- ♦ What can they do?
- ♦ Unification?

Command Line Client (CLI)

Current command line clients support a number of features. These should roughly be available in any harmonized or consolidated client, such that users can easily pick up how to use the new clients. The essential question in this aspect is whether to integrate the functionality or access EMI-ES in all existing middleware clients or to write a new middleware client allowing access to EMI-ES, thus consolidating functionality in a single client.

As an addition, functionality to access EMI-ES could be integrated in the existing clients, but we do not see this as a goal for EMI. The integration in the existing clients would also depend on the decisions regarding the client library (see below).

- Features of the client
  - ♦ submission
  - ♦ job monitoring
  - ♦ match-making
  - ♦ file staging
  - ♦ ...
- How to implement
  - ♦ Single client for all EMI-ES implementations
  - ♦ vs. Integrate functionality in existing clients
  - ♦ We would like a **PTB decision** about whether it is acceptable to only offer a client accessing EMI-ES or whether we need something more, i.e. accessing all previously existing services.
  - ♦ ARC library is generic and can be used in a sense as a client library for EMI-ES
  - ♦ The HiLA approach is similar
  - ♦ We intend to evaluate existing generic logic in ARC and HiLA for which one to use

We will discuss the individual proposed solutions in the following sections.

Single EMI-ES command line client

Pros

1. New users, who have not used any Grid middleware and client before, would only have to learn a single interface in order to access all EMI middleware, making an even larger set of resources available to them.
2. EMI-ES contain the essential Grid interfaces and is thus clean and lean. It is not laden with legacy functions that are only needed rarely. The single EMI-ES client would make this visible.
3. Reduction of active code, as only the single client would need to be maintained. This is one of the key EMI targets.
4. Some clients are currently available on multiple platforms, while others are only available on selected platforms. Starting with a new client, it would be possible to aim for platform independence, thus allowing users to use the platform of their choice rather than the developer's choice.

Cons

1. A lot of effort has been spent over years to develop the existing clients. They do their tasks and the do them well.
2. Users are used to use the existing clients and know their interfaces in terms of options and arguments and their use.

3. Users may have developed scripts that use the existing clients and therefore abandoning the existing clients would lead to user's disappointment. Users would have to adapt their scripts if they want to use the new single EMI command line client and its potential new features.

Consequences

The existing command line clients would not be developed further within the scope of EMI. An exception to this would be if one of the existing clients was chosen for the single client, e.g. because it is already implemented in a generic fashion, such that it can support the EMI-ES access with only a new backend, i.e. changes invisible to the users.

Integration of EMI-ES functionality in existing clients

Pros

1. Users would be able to keep using the clients they've always been using with the additional benefit of being able to access other additional middleware through the EMI-ES interfaces.
2. Less initial overhead as new new client framework would have to be implemented.

Cons

1. This approach requires at least three (ARC, gLite, UNICORE) clients to be maintained, not leading to a reduction of code. On the contrary, code will even increase, as the EMI-ES client library, which we assume will be developed, will have to be integrated in multiple clients.
2. Users may be distracted by the additional features of their clients, not knowing what to do with them.
3. There may even be potential conflicts within the clients, e.g. when the existing client so far required a library that conflicts with a library required for accessing the EMI-ES interfaces.
4. The uptake of the new EMI-ES functionality would be hampered, making it badly accepted right from the start.

Consequences

All existing clients would be developed further within the scope of EMI. Users could keep using the clients they are used to and get the additional benefit of being able to use other middleware through the EMI-ES interface.

Client Library (API)

From the point of view of the client library, the question is a little more difficult to answer. Whereas end users do not care much in which programming language their clients are written, developers are more limited in this regard. They want to use their problems in the language they use every day.

One interesting approach to solve the difficulty of choosing the right library is SAGA, the Simple API for Grid Applications. It defines abstract interfaces independent from the programming language and provides language bindings for various languages. There are two benefits in this. The interfaces for different Grid implementations are abstract and thus provide access to different implementations. Additionally, with the interfaces being defined independent from the programming language, the switch from one language to another becomse quite easy.

On the other hand, we found SAGA not to be very wide spread and the implementation of SAGA adaptors for EMI-ES may become difficult, as they would have to be maintained for each implementation of the SAGA specification, which is at least one for each of the supported programming languages.

At the same time, there is already a similar approach available from the ARC middleware. The API is defined in an abstract manner and multiple programming languages are supported via an automated language wrapper.

1. SAGA or not SAGA: by now I'm in favor of not using SAGA, as it doesn't seem to be wide-spread.
2. If not SAGA, what else? Martin suggested reworking of existing libraries.
3. From whose point of view should the requirements be driven? Martin wrote an interesting email on February 3rd (see below).

   From the EMI point of view, as I see it, it would be to have one CLI and library (API) supporting all the Execution Services in EMI. With the EMI-ES agreement, this should be a CLI and library able to interact with an EMI-ES. That is only the minimal solution, since it could be extended to support the full features of the respective ESs in EMI.

   From the developers point of view: Since the compute functionality, which is to be supported in the end product, is already existing in any of the 3 compute clients/APIs, the goal would be to reuse functional stable code, which has proved its worth. This of course means that some components/parts must be dropped (3->1).

   From existing users and third party developers* point of view, which I think is the most important: Here I am only concerned about existing users and third party developers, and in this respect users are using the CLI, while third party developers are using the API. Users would expect a CLI to be similar to the one they are already using, and it should at least deliver the same level of functionality. Users can be convinced to use a different CLI, but only if it does not limit their functionality. The same with third party developers. Additionally they would also expect some backwards compatibility, and concepts not changing too much.

   * Users and third party developers already using any of the 3 compute area CLIs/APIs.

   For me these three points of view cannot be unified.

The following questions should be answered next:

- Compare the capabilities of current client libraries in a similar manner as we have done for the command line interfaces.
- Consider if a common CLI implementation would base on a unified library.
- Which programming languages need to be supported? Evaluation of existing libs will help this decision.
- Try and avoid to add more functionality to client or libraries as a first rule in order not to overload it.

Job Description Languages

Current clients and client libraries support a variety of job description languages

- JSDL
- Globus RSL
- gLite JDL
- UNICORE Job Description based on Json

These job description languages already need to be mapped to the job description that the interfaces of the middleware services consume. In the case of the UNICORE Json based job description, for example, the services consume JSDL, so the Json notation gets mapped to JSDL.

The EMI-ES defines yet another job description language, which the services consume. From our point of view, job descriptions in clients and the job description consumed by services may differ and even should.

XML based job description languages are very complex and not exactly user friendly. Sometimes they contain information that is required by the language specification, but not even known to the user nor relevant. In such cases, the user would prefer just a simple format stating exactly what he requires, no more, no less. This can be demonstrated with the UNICORE Json notation:

```
{

Executable
        "/bin/date"
}
```

which the UCC then maps into JSDL as consumed by the UNICORE services

```
<jsdl:JobDescription>
  <jsdl:Application>
    <jsdl1:POSIXApplication xmlns:jsdl1="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
      <jsdl1:Executable>/bin/date</jsdl1:Executable>
    </jsdl1:POSIXApplication>
  </jsdl:Application>
</jsdl:JobDescription>
```

Documentation about this format can be found at http://www.unicore.eu/documentation/manuals/unicore6/ucc/jobdescription.html. One can say in general that for each single line of Json notation, several lines of XML get added to the JSDL. The Json notation is also more "forgiving" to the user than XML. Minor syntax errors don't make everything fail.

In summary, we can state that mappings of user friendly job descriptions to job descriptions consumed by services are already in use and we should not force users to abandon job descriptions they are already familiar with. These mappings, as far as they don't exist yet or have not been implemented, would be developed as part of the client APIs in the respective programming languages.

Next steps

- CLI
  - Evaluation of ARC and HiLA abstract libraries and the clients based on them
  - Comparison of available commands in the existing clients
- Libs
  - Look at capabilities of current libraries and compare them similar to CLIs

-- BjoernHagemeier - 18-Mar-2011

This topic: EMI > EmiJra1T2ComputeClientProposal
Topic revision: r9 - 2012-04-02 - BjornHagemeierExCern