European Middleware Initiative

EMI JRA1 Compute Client Consolidation and Harmonization

EMI DOCUMENT

Document identifier:	**EMI-DOC-JRA1-Client_Consolidation_v1.0.odt**
Date:	**09/05/2012**
Activity:	
Lead Partner:	**JUELICH**
Document status:	**Draft**
Document link:	

Abstract:
This document is a proposal for client consolidation in the compute area.

Editor:	Björn Hagemeier (JUELICH)
Contributors:	Alvise Dorigo (INFN PD)
	Andre Merzky (LSU)
	Zsombor Nagy (NIIFI)
	Zdeněk Šustr (CESNET)
	Antony Wilson (STFC)

Document Log

Issue	Date	Comment	Author / Partner
1	04/05/12	Initial import into document	Björn Hagemeier / JUELICH
2			
3			

Document Change Record

Issue	Item	Reason for Change
1		
2		
3		

TABLE OF CONTENTS

Table of Contents

1. SURVEY OF EXISTING CLIENTS

We conducted a survey of existing clients for compute and related interfaces to set the ground for the consolidation and harmonization activity. The effective date of the survey is the EMI-2 release due at the end of April 2012. The full survey can be found in the EMI wiki[1], and is summarized below.

1.1. LIST OF EXISTING CLIENTS

Clients can mean three different things: libraries, CLIs (Command Line Interfaces) or GUIs (Graphical User Interfaces). UNICORE has its **URC** (Unicore Rich Client) which is a GUI for accessing resources with Unicore Atomic Services interface, and other communities also developed different GUIs to access ARC and CREAM resources. But due to the lack of time and resources the GUIs are excluded from the following discussion.

Libraries

(UNICORE) **uas-client**	Java library providing access to Unicore Atomic Services.
(UNICORE) **bes-client**	Java library providing access to BES resources.
(UNICORE) **emi-es-client**	Java library providing access to EMI-ES resources.
(UNICORE) **HiLA**	Java library which provides a high-level uniform API on top of the previous three libraries.
(ARC) **libarcclient**	C++ library which provides a high-level API to access EMI-ES, BES, Classic ARC (gridftp), WS ARC (extended BES), and CREAM resources.
(CREAM) **EMI-ES library**	C++ library which provides a high-level API to access EMI-ES
SAGA	SAGA is a standardized API for developing distributed applications that can run on Grid and Cloud infrastructure. It has implementations in C++, Java and Python. None of them has support for EMI-ES currently.

Table 1: Available libraries in the compute area

1 https://twiki.cern.ch/twiki/bin/view/EMI/EmiJra1T2ComputeClientSurvey

CLIs

UCC	A production CLI using the **uas-client** and **bes-client** libraries.
HiLA Shell	A demo CLI which uses the **HiLA** library (thus capable of accessing EMI-ES resources)
ARC CLI	A production CLI using the **libarcclient** library (thus capable of accessing EMI-ES resources)
CREAM EMI-ES CLI	A prototype CLI using the **CREAM EMI-ES** library (thus capable of accessing EMI-ES resources)
CREAM CLI	A production CLI providing access to CREAM resources.
WMS CLI	A production CLI providing access to the WMProxy service.
L&B	A production CLI providing access to the Logging and Bookkeeping services. L&B is a monitoring tool and not for job submission or management.

Table 2: Available CLIs in the compute area

The L&B CLI has been considered here for reasons of completeness only. It will not be mentioned in any of our scenarios below, because support of EMI-ES in L&B would only make sense if the WMS would adopt EMI-ES at the same time.

1.2. FEATURES

All clients are capable of job submission and management, and they have support for moving input and output data. Only the UNICORE clients have strong workflow support. The ARC Client does brokering on the client side, whereas the UNICORE clients and the WMS clients send jobs to a brokering service. The JSDL job submission language is supported by all of the clients, and the ADL is of course supported by all of the EMI-ES-capable ones. The UCC supports UNICORE’s own JSON-based job descriptions and workflow descriptions. The ARC client has additional support for XRSL and JDL. The ARC client also has built-in support for handling credentials (creating proxies, communicating with MyProxy and SLCS services).

1.3. PLATFORMS

The UNICORE clients are written in Java, which makes them run on almost all platforms. The ARC client runs on all major platforms including Mac OS X and Windows. The CREAM, WMS and L&B clients run on specific versions of Linux only.

1.4. EXISTING SUPPORT FOR EMI-ES

The UNICORE developers have implemented a native client library for EMI-ES, and based on this a backend for the HiLA API has been implemented, which automatically exposes EMI-ES functionality in any code based on HiLA, e.g. the HiLA Shell.

The CREAM developers have implemented a gSOAP based prototype library and a CLI using it to access EMI-ES.

The ARC developers have implemented plugins for the libarcclient library to access EMI-ES resources, which made the current ARC CLI capable of submitting, managing and querying jobs on EMI-ES computing elements.

All three of the solutions above are considered as working but not mature. There are still issues with the specific interpretation of the EMI-ES specification. Some more work is needed to make all the client implementation be able to speak with all the server implementations and thus reach full interoperability.

The WMS and the L&B have no plans currently to include any EMI-ES support in the near future.

1.5. USERS

The HiLA library currently has only a handful of users, while hundreds use the UCC. The CREAM CLI and the WMS CLI are used by approximately a thousand users combined.. The CREAM EMI-ES client is new, has no user base. The ARC CLI is used by approximately a thousand users, and even more use the libarcclient indirectly via some application on top of it.

1.6. DEVELOPER EFFORTS AND POST-EMI PLANS

UCC and HiLA both has approximately 0.2 FTE development effort, and after EMI they plan to continue development as it was before EMI. The CREAM CLI and the WMS CLI both has 0.5 FTE development effort currently, the post-EMI plans are not clear yet. The ARC CLI and the libarcclient has now approximately 1.5 FTE development effort, and after EMI the NorduGrid collaboration will maintain it.

2. DEFINITIONS

API	**Application Programming Interface**. For the purpose of this document, we consider an API to be language-independent, i.e. the same can be implemented for multiple programming languages.
CLI	**Command Line Interface**. A user interface that a user employs to initiate certain actions in a system. The means of interaction is the command line.
GUI	**Graphical User Interface**. A user interface that allows users to interact with electronic devices with images rather than text commands[2].
library	A **library** is a set of functions implementing a given API. The API is considered to be the official interface of the library that is exposed to the users. Additional functionality that the library may use internally may be part of the library.

2 http://en.wikipedia.org/wiki/Graphical_user_interface

3. ASSUMPTIONS

No process should be followed without knowing its guiding principles. Some of these are mentioned here to inform the reader why certain decisions were taken or why some potentially obvious scenarios have been left out from the discussion.

A major restriction of the consolidation activity was that any proposal from this work must be sufficiently small, such that product teams will be able to implement it by the end of 2012.

Some of the scenarios we considered involved code-wrapping across programming languages. That is, C++ code would have been used from within Java code. This approach can be dangerous, as errors in the C++ code can result in fatal crashes of the Java Virtual Machine (JVM). Therefore, we decided to avoid scenarios involving this kind of code wrapping.

It was generally agreed upon by members of the task force that the scenarios proposed below must support API or library access to the middleware services. Thus, scenarios only providing CLI access to the middleware were considered insufficient and were not listed below.

In addition to this, a common API definition across multiple programming languages is favored over the plain provision of individual APIs for each language. Scenarios that did not provide a common API have also been ruled out right from the start.

There are strong indications that neither the UNICORE Rich Client (URC) nor the UNICORE Commandline Client (UCC) will integrate EMI-ES functionality. The reason for the URC not doing so is that has never been part of the EMI project and is purely developed and maintained outside of the project's scope. The UCC product team is very small and merely sufficient to maintain the client, but not to implement new features.

The HiLA Shell, which is part of EMI as a CLI and can actually access EMI-ES endpoints, has never been intended to be used in production. It was incepted as a demonstrator application to showcase the use of the HiLA API. Developers interested in using the HiLA API will find a comprehensive source of examples in the individual HiLA Shell commands’ implementations. In a similar way, the SAGA CLI has been intended for demonstration purposes only. Therefore, wecannot recommend its use for production environments at the moment. Substantial work would be required to enhance either one of the above mentioned CLIs to become production ready.

Besides the specific UNICORE components like the URC, we generally ruled out graphical clients as well as workflow functionality. This was done for several reasons. First of all, no GUI client product is part of the EMI project. Secondly, we considered it enough of a challenge to implement only the library and CLI clients until the end of 2012. Therefore, the above mentioned functionality is out of scope.

ARC middleware developers would like to settle down with only one interface for their Computing Element. The EMI-ES interface could become the interface of choice for them. This would result in eventually phasing out their GridFTP and WS (extended BES) based submission interfaces. This increases the sustainability of the EMI-ES support in the ARC CLI and library.

The current CREAM EMI-ES client is using gSOAP for handling web service communication, but gSOAP is not seen as a preferred option because of licensing problems of future versions and also because of the complexity of working with the generated code.

The part of the WMS responsible for submitting jobs to the CE, would happily adopt a C++ API for EMI-ES activity submission. This could be implemented by a library from the ARC middleware providers.

4. SCENARIOS

In the following paragraphs, we list scenarios that we consider valuable enough to be taken into consideration. We start with the most promising three scenarios that we named A, B, and C. Following that, we provide summaries of additional scenarios that we find less promising.

4.1. SCENARIO A: NEW CLIENT AND LIBRARIES BASED ON SAGA

CLI	API	C++ library	Java library	Python library
new based on the existing SAGA clients	SAGA specifica-tion	SAGA C++	jSAGA	Bliss (pure Python SAGA implemen-tation)

Description

In this scenario, the SAGA specification is used as the definition of the API. There are SAGA implementations for C++[3], Java[4] and Python[5], all of them implement the SAGA API, or at least some parts of if, and all of them use adaptor plugins to access services with different interfaces. They do not share these plugins, which are written respectively in C++, Java and Python. Thus, an EMI-ES plugin is written for each implementation. Then one of them will be chosen for the purposes of the CLI, and a new CLI is written using that library. There are existing command line clients which can be used as a starting point. Probably a Python CLI would be the easiest, fastest option.

Advantages

SAGA has a well-defined API fully described in a formal way, developed and refined for several years. It is also extensible, if there would be some EMI-ES specific requirements which are currently not supported by the SAGA API. It is implemented in C++, Java and also has a new pure Python implementation, so only the adaptor plugin has to be written (although a separate one for each language). The library provides several built-in functionalities, e.g. asynchronous calls, session handling, etc.. Adaptor development is relatively simple. The CLI using the SAGA library can use the existing adaptors too, e.g. BES, Globus, Condor, SSH, GSISSH, PBS, etc., and other existing tools using SAGA, like a pilot job framework or a MapReduce implementation, can then also use EMI-ES.

Disadvantages

The adaptors have to be separately written for all three languages. A completely new CLI needs to be written, possibly based on an already existing one. The SAGA abstraction does not support the full feature set of EMI-ES (e.g. no support for NotifyService, delegation is not straightforward), which means that some compromises or extensions are needed. The SAGA C++ code base is relatively heavy, has long-standing bugs, and depends on the Boost[6] framework, which according to the SAGA developers makes it very difficult to maintain and to port to different platforms. The new Python implementation (Bliss) is new and incomplete. The SAGA project has insufficient manpower to maintain, package or deploy a new adaptor, so that effort should be also partially provided by EMI, which makes it unclear what happens after the EMI project ends.

3 http://www.saga-project.org/
4 http://grid.in2p3.fr/jsaga/
5 https://github.com/saga-project/bliss
6 http://www.boost.org/

Consequences

In this scenario none of the existing CLIs and libraries can get new developments within EMI, all the effort is put into the EMI-ES SAGA adaptors and the new CLI, which will be offered as the official EMI libraries and CLI.

Scenario A: New client and libraries based on SAGA	New development allowed in project year 3
UCC	**no**
HiLA Shell	**no**
WMS CLI	**no**
CREAM CLI	**no**
ARC CLI	**no**
libarcclient	**no**
HiLA	**no**
SAGA adaptors	**yes**
New CLI	**yes**

4.2. SCENARIO B: THE HILA API AND THE ARC CLI

CLI	API	C++ library	Java library	Python library
ARC CLI	HiLA	new library or the libarcclient modified	HiLA	SWIG-wrapped C++

Description

In this scenario the ARC CLI is provided as the official EMI CLI. The API of HiLA is documented and HiLA itself is offered as the official Java library. In order to have a uniform API across languages either a new C++ library needs to be written which has the same API as HiLA, or the libarcclient needs to be changed or extended to support the same API. The Python library then can be created with wrapping the C++ library with SWIG (as it is done currently in libarcclient).

Advantages

The ARC CLI is a production-quality CLI with hundreds of users. It provides access to resources with the BES interface, and also to CREAM resources and to classic (gridftp) and WS (extended BES) ARC resources. It has credential handling and data staging capabilities. HiLA provides a native Java library with a simple and clean interface, and the same interface is provided by the C++ library. If the libarcclient is modified (instead of writing a new C++ library) then it provides all the functionality of the ARC CLI mentioned before. If a new C++ library is written, that could be more lightweight, easier to make stable and maintain than the libarcclient.

Disadvantages

The HiLA API does not support the full feature set of EMI-ES (e.g. no resource info, no bulk operations, no delegation), so compromises or extensions are necessary. In case of a new C++ library: introducing a new component; limited functionality compared to the libarcclient (the libarcclient already has capability to submit to EMI-ES plus several other interfaces, including data staging and credential handling), thus questionable sustainability. In case of modifying the libarcclient: the ARC CLI has to be modified too, it breaks backward compatibility, all the existing users of the libarcclient are forced to adapt their applications, at least if they do not want to or cannot stick to the already existing versions.

Consequences

In this scenario the following components can get new development: the ARC CLI (which is the official EMI CLI), the HiLA (which is the official Java library), the libarcclient (if it is modified to have the same API as the HiLA then it is the official C++ library which needs to be improved to provide stable EMI-ES access; if a new C++ library is written, the libarcclient still needs to get new development because the ARC CLI uses it to access the EMI-ES resources), and the new C++ library (in case of choosing that option).

Scenario B: The HiLA API and the ARC CLI	New development allowed in project year 3
UCC	**no**
HiLA Shell	**no**
WMS CLI	**no**
CREAM CLI	**no**
ARC CLI	**yes**
libarcclient	**yes**
HiLA	**yes**
new C++ library	**yes** (in case of a new C++ library)

4.3. SCENARIO C: THE LIBARCCLIENT API AND THE ARC CLI

CLI	API	C++ library	Java library	Python library
ARC CLI	libarcclient	libarcclient	new library or HiLA modified	SWIG-wrapped C++

Description

In this scenario the ARC CLI is provided as the official EMI CLI. The API of the libarcclient is documented and the libarcclient itself is offered as the official C++library. In order to have a uniform API across languages either a new Java library needs to be written which has the same API as the libarcclient, or the HiLA needs to be changed or extended to support the same API. The Python library can be created with wrapping the libarcclient with SWIG (as it is done currently in libarcclient).

Advantages

The ARC CLI is a production-quality CLI with hundreds of users. It provides access to resources with the BES interface, and also to CREAM resources and to classic (gridftp) and WS (extended BES) ARC resources. It has credential handling and data staging capabilities. The libarcclient is the library used by the ARC CLI to provide all these features. The API is uniform across languages. If the HiLA is modified (instead of writing a new Java library) then it provides additional functionality (e.g. accessing UAS resources). The HiLA is a relatively new component, it has less users than the libarcclient, so there is less resistance to changing it.

Disadvantages

The libarcclient API does not support the full feature set of EMI-ES (e.g. no fine-grained control of the resource and job information query, no support to pause an activity), so compromises or extensions are necessary. In case of a new Java library: introducing a new component; limited functionality compared to the HiLA (which already has capability to submit to EMI-ES plus other interfaces), thus questionable sustainability. In case of modifying the HiLA: it breaks backward compatibility, existing users of the library needs to adapt their applications.

Consequences

In this scenario the ARC CLI and the libarcclient components will get new development, and either HiLA or a new Java library depending on the decision.

Scenario C: The libarc-client API and the ARC CLI	New developments allowed in project year 3
UCC	**no**
HiLA Shell	**no**
WMS CLI	**no**
CREAM CLI	**no**
ARC CLI	**yes**
libarcclient	**yes**
HiLA	**yes** (in case of modifying it) **no** (in case of a new Java library)
new Java library	**yes** (in case of a new Java library)

4.4. OTHER SCENARIOS

Besides the three scenarios described above we considered about a dozen of others. Most of them were dropped based on the assumptions detailed in the previous section. But some of them need more explanation.

Scenario: new client, new API, new libraries

This scenario involves specifying a new API most suitable for EMI-ES, implementing libraries in C++ and Java (optionally in python), and writing a new CLI which uses one of the libraries. This solution can result in a lightweight and stable product having the full feature set of EMI-ES. But the effort seems too big, and considering the fact that the ARC client can also access EMI-ES resources, the number of potential users of this new client and library may not be big enough to make it sustainable.

Scenario: ARC CLI, new API, new libraries

In this scenario the ARC CLI is offered as the official EMI CLI, and none of the existing libraries are provided as official libraries, but a new API is specified and new libraries are written in both C++ and Java. This could result in a lightweight and stable library having the full feature set of EMI-ES. The sustainability of this solution highly depends on the libarcclient and HiLA adopting the new libraries in their EMI-ES plugins. Considering the fact that both of these existing libraries already have EMI-ES plugins, it seems hard to justify why they would adopt the new libraries, which may make the number of potential users of the new libraries too small to make it sustainable.

Scenario: ARC CLI, new API, adapting the existing libraries

This scenario is similar to the previous in that it offers the ARC CLI as the official CLI, and it also defines a new API, but instead of writing new libraries, here both the existing libraries are adapted to the new API. This could mean changing them, extending them (while keeping the old API) or writing thin adaptors which translate the calls of the new API to the calls to the existing libraries. In each case this seems to be a much bigger effort than Scenario B and C described above. Whereas it has the advantage of a new API more suited to the EMI-ES needs, it will still be impossible to provide this scenario without changing existing libraries. This is because existing libraries may not provide all features of EMI-ES and thus will have to be adapted to the new use cases. This makes Scenario B or C preferred to this one.

5. API COMPARISON

In the following, you will find a detailed comparison of EMI-ES operations and methods available in the existing client libraries. Five client implementations and APIs have been compared to each other.

- Arc::EMIESClient (internal class used by libarcclient)
- libarcclient
- HiLA API
- SAGA
- gLite ES C++ Client Library

All of these but SAGA can be understood as library implementations, i.e. they implement the actual functionality and can access EMI-ES services today. SAGA is an abstract API for which multiple adaptors are available. Due to the lack of an adaptor for EMI-ES in SAGA, we rather considered if and how SAGA concepts can be mapped onto EMI-ES, which would be made available as a new adaptor.

5.1. SUPPORTED OPERATIONS

In order to provide the information contained in the following sections in a rather condensed manner, we will first describe which operations are generally supported and then go on to the more problematic ones that need attention in one or several of the implementations.

The operations mostly supported by all implementations are:

- CreateActivity
- GetActivityStatus
- GetActivityInfo
- PauseActivity
- ResumeActivity
- CancelActivity
- WipeActivity
- ListActivities

We explicitly state ‘mostly supported’ here, as many of the existing interfaces are not defined for bulk operations as supported by EMI-ES. Therefore, most APIs only support the invocation of these operations on individual activities. The SAGA API is an exception, as it defines bulk operations. These would need to implemented accordingly in an EMI-ES adaptor. The libarcdata API also supports bulk operation, however it is not yet implemented internally.

5.2. PROBLEMATIC OPERATIONS

This section documents those operations that are problematic in that not all current client implementation implement them.

GetResourceInfo

Whereas HiLA and the gLite ES C++ client library do not have any support for GetResourceInfo, SAGA and the ARC libraries support this operation.

QueryResourceInfo

QueryResourceInfo is supported only by SAGA by means of the ServiceDiscovery API and InformationServiceNavigator API. As described above, this support is potential only, as it would need to be mapped to concrete queries by an adaptor.

NotifyService

This is implemented by all but SAGA. The SAGA API does not provide this. The libarcclient implementation does this automatically during job submission, i.e. the user does not have any means to manually upload any additional data before notifying the service.

RestartActivity

This operation has only been implemented by Arc::EMIESClient and the gLite ES C++ Client library. The other client implementation do not support it.

InitDelegation, PutDelegation, GetDelegationInfo

The operations are fully covered by the ARC clients, which implement them as part of the submission process. The user does not explicitly init, put, or get a delegation. It is also covered by the gLite ES C++ Client library. SAGA and HiLA would implement this functionality in the same way ARC did it, i.e. making the sequence of init and put delegation part of the submission process. As a matter of fact, the library that has been used to implement the HiLA EMI-ES plugin does already provide this functionality.

6. RECOMMENDATION

After careful evaluation of the available options, and taking into consideration all boundary conditions, task force members have come to the conclusion that "**Scenario C: The libarcclient API and the ARC CLI**" as described above is the most promising one.

The scenario involves the ARC CLI, libarcclient API and library. The API will also be adopted for the Java programming language.

This will have the following consequences for the products:

- The UNICORE Commandline Client (UCC) will only be maintained during the remaining time of the EMI project. No new developments, in particular those related to the adoption of EMI-ES, will be accepted.
- HiLA Shell will not receive any new developments either. It will be maintained during the course of EMI. Naturally, if the HiLA library receives new developments, it is expected to take little effort to follow these changes in HiLA Shell. Thus, the additional features would become available.
- In case the decision for the Java library targets a new one, this will have to be developed. In that case, the HiLA library will not receive any new developments, but merely be maintained.
- The ARC CLI, which is production ready and widely used, will follow the implementation of EMI-ES in libarcclient and be adapted as necessary to offer all features of EMI-ES. It will be maintained and new developments are possible.
- The CREAM and WMS clients will not receive any new development, but merely be maintained.