Table of Contents

Task Force Client Harmoniation: Evaluation of Existing Clients

Authors

- Martin Skou Andersen
- Alvise Dorigo
- Björn Hagemeier

Introduction

This document is an evaluation of the existing clients for all middlewares in the compute area. It will mostly focus on the compute area. However, the existence of features from the data area will be noted.

Middleware Clients

The middlewares mentioned in the following sections are ordered alphabetically, the order does not express any preferences.

ARC

Classes in ARC

The ARC middleware builds on several sets of classes both utilized by clients and services. These set of classes are under one called HED (Hosting Environment Daemon). Despite the name, the classes can also be used client side. HED cover common, data handling, message handling/communication, module loading, security, info system classes among others. The data, message and security handler classes are general classes, which is extended by specific modules, e.g. for data handling SRM, gsiftp, http, etc., for message handling tls, http, gsi, etc., and for security handling x509, usernametoken etc. These modules are pluggable and can be used on the server as well as on the client side.

Documentation: The Hosting Environment of the ARC middleware
http://www.nordugrid.org/documents/ARCHED_article.pdf

libarcclient

libarcclient is the collection of general grid computing client classes into a library. It suppors job submission, resubmission and migration, job management in form of job querying, output retrieving, killing, cleaning, etc., and resource discovery and querying. On top of these general classes, middleware specialized modules exist for Classic ARC, WS ARC, CREAM and UNICORE. Additionally the library has job description translation capabilities, currently capable of translating between the JSDL, XRSL and JDL job description languages.

Since ARC services by concept are decentralized, resource brokering is carried out on the client side. In libarcclient when targets have been discovered brokering is carried out by matching the targets against the user specified job description, and the matching targets are then ranked according to the chosen custom pluggable broker, currently supporting random, fastest queue, benchmark and data location brokering.

All of the above mentioned classes are all written in C++, however to support third party developers, not necessary developing in C++, Java and Python bindings for libarcclient and most of the other classes exist. Also these classes and the language bindings are fully portable, working on Linux, Windows and Mac OS X.

Documentation: libarccliet - A Client Library for ARC
http://www.nordugrid.org/documents/client_technical.pdf

WS ARC compute client

This client is a command line client, and having all the features of libarcclient. It is also available on Linux, Windows and Mac OS X.

Documentation: ARC Clients: User's manual http://www.nordugrid.org/documents/arc-ui.pdf (WS ARC)

Classic ARC compute client

This client will be phased out in the near future. It is only able to interact against the classic ARC CE (Grid-Manager). Builds on the classic arclib library.

Documentation: ARC User Interface: User's manual http://www.nordugrid.org/documents/ui.pdf (Classic ARC)

Platforms

Component	Linux	Windows	MacOS X
libarcclient	Yes	Yes	Yes
WS ARC compute client	Yes	Yes	Yes
Classic ARC compute client	No	No	No

gLite

In gLite there're two set command line interfaces: the CREAM-CE CLI and the WMProxy CLI (also known as WMS-UI). The former is conceived to allow the user to perform direct submissions (and other job related operations) to the CE, the latter is to submit jobs to the resource broker (a.k.a. WMS, workload management system) that successively dispatches the jobs to the job requirements matching CEs.

Both CREAM CE and WMproxy services expose two (different) WebService interfaces. CREAM CE (that is conformant to WS-I specification) is implemented in Java (using the Axis framework) and WMProxy is implemented in C++ (using manual SOAP encapsulation into a HTTPS channel).

At the client side (CREAM CLI) the language for implementation is C++; the SOAP framework is gSOAP, with a plugin (gsoap-plugin, developed by gLite) for the encrypted communications. The gore details of the data structures, functions, service's faults and related hierarchy, code mapping to human readable messages, etc. (all of them autogenerated by the parsing of the WSDL interface of the CREAM service) are hidden by a small framework of classes that expose to the CLI developers a relatively simple C++ interface. This made the developer's work more easy than handling directly the autogenerated stuff.

The WMS-UI is implemented in C++ too basing on shared WMProxy C++ API.

At the moment the C++ CREAM-API are used by a C++ CLI for the direct interaction of the user with the CREAM-CE; by the ICE component (an element of the WMS that makes the workload manager able to submit to the new CREAM-CE), by Condor that aims at submit jobs to the CREAM-CE.

CREAM C++ APIs and CLI

The attached diagram (cliarch.pdf) outlines the architecture of the client side of CREAM (APIs and CLI). The CREAM operations are: JobRegister, JobStart, JobCancel, QueryEvent, JobInfo, JobList, JobStatus, JobLease, LeaseInfo, JobSuspend, JobResume, JobPurge, DelegateProxy, RenewProxy, ServiceInfo, EnableJobSubmission, DisableJobSubmission, AcceptNewJobSubmission. For each of them there'is a particular C++ class that can be built by a dedicated factory, and a related single command line program.

The documentation of CREAM C++ client APIs can be accessed here. A complete tutorial to build your own CREAM C++ CLI is here. It contains build-able examples, Makefile, all software dependencies needed on a SLC5 64bit (port to SLC4 and other glite supported platform, like CentOS, should be straightforward).

At the moment the platforms guaranteed to support build and run of the CREAM C++ CLI are SLC4/SL5. We're investigating how and if is possible to build by hand outside the etics workarea, in order to compile the source code on a different unix flavour.

On Snow Leopard (Mac OS X 64bit, 10.6.4), without using etics, building the APIs is possible but very hard; autoconf/configure must be bypassed most of the times, some source files the APIs depend on must be compiled by hand (even without using make) and takes very long, some source code (like c-ares) needed to be slightly modified. The linking phase is possible (still only by hand, issuing explicit g++ command line + options). But it produces executable that at runtime complains that some symbol is missing and finally crashes. The portability of the c++ code itself (APIs and CLI) is then verified; indeed they compile; but the complete build process produces exe that are not usable. Building everything outside a etics workarea is, in principle, possibile but it needs a lot of work.

Brokering capability

The CREAM CLI doesn't support any kind of brokering capabilty; the CLI simply sends a JDL (and its ISB if needed) to a specified CREAM CE URL.

The WMS-UI itself doesn't support any kind of brokering capabilty too; indeed it sends job to a real broker (workload manager, WM) that is responsible to mange the jobs, including the match-making of the best fitting destination CE resource; then the brokering capability is actually implemented at server side.

CREAM UI User Guide

CREAM-guide (from page 18)

WMS UI User Guide

gLite UI

Platforms

Component	Linux	Windows	MacOS X
CREAM C++ APIs and CLI	YES (SL4/SL5 both 32/64 bits)	NO	Compiles by hand; runtime problem with linking (still to understand)
WMS UI	YES (SL4/SL5 both 32/64 bits)	NO	NO

UNICORE

Internal Client Classes

The internal client classes of UNICORE are used by many components of the UNICORE ecosystem. Even servers rely on them, as they sometimes act as clients towards other servers. The client classes are fairly low level, most aspects of Grid access need to be dealt with explicitly. Also, as UNICORE 6 is based on WSRF services, a lot of the internals of WSRF are exposed by the interfaces. The client classes exist for all available UNICORE 6 services, including storage and workflow.

Documentation of the internal client classes (among others) can be found here:
http://unicore.eu/documentation/manuals/unicore6/unicorex/apidocs/

UNICORE Commandline Client (UCC)

UCC is a commandline client for UNICORE 6 services. It offers a set of basic commands (Run job, get output, transfer files, etc). In addition, it can be extended in various ways. Being a specific client for the UNICORE 6 middleware, all features offered by the UNICORE Atomic Services (UAS) can be accessed through this client.

Documentation for the UCC can be found at:
http://unicore.eu/documentation/manuals/unicore6/ucc/index.html

High Level API for Grid Applications ()

The High Level API for Grid Applications (HiLA) has been designed as an abstract interface for Grid resources, which can be implemented to access different middleware. So far, implementations for UNICORE 5 and UNICORE 6 as well as OGSA-BES services have been developed. HiLA's primary intention is to hide the complexity of existing low-level APIs behind and easy to understand concise facade. Also, security aspects are intended to be dealt with via configuration rather than explicitly dealing with it in the application code.

Being an abstract interface targeted at providing access to multiple middlewares, HiLA does not offer all functionality available in UNICORE 6 servers. HiLA can submit atomic jobs and do data handling using UNICORE's internal client classes. Implementations for OGSA-BES services and UNICORE 5 used to be available, but have not been ported to the HiLA 2.x branch yet. Most likely, only OGSA-BES would be ported, as UNICORE 5 is hardly used anymore.

HiLA documentation can be found at: http://unicore.eu/community/development/hila-reference.pdf

Brokering

Internal client classes

- these would allow to do any kind of brokering and they are also capable of sending work assignments to the service orchestrator, however this approach would be rather low level

UCC

- send jobs to service orchestrator
- do local matchmaking and assign in round robin fashion

HiLA

- does not do anything at the moment

Platforms

Component	Linux	Windows	MacOS X
Internal client classes	YES	YES	YES
UCC	YES	YES	YES
HiLA	YES	YES	YES

Summary

-- BjoernHagemeier - 09-Nov-2010

- cliarch.pdf: Simplified architecture of CREAM C++ APIs and CLI

- cliarch.pdf: Simplified architecture of CREAM C++ APIs and CLI

This topic: EMI > EmiJra1T2ComputeClientEvaluation
Topic revision: r14 - 2012-04-02 - BjornHagemeierExCern