Future of the apps area software build system

Torre Wenaus, BNL/CERN
LCG Applications Area Manager

http://lcgapp.cern.ch

Applications Area Internal Review
October 20, 2003

- RTAG
- Evaluation
- SCRAM build system dev program, status, CMS manpower
- What to do, why, why not
- The prototype
- Plan, coming to a decision
- Timeline, project impact

RTAG on LCG Software Process Management

- *Excerpts from RTAG final report recommendations…*
- Supporting tools:
  - Build tool: GNUmake
  - Configuration management: CMT, SCRAM
  - Tools for managing platform-dependencies, packaging, exporting: autoconf, CMT, SCRAM/DAR, rpm, GRID install and export tool
- Specific free tools:
  - automake, autoconf, gmake
  - CMT, SCRAM (choose one, both appear to meet requirements)

SCRAM vs. CMT (vs. configure+make)

- As the RTAG pointed out, we have several related needs:
  - Configuration, dependency management
  - Software build
  - Packaging, distribution
- SCRAM and CMT both have capabilities in all these areas
- Approach a year ago, in line with the RTAG, was to select one or the other to address all these functions
- configure+make based build system was advocated by ALICE/ROOT
  - Not seriously considered as a viable third option to SCRAM, CMT for the build system
    - Mainly because it was not clear it could answer the need for a configuration system with adequate inter-package dependency handling
  - In retrospect probably a mistake on my part
  - configure+make support was, however, put in the SCRAM workplan

SCRAM/CMT Evaluation

- SCRAM vs. CMT technical evaluation done twice last year
  - Once by ATLAS, CMS experts (A. Undrus, I. Osbourne)
    - Recommended SCRAM, but complaints were made that 'CMT as used in ATLAS' was evaluated rather than CMT itself
  - Again by TW (and taking into account input received)
    - http://lcgapp.cern.ch/project/mgmt/scramcmt.html
    - Selected SCRAM
- Evaluation specified a development program required to bring the SCRAM build system up to requirements, all relating to the SCRAM build system, and to be carried out in an agreed collaboration with CMS:
  - Speed improvement
  - Support parallel make
  - Add more flexibility; e.g. one cpp ber binary application is limiting
  - Develop a "configure;make;make install layer"
  - **Windows support.**

SCRAM in LCG Today

- The SCRAM functionalities *other than* the build system have worked well, generate little fuss, and have been stable
- For the build system it is a different story
- Where are we with respect to the most important elements of the development program?
  - Windows support: SCRAM on Windows still not functional enough to build SEAL – instead CMT was used in the latest release, in order to meet urgent LHCb need for Windows builds
  - Build system speed: A no-op 'scram b' of POOL takes 6 minutes, a ridiculous length of time
  - Configure/make/make install layer: Nothing existing, nothing done; the available effort was consumed by the above items
- Exacerbating the bad situation with the build system and the incomplete work program is that the two CMS experts on SCRAM have effectively left the project

Where we are today

- SCRAM based infrastructure for configuration management, software builds, packaging and distribution
- A SCRAM build system that is today inadequate in important respects despite significant invested manpower
- On our own, essentially, in dealing with this deficiency

- We adopted a large & sophisticated (even if not the largest and most sophisticated) system
  - Requiring help/support outside the project
  - Resulting in the project competing for support attention with the priorities of a manpower-stressed experiment
- What we actually *need*, I believe, could be met by a *small* system that we could efficiently support and adapt
  - LCG applications area requirements are not so different from the open source community to reject without trial the standard open source approach
- Before diving into another big system like CMT with similar characteristics, or throwing more effort into SCRAM, I think we should prove or disprove a lighter weight approach based on standard open source tools
  - (And I do not mean adopting or re-re-implementing SRT)

Build system prototype

- So, I decided to see how far I could get with a quick prototype
- Objectives:
  - Understand whether we can leverage standard open source tools to build a **small** but solid system that meets our needs and is manpower-efficient
  - Build all of POOL, examples etc. (and later SEAL) with full support for hierarchical package level dependencies
  - Clean, maintainable makefiles and configuration information, not significantly more complicated than SCRAM
  - Work in conjunction with non-build parts of SCRAM (scram project …, release package version configuration from SCRAM)
  - Fast configure, very fast make
  - *User-friendly*, whether the user is an LCG developer, an experiment integrator or user, or an outsider trying out our software
    - Support typical developer environment in which packages in development come from local area, the rest from central release areas
    - Support also developer environments in which pieces of different projects (e.g. POOL and SEAL) are both in development

autoconf+make prototype – 'appwork'

- Started Oct 3, building on a prototype work environment 'appwork' providing SCRAM-independent, configure+make based usage of LCG software developed over several days in August
  - That version used a configure system derived from ROOT's
  - In going to a system to build the LCG software itself, autoconf was found to be better suited (more capable and maintainable)
- POOL was building on Oct 6 with hierarchical dependency support
  - Leaving of course lots left to do before we have a real system, but indicating we are not talking about a major development effort

Characteristics of the prototype

- Hierarchical dependencies resolved once, at configure time, not at build time, so make is fast (and the configure step is as well)
  - Configure step ('bootstrap') takes ~10sec
    - Dependency handling based on the recursive text substitution of m4, used by automake
      - Dependencies resolve into included makefile fragments
  - No-op make on POOL + examples takes ~7sec
    - Uses recursive make; plan to try and compare include-based single makefile, as long as package-level make can still be supported
- Make can be done at top level (POOL_x_x_x/src) or lower levels
- Bootstrap generates setup.csh for runtime environment setup
- Top 'appwork' level
  - provides general environment setup
  - will be the basis for supporting multiple development projects in one environment
    - e.g. concurrently modifying chunks of SEAL and POOL and developing an application using SEAL and POOL

BuildFile vs. Makefile

- Makefile similar to corresponding BuildFile
- Dependencies not expressed directly in the Makefile (at least not for those packages used by other packages), but in m4 based configure information
- Packages using package X, and package X itself, express the dependency and trigger the right .mk includes via "include $(lcg_use_X)"

```
================ BuildFile:
<include_path path=.></include_path>
<export autoexport=true>
  <lib name=lcg_PersistencySvc>
</export>
# Dependency on StorageSvc
<use name=StorageSvc></use>
# Dependency on FileCatalog
<use name=FileCatalog></use>
# This provides at least one plugin
PLUGIN_MODULE := PersistencySvc
```

```
================= Makefile:
include $(SRCTOPDIR)/config.mk
# Dependencies
include $(lcg_use_PersistencySvc)
# This provides at least one plugin
PLUGIN_MODULE := PersistencySvc
LIBS = liblcg_PersistencySvc.so
all: $(LIBS) plugin
```

m4 and mk code written for POOL, SEAL, external packages

```
-rw-r--r--    1 wenaus   zp            114 Oct  6 08:36 lcg_AthenaExampleDict.mk
-rw-r--r--    1 wenaus   zp             92 Oct  6 00:03 lcg_AttributeList.mk
-rw-r--r--    1 wenaus   zp             83 Oct  6 12:37 lcg_Collection.mk
-rw-r--r--    1 wenaus   zp             74 Oct  6 00:06 lcg_DataSvc.mk
-rw-r--r--    1 wenaus   zp            125 Oct  6 08:35 lcg_ExampleBase.mk
-rw-r--r--    1 wenaus   zp             82 Oct 15 00:57 lcg_ExampleDict.mk
-rw-r--r--    1 wenaus   zp             86 Oct  6 00:03 lcg_FileCatalog.mk
-rw-r--r--    1 wenaus   zp             77 Oct  5 23:47 lcg_POOLCore.mk
-rw-r--r--    1 wenaus   zp             95 Oct  6 00:05 lcg_PersistencySvc.mk
-rw-r--r--    1 wenaus   zp             92 Oct  5 23:25 lcg_PluginManager.mk
-rw-r--r--    1 wenaus   zp             82 Oct 15 00:57 lcg_PointerDict.mk
-rw-r--r--    1 wenaus   zp             76 Oct  6 08:32 lcg_PoolExamples.mk
-rw-r--r--    1 wenaus   zp             83 Oct  6 00:16 lcg_Reflection.mk
-rw-r--r--    1 wenaus   zp            104 Oct  6 00:15 lcg_ReflectionBuilder.mk
-rw-r--r--    1 wenaus   zp             77 Oct  5 23:26 lcg_SealBase.mk
-rw-r--r--    1 wenaus   zp             83 Oct  5 23:26 lcg_SealKernel.mk
-rw-r--r--    1 wenaus   zp             89 Oct  5 23:26 lcg_SealPlatform.mk
-rw-r--r--    1 wenaus   zp             83 Oct  6 00:04 lcg_StorageSvc.mk
-rw-r--r--    1 wenaus   zp            103 Oct 14 22:13 lcg_TinyEventModelDict.mk
-rw-r--r--    1 wenaus   zp            265 Oct  5 12:27 lcg_aida.m4
-rw-r--r--    1 wenaus   zp            505 Oct  5 12:21 lcg_boost.m4
-rw-r--r--    1 wenaus   zp            149 Oct  5 23:29 lcg_boost.mk
-rw-r--r--    1 wenaus   zp            511 Oct  5 13:23 lcg_bz2lib.m4
-rw-r--r--    1 wenaus   zp            513 Oct  5 12:14 lcg_clhep.m4
-rw-r--r--    1 wenaus   zp           3481 Oct 18 16:02 lcg_config.m4
-rw-r--r--    1 wenaus   zp            889 Oct 16 23:22 lcg_dependencies.m4
-rw-r--r--    1 wenaus   zp            614 Oct  6 00:44 lcg_edgrls.m4
-rw-r--r--    1 wenaus   zp            214 Oct  6 00:31 lcg_edgrls.mk
…etc
```

Expressing dependencies

```
============ lcg_pool.m4:
m4_define([req_PersistencySvc],[req_StorageSvc req_FileCatalog] AG_/lcg_PersistencySvc.mk)
m4_define([req_StorageSvc],[req_POOLCore req_ReflectionBuilder
     req_PluginManager] AG_/lcg_StorageSvc.mk)
m4_define([req_POOLCore],[req_SealKernel req_uuid] AG_/lcg_POOLCore.mk)
…
lcg_use_PersistencySvc="req_PersistencySvc"

============ lcg_seal.m4:
m4_define([req_SealKernel],[req_SealBase req_PluginManager req_boost]
     AG_/lcg_SealKernel.mk)
m4_define([req_ReflectionBuilder],[req_Reflection]
     AG_/lcg_ReflectionBuilder.mk)
m4_define([req_Reflection],AG_/lcg_Reflection.mk)
m4_define([req_PluginManager],[req_SealBase] AG_/lcg_PluginManager.mk)
```

- req_X is recursively expanded until it is a string of .mk files to be included via “include $(lcg_use_X)”

Typical makefile include fragment

```
============ lcg_POOLCore.mk:
ifneq ($(lcg_POOLCore),yes)
LDFLAGS += -llcg_POOLCore
lcg_POOLCore=yes
endif

============ lcg_uuid.mk:
ifneq ($(lcg_uuid),yes)
INCLUDES += -I$(UUIDINCDIR)
LDFLAGS += -I$(UUIDINCDIR) -L$(UUIDLIBDIR) -luuid
lcg_uuid=yes
endif
```

- Reentry protected because at the moment multiple instances of includes arising from redundant dependencies are not filtered (trivial to add)

Locating software

- Release configuration (package versions) obtained from SCRAM
  - Expressed (currently) in lcg_versions.m4

```
AC_DEFUN([LCG_VERSIONS],[dnl
AC_SUBST(lcg_ver_pool,POOL_1_3_0)
AC_SUBST(lcg_ver_pi,PI_0_4_1)
AC_SUBST(lcg_ver_seal,SEAL_1_1_0)
AC_SUBST(lcg_ver_gcc3,3.2)
AC_SUBST(lcg_ver_sockets,1.0)
AC_SUBST(lcg_ver_uuid,1.32)
…
```

- Software located using standard autoconf approach
  - Flexible; powerful; convenient site customization
  - Works well in conjunction with SPI's automated distribution tool

```
AC_DEFUN([LCG_MYSQL],[dnl
AC_REQUIRE([LCG_CONFIG])
MYSQLINCDIR=[$LCGEXT/mysql/$lcg_ver_mysql/$ARCHNAME/include]
AC_CHECK_FILE([$MYSQLINCDIR/mysql.h],,
[AC_MSG_ERROR([cannot find MySQL headers in $MYSQLINCDIR])])
AC_SUBST(MYSQLINCDIR)dnl
…
```

Using the prototype

```
tar xzf appwork.tar.gz
cd appwork
make
  (if the makefiles were in the repository you would do scram project POOL POOL_1_3_0)
  (or w/no SCRAM: mkdir POOL_1_3_0; cd POOL_1_3_0; echo "SCRAM_PROJECTNAME=POOL" > .SCRAM)
  (instead, there is a pre-installed POOL_1_3_0 made by 'scram project' with the makefiles)
cd POOL_1_3_0
  (if the makefiles were in the repository: cvs co -r POOL_1_3_0 src)
cd src
./bootstrap
./setup.csh
make
rehash
SimpleWriter.exe
```

- Build is done under POOL_1_3_0/build/rh73_gcc32/
- Libraries, exes installed under POOL_1_3_0/rh73_gcc32/

/afs/cern.ch/user/w/wenaus/public/appwork-dist/appwork.tar.gz

See README. At your own risk; not adequately tested yet

Must be run at CERN

ToDo on the prototype

- Test and refine the tarball distribution and developer/user environment
- Flexible control over what sw installations are/are not used in the environment
  - Use, or don't use, centrally installed releases, and which ones
  - Site-specific configurations
- Add POOL tests
- Extend to SEAL
- Support evaluations and bugfix, refine
- Add testing support
- Windows (cygwin)
- Try out include-based rather than recursive make
- …

Proposal

- Retain SCRAM for now but halt improvements on its build system
  - Given experience to date, and the CMS move, I think we can conclude that proceeding with the SCRAM build system is extremely unlikely to emerge as the optimal choice
  - The rest of SCRAM works stably and is the basis of our infrastructure
    - minimize disruption and keep it, at least unless/until cost vs. benefit drives us away
- Address the immediate needs of LHCb in the way begun in SEAL, adding CMT requirements files to POOL as well as SEAL and using CMT on Windows
- Continue to allow/support CMT requirements files independently of the in-house build tool decision
  - If it is manageable in terms of maintainability and manpower; assess this
  - Since we have, and will continue to have, two experiments using CMT
  - As part of our effort to better support experiment integration
- Evaluate, refine the autoconf+make prototype with a view to
  - Adopting this approach if it proves to meet our needs with a significantly simpler system than the other options
    - Projects should evaluate it, and its impact on them
  - Using it *at least* for an end-user layer, to avoid exposing SCRAM or CMT to users, if it does not prove to be the best choice for our in-house build system
  - Timescale TBD