl` iterates over all modules (top-level directories) & invokes `dmake` in each directory
  - obscure `build.lst` files
  - recursive make
    - (technically (almost) no recursion but morally equivalent)
    - *"Recursive Make Considered Harmful"*, Peter Miller, 1997
    - re -stat lots of files on every `dmake` invocation...
- all `dmakes` in module done: `build.pl` invokes `deliver.pl`
  - copies files listed in `d.lst` to `"solver"`
    - doesn't "solve" anything (Solar Version)
    - dumping ground for inter-module build

Example: OOo build (from scratch + run all tests)

- `./configure --enable-foo`
- `./bootstrap`
- `source LinuxX86-64Env.Set.sh`
- `cd smoketestoo_native`
- `Xephyr :42 &`
- `DISPLAY=:42 build --all -P2 -- -P2`
- `DISPLAY=:42 subsequenttests`

Example: OOo build (incremental)

- Let's do some change in vcl...

- `touch vcl/inc/vcl/window.hxx`
- `cd instsetoo_native`
- `build --prepare --from vcl`
- `build --all -P2 -- -P2`

Example: OOo build: clean a single module

- `cd module`
- `deliver -delete`
- `rm -rf $INPATH`

- (alternatively:)

- `cd module`
- `build --prepare --from module`

Example: OOo build: run subsequenttests in a module

- `cd module`
- `DISPLAY=:42 OOO_SUBSEQUENT_TESTS=t build -P2`

Goals for a Better Build System

- lean prerequisites
  - use standard tools
  - don't want to maintain another dmake
- full dependencies for incremental builds
- easy to use & reliable even for non-experts
- easier parallelism, less bottlenecks, better scalability
- less boilerplate in makefiles
- less "creativity" in makefiles
  - there should be one obvious way to to things
- automatically run tests during build
- … all of that with an incremental migration path

Goals for a Better Build System: LO perspective

- LO different from OOo and other OOo based projects:
  - Not large-corporation oriented, but community-oriented
  - *"Every time an incremental build fails a potential contributor is turned away from the project."*
- developers push directly to master, not to feature branches
  - low-level headers tend to change a lot
- incremental builds really have to “just work”!

Example:
current LO build (from scratch + running all tests)

- `./autogen.sh --enable-foo`
- `make check`

Example: LO build (incremental)

- Let's do some change in vcl...

- `touch vcl/inc/vcl/window.hxx`
- `make`

Example: LO build: clean a module

- `make module.clean`

Example:
LO build: run subsequenttests in a module

- `make module.subsequentcheck`

Example:
LO build: run subsequenttests in a module

- `make module.subsequentcheck`

- … and if it crashes you get a stack trace … automagically!
  - (except if you're unlucky and have to build on Windows... patches welcome)

Bonus Examples: LO build: debugging features

- Run tests in gdb:
  - `GDBCPPUNITTRACE="gdb --args" make`
- Run tests under Valgrind:
  - `VALGRIND=memcheck make module.check`
  - `VALGRIND=memcheck make module.subsequentcheck`
- Run `soffice` in gdb:
  - `make debugrun`

gbuild Architecture

- one GNU `make` process to build everything
  - but can also build single module
- based on GNU `make` 3.81+ features:
  - `eval`
  - target local variables
- one makefile per deliverable
- full dependencies
  - can be turned off (tinderbox, distro builds)

gbuild Files

- `solenv/gbuild`: core implementation
  - `solenv/gbuild/platform`: platform specific bits
- `Repository.mk`: define all linktargets/jars
- `RepositoryExternal.mk`: bundled external libs
- `RepositoryFixes.mk`: ugly hacks
- `RepositoryModule.mk`: for single process build
- `config_*.mk`: configure output
- `*/*.mk`: user makefiles

gbuild Implementation

- pseudo-OOP in GNU make

```
$(eval $(call gb_Class_method,instance,
 param...))
```

- `solenv/gbuild`: 12.5k lines of .mk
- `solenv/gbuild/platform`: 4k lines .mk + 100 lines .awk

- for comparison: `solenv/inc`: 25k of dmake

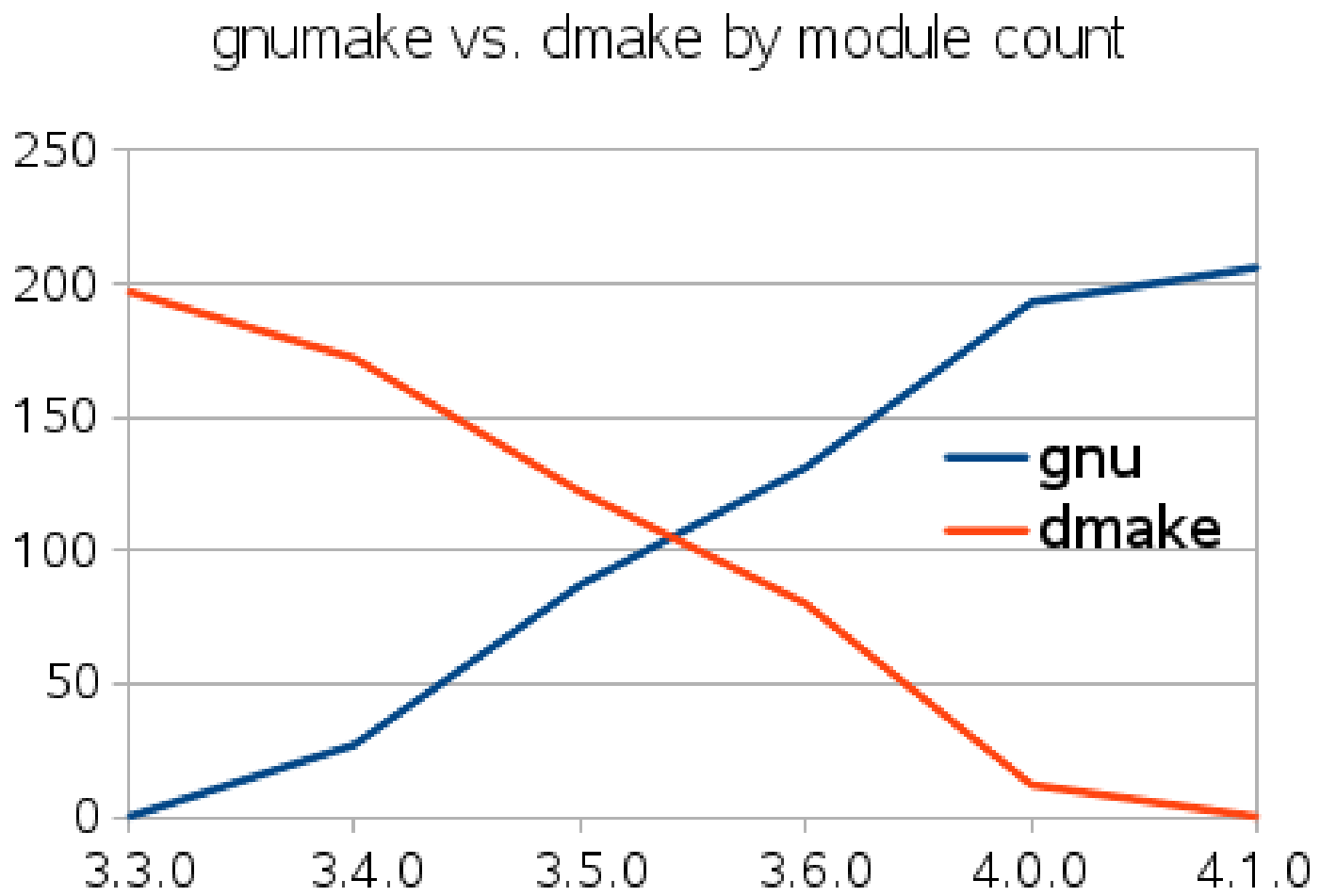
gbuild Old Features (already a year old)

- supports standard environment variables like `CPPFLAGS`, `CXXFLAGS`, `LDFLAGS`
- cross compilation support
- new platforms:
  - *BSD, Android, iOS, Solaris/GCC, MSVC2012, AIX
- mergedlibs
- check object owner
- `--enable-selective-debuginfo="sw/ svx/ xmloff/"`
- full dependencies for `svidl`, UNO IDL
- new targets: Asm, Yacc/Lex, Configuration, PyUno, Extension, Dictionary, Scp/InstallModule, Cli, ExternalProject, UI

Five Year Plan

1) get rid of `dmake` / `build.pl`
2) runnable installation: `instdir`
3) get rid of `solver`
4) shrink `scp2`

Incremental Migration (image from M.Meeks)



Migration completed

- http://skyfromme.wordpress.com/2013/02/28/one/
  - `dmake` / `build.pl` / `deliver.pl` dead and gone
- everything built by one GNU `make` process now
- converted last 40 modules [Peter, Matúš]
- new targets:
  - GeneratedPackage, PackageSet, AllLangPackage HelpTarget, AllLangHelp, ExternalExecutable [dtardon]
  - Gallery [mmeeks]
  - AutoInstall [Bjoern, Matúš]
  - PythonTest [David O]

gbuild is a Community Effort

- thanks to regular contributors:
  - David Tardon
  - Norbert Thiebaud
  - Matúš Kukan
  - Peter Foley
  - David Ostrovsky
  - Bjoern
- and many more than would fit on this slide

Improved build performance

- don't start build from scratch by writing out 10k empty object `.d` files [Bjoern]
  - saves 10 seconds on Linux
  - saves 10 minutes on Windows
- build included `.d` files as side-effect of target [mst]
  (saves a restart (only on successful build))
- reduce "`mkdir -p`" calls in rules [Matúš, Bjoern, mst]
- only re-link if library ABI (exported symbols) changes [mst]
  (idea from Ami Fischman of Chromium)

Python Test [David O]

- make it easier to write tests with less boilerplate
  - no annoying UNO `queryInterface` clutter
- make the tests easier to debug than JunitTest
  - run in-process
  - GDB can print python language stack
  - though not as easy as CppunitTests yet:
    - needs more GDB features like stack-frame filters
    - needs ability to set breakpoint in Python code
- working on Linux, Mac, Windows now
- converted a few JunitTests over

Windows improvements

- code signing [Fridrich]
- support MSVC 2010 / 2012 [David O, Peter]
- use debug runtimes with `--enable-dbgutil` [mst]
- use precompiled headers [Luboš]
- 64 bit (experimental) [Tor, Fridrich]
- no more `oowintool` [Peter]
- simple selection of MSVC version [Tor]
- GCC-wrapper for MSVC [Peter]
  - build bundled autotools using externals with MSVC

Mac OS X improvements

- support SDKs 10.6/10.7/10.8/10.9 [Tor]
- support building with clang / libc++ [Tor, Stephan]
- code signing [Tor]
- 64 bit (experimental) [Tor]
- WIP: Mac-like App structure [Tor]

misc features (1)

- config headers [Luboš]
  - `config_host/*.h.in`
  - generated by `configure.ac`
  - remove loads of -D from compiler command line, and actually force rebuilds on changes
- usability: user-friendly make targets [Luboš]

```
make CppunitTest_sw_macros_test
```

- clang compiler plugin support [Luboš]
  - extra warnings for misusing LO internal interfaces
  - simple code rewriter, already used

misc features (2)

- `BUILDDIR != SRCDIR` [Norbert]
- binary external tarballs [Norbert]
  - just unpack these and don't build
  - makes tinderboxes faster by 15 %
- `gb_Package_PRESTAGEDIR` [Bjoern]
  - provide a partial build result as a "cache" and re-use it
- `autodoc` replaced with `doxygen` [mst]
  - ~60k LOC `autodoc` replaced by 1k LOC of UNO IDL code in `doxygen`
- module dependency graph utility [mmeeks, David O]

Runnable Installation: `instdir`

- `instdir` [dtardon, Matúš, mst]
  - runnable LO installation, known to work on Linux, Windows, Mac
  - is updated simply by incremental build

    => faster “`make check`”
  - replacement for "`make dev-install`"
  - obsoletes the horrible "`linkoo`" hack

gbuild Current Work In Progress: kill `solver`

- `solver`: an anachronism
  - misleadingly named (Solar Version)
  - initially designed for partial builds: only check out a single module from CVS, build that against headers & libraries on NFS share
  - partial builds mostly obsolete with today's disk sizes
- entirely obsolete now, all files are in `instdir` and `workdir`

Storage Deduplication

- don't copy stuff pointlessly around
  - move all public headers to global `include/` dir [Bjoern]
    - no more `solver/*/inc`
    - copying headers may also break incremental builds
- use headers of externals directly from UnpackedTarball dir
- special case: zip removal [dtardon]
  - used to spend lot of time pointlessly zipping and un-zipping files

gbuild TODO: scp2

- scp2: defines contents of installation sets
  - duplicating a lot of conditionals that are already in makefiles
  - lots of boilerplate
  - own way to define library names
- do we still need this? can make do the job directly?

gbuild Current Work In Progress: `scp2`

- work ongoing to remove the duplicative file definitions
  - Package filelists [dtardon]
    - Package copies files to `instdir`
    - writes a list-of-files-file, reference it from `scp2`, installer looks up files in `instdir`
  - Auto-Installed LinkTargets [Bjoern]
    - register Library and Executable in `Repository.mk`,
    - then `scp2` entries are auto-generated
  - config files (unorc etc.) (“Profile”)
    - need to be written by a Makefile anyway for `instdir`

gbuild Current Work In Progress: scp2

- what parts of scp2 will survive?
  - there are things like
    - weird definitions for instset root-directories
    - module structure
    - Windows Registry entires
    - Windows Start menu entries
    - translated strings (.ulf files)
  - can this also be replaced? who knows...
  - if the top-level knows all the files that go into the instset then scp2 doesn't need to track files

Windows build performance

- Windows is slow
- Cygwin is slow
  - POSIX `stat()` call emulation, likely slow
  - `fork()` copies whole process memory
- we use Cygwin `make`
  - also has issues losing jobserver tokens
- can we use native Win32 GNU `make`?
  - reliable enough?
- (at least gbuild is faster than `dmake` based build system was)

Build now officially "ridiculously easy"

"The whole thing built. Without errors. I had working libreoffice debug binaries in six easy, well-documented steps.

That was amazing — it changed my mind about ***how much a project can improve its build experience if the developers really decide to prioritize it****."* – Karl Fogel

http://www.rants.org/2013/07/28/libreoffice_insanely_easy_build_process/

Parallelism:
never forget the N in “make -jN”

```
<tml__> whoa, the load average of my linux box is 372
<tml__> wonder what is going on
<mst___> tml__, accidentally ran "make -j"? hmm... but your box would be dead then
<tml__> hmm, I seem to have run PARALLELISM= nice make check
<tml__> which I guess means what you said;)
```

Lessons Learned: Namespace Pitfalls

- everything one make process => namespace problems!
  - variable names
    - target local variables not a problem
      - except if initialization forgotten :)
    - prefixes everywhere to avoid collisions
      - gbuild core variables prefixed with gb_
      - variables in user makefiles discouraged
      - user make file variables prefixed with module_
  - pattern rules
    - GNU make 3.81 vs. 3.82 pattern rules
      - some effort to support both

Lessons Learned: Performance

- unwanted parallelism:
  - do not want to link sw in parallel with sd, sc... on your laptop
  - workaround with artificial build order only deps
- portable shell good for performance:
  - dash is faster than bash

Lessons Learned: That Other OS

- Windows makes build system developers unhappy:
  - `make` bug 20033: `make 3.81 -jN` crashy
  - command line length limit
  - cygpath pain
    - finally required `make` with support for DOS paths
  - filesystem, process creation slow...

Lessons Learned: The Good

- full dependencies work!
  - quite simple to extend `svidl`, `idlc` to write make dependencies
- fast no-op builds
- most user makefiles relatively simple
- consistently use `DLLPUBLIC` annotations
- cleaned up cruft like `setsolar`, `set_soenv`... no more shell environment
- sane & consistent way to use external libraries which may be from system or bundled

Lessons Learned: The Not So Good (1)

- core gbuild implementation quite complex and difficult to understand
  - lots of function abstractions
  - `make` is not a very good programming language
  - *"migrating from obscure dmake system to a pile of impenetrable spaghetti masquerading as make files"*
- response files necessary to work around command line length limits on Windows:
  - fortunately `make` 4.0 has grown `$(file ...)` function
- cannot use cygwin's `make` package

Lessons Learned: The Not So Good (2)

- no checking of parameters when calling a function (or that function even exists)
- no multi-target build rules
  - used to work in `dmake`
  - GNU `make` rule can have multiple targets but is invoked once per target then :(
  - requires ugly `touch` rules
- inheritance of target local variables
- evaluating target local variable in dependencies
- bottleneck in parsing? parallelizable?

MILANO CONFERENCE2013

Thank you for listening

- Questions?