Developing ARC code
A tutorial

David Cameron
University of Oslo/NeIC

NorduGrid 2013
Siauliai, Lithuania
4 June 2013
Disclaimer

- This tutorial is based on ARC v3.0.0 contained in release 13.02. Information may not be valid for other versions
- The tutorial is biased towards RHEL-based platforms, however most information is valid for other platforms
The code

- ARC source code is released under the Apache 2.0 license
  - http://www.apache.org/licenses/LICENSE-2.0
- Source code is hosted in a subversion repository
  - http://svn.nordugrid.org/
  - Most work is done in trunk
  - Branches for releases and major new developments
  - Tags per release
Supported Platforms

- Traditionally RHEL and Debian-related platforms have the strongest support
  - RHEL, SL(C), CentOS, Fedora, Debian, Ubuntu
  - These platforms always receive packages as part of release
  - Packages also exist in native repositories

- Nightly builds
  - http://download.nordugrid.org/builds/

- Most components can be built on Mac OS X

- Limited parts can be build on Windows (clients only)
  - (Rather outdated) packages are available for Mac and Windows

- Success has been reported on other platforms
  - OpenSUSE, SLES, Slackware

- Note: Features are generally limited by availability of 3rd party dependencies (see next)
• Even though we try to keep dependencies to a minimum, there are quite a few...
• The simplest way to install all necessary and optional dependencies (RPM-based platforms):
  – `yum-builddep nordugrid-arc`
• Note: package names may vary by platform
• Core dependencies (can't build anything without these):
  – `pkgconfig`, `openssl-devel`, `coreutils`, `libxml2-devel`, `xmlsec1-devel`, `xmlsec1-devel`, `libuuid-devel`, `libtool-ltdl-devel`, `glibmm24-devel`, `glib2-devel`, `gettext-devel`
• Nearly-core dependencies (can't build key services without these):
## External Dependencies

### Optional dependencies

- doxygen (API documentation)
- swig (java and python bindings)
- python-devel (python bindings)
- java (java bindings)
- nss-devel (Certificate handling using NSS)
- gridsite-devel (GACL-based GridFTP SE)
- python-twisted-web, python-twisted-core, pyOpenSSL (ACIX)
- cppunit-devel, pylint, junit (unit tests)
- xrootd-client-devel (xrootd data transfer protocol)
- lfc-devel (LCG File Catalog)
- gfal2-devel (Grid File Access Library for 3rd party copy)
Building the code

- ARC uses the GNU autotools framework
  - Very nice for portability but can be tricky to learn...
- configure.ac configures the build and adds options to ./configure
- In each directory a Makefile.am controls the build
- Typical example:

  ```
  lib_LTLIBRARIES = libarccrypto.la
  libarccrypto_ladir = $(pkgincludedir)/crypto
  libarccrypto_la_HEADERS = OpenSSL.h
  libarccrypto_la_SOURCES = OpenSSL.cpp
  libarccrypto_la_CXXFLAGS = -I$(top_srcdir)/include $(GLIBMM_CFLAGS) $(OPENSSL_CFLAGS) $(AM_CXXFLAGS)
  libarccrypto_la_LIBADD = $(top_builddir)/src/hed/libs/common/libarccommon.la $(GLIBMM_LIBS) $(GTHREAD_LIBS) $(OPENSSL_LIBS)
  libarccrypto_la_LDFLAGS = -version-info 2:0:0
  ```

- Only edit the Makefile.am, not Makefile.in or Makefile!
- It is better not even to look in the autotools-generated files, just let autotools work its magic
Build Sequence

- ./autogen.sh (only necessary after first checkout from svn)
- ./configure <options>
- make
- make test (to run unit tests)
- make install (as root depending on install location)

To create RPMs
- make dist
- cp nordugrid-arc-trunk.tar.gz <rpmbuilddir>/SOURCES
- cp nordugrid-arc.spec <rpmbuilddir>/SPECS
- cd <rpmbuilddir>/SPECS
- rpmbuild -ba nordugrid-arc.spec
Code Structure

- http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk
  - Core libraries + API
  - Plugins
  - Other libraries (DTR)
  - Services
  - CLI tools
  - Utilities/test code
  - Bindings for other languages
  - Building and packaging infrastructure
Languages

Total lines of code grouped by language:
- **cpp**: 163208 (72.91%)
- **perl**: 21300 (9.52%)
- **sh**: 18676 (8.34%)
- **php**: 15352 (6.86%)
- **python**: 4620 (2.06%)
- **ansic**: 656 (0.29%)
- **java**: 27 (0.01%)

(Generated using David A. Wheeler's 'SLOCCount'.)

(Not including generated python/Java bindings)
Development Guidelines

- After ~10 years development stability is the main aim
  - Current developers have rather conservative attitude :)
  - Disruptive changes or new external dependencies must be well justified!
- Some parts of the code have grown ugly over time
- New contributions are welcome
- No formal code review, but feedback is often given from svn commit emails
- Big or disruptive changes should be done in branches and approved before merging into trunk
Extending ARC

- The plugin architecture of ARC makes it easy to extend without changes to the core code
  - A new plugin can be added without re-compiling or re-installing code
- Plugins must follow certain rules
- Easiest way is to copy 'n' paste an existing plugin
How to add a new DMC

- A DMC plugin exists for each data access protocol
- Firstly create a subclass of DataPoint
  - DataPointDirect in this case

```cpp
#include <arc/data/DataPointDirect.h>

namespace Arc {

  class DataPointMyProtocol : public DataPointDirect {
    
  };

} // namespace Arc
```
How to add a new DMC

- Implement the required virtual methods

```cpp
class DataPointMyProtocol : public DataPointDirect {
public:
  // Constructor
  DataPointMyProtocol(const URL& url, const UserConfig& usercfg, PluginArgument* parg);
  // The following methods from DataPoint must be implemented
  virtual DataStatus Check(bool check_meta);
  virtual DataStatus Remove();
  virtual DataStatus CreateDirectory(bool with_parents=false);
  virtual DataStatus Stat(FileInfo& file, DataPoint::DataPointInfoType verb);
  virtual DataStatus List(std::list<FileInfo>& file, DataPoint::DataPointInfoType verb);

  DataPointMyProtocol(const URL& url, const UserConfig& usercfg, PluginArgument* parg)
    : DataPointDirect(url, usercfg, parg) {}

  DataStatus DataPointMyProtocol::Check(bool check_meta) { return DataStatus::Success; }
  DataStatus DataPointMyProtocol::Remove() { return DataStatus::Success; }
...
```
How to add a new DMC

- Add an Instance() method for the plugin loader to use when choosing a plugin
  - In this case return NULL unless the URL is of the form my://...

```cpp
class DataPointMyProtocol : public DataPointDirect {
public:
    // Instance is called by the DataPointPluginLoader to get the correct DMC
    // instance. If returns a DataPointMyProtocol if the URL is of the form my://
    // or NULL otherwise.
    static Plugin* Instance(PluginArgument *arg);
    ...
};

Plugin* DataPointMyProtocol::Instance(PluginArgument *arg) {
    DataPointPluginArgument *dmcarg = dynamic_cast<DataPointPluginArgument*>(arg);
    if (!dmcarg)
        return NULL;
    if (((const URL &)(*dmcarg)).Protocol() != "my")
        return NULL;
    return new DataPointMyProtocol(*dmcarg, *dmcarg, dmcarg);
}```
How to add a new DMC

- Finally register the plugin
  - Note: outside the Arc namespace!

```cpp
} // namespace Arc

// Add this plugin to the plugin descriptor table
Arc::PluginDescriptor ARC_PLUGINS_TABLE_NAME[] = {
    { "my", "HED:DMC", "My protocol", 0, &Arc::DataPointMyProtocol::Instance },
    { NULL, NULL, NULL, 0, NULL }
};
```
How to add a new DMC

- That's basically it for the code – just fill in the protocol-specific details
  - To include this DMC in ARC, copy the structure of other DMCs in src/hed/dmc
  - Remember to add any new dependencies in the Makefile.am and configure.ac
  - Any new directories in the tree must be added to the list at the end of configure.ac
  - Don't forget packaging! Spec files, Debian control files

- This full example is part of the API documentation:
  - http://www.nordugrid.org/documents/code/sdk/classArc_1_1DataPoint.html#details
How to add a Service

- As with plugins, adding a HED-based service is relatively straight-forward
  - The container and interface are already there, you just have to fill in the functionality
  - The easiest way again is to use a simple existing service as a template
- The following example is a simple “echo” service
How to add a Service

Firstly create a class derived from RegisteredService

```cpp
#include <arc/infosys/RegisteredService.h>
#include <arc/message/Message.h>
#include <arc/XMLNode.h>

class EchoService: public Arc::RegisteredService {

  public:
    /// Make a new EchoService. Sets up the process handler.
    EchoService(Arc::Config *cfg, Arc::PluginArgument* parg);
    /// Destroy the EchoService
    virtual ~EchoService();

    /// Main method called by HED when service is invoked.
    virtual Arc::MCC_Status process(Arc::Message &inmsg, Arc::Message &outmsg);

    /// Supplies information on the service for use in the information system.
    bool RegistrationCollector(Arc::XmlNode &doc);
};
```
How to add a new Service

- The process() method gets called when the service is contacted
  - Normally the incoming XML message is examined and action taken, then a response is sent in the outgoing XML message

- In our example service the request and response payloads look like

```xml
<EchoServiceRequest>
  <Say>hello, grid</Say>
</EchoServiceRequest>  
<EchoServiceResponse>
  <Hear>hello, grid</Hear>
</EchoServiceResponse>
```
How to add a new Service

- The process() method (simplified, no security)

```cpp
Arc::MCC_Status EchoService::process(Arc::Message &inmsg, Arc::Message &outmsg) {
    std::string method = inmsg.Attributes()->get("HTTP:METHOD");

    if (method == "POST") { // Only POST is supported
        Arc::PayloadSOAP* outpayload = new Arc::PayloadSOAP(ns);
        outpayload->Namespaces(ns);

        Arc::XMLNode op = inmsg->Child(0);
        if (MatchXMLName(op, "Say")
            std::string message((std::string)op["Say"]);
            // Construct the response message
            Arc::XMLNode resultelement = outmsg.NewChild("EchoServiceResponse");
            resultelement.NewChild("Hear") = message;
            return Arc::MCC_Status(Arc::STATUS_OK);
        }
    }
    return Arc::MCC_Status();
}
```
How to add a new Service

- As with plugins, the Service has to be registered to be picked up by the configuration

```cpp
Arc::PluginDescriptor ARC_PLUGINS_TABLE_NAME[] = {
    { "echoservice", "HED:SERVICE", NULL, 0, &EchoService::get_service },
    { NULL, NULL, NULL, 0, NULL }
};
```
How to add a new Service

- Services are configured by XML or INI internally translated to XML
- Started by the arched executable
  - Normally wrapped in an init script
- See example XML configurations for A-REX at http://svn.nordugrid.org/trac/nordugrid/browser/doc/trunk/examples/xml
- Using INI-style requires adding the service to src/hed/profiles/general/general.xml.in
- Configuration in ARC in general is messy at the moment and needs cleaned up (volunteers?)
Communications

- One wide e-mail list for general discussions (nordugrid-discuss@nordugrid.org)
- List for receiving svn notifications (nordugrid-svn@nordugrid.org)
- Bugs are managed by Bugzilla (http://bugzilla.nordugrid.org)
- NorduGrid web page contains “official” information and documentation (http://www.nordugrid.org)
- Wiki pages contain “work in progress” documentation and ideas (http://wiki.nordugrid.org)
- Most developers are on skype, and sit in a dedicated chatroom for development discussions / ranting / philosophical debates...
Lots of documentation exists for sys admins of ARC services and for external developers who want to use ARC
- http://www.nordugrid.org/documents/code/

But not so much for ARC developers themselves
- “The code is the documentation”
- Current developers are the best resource
Any questions?