ALICE OFFLINE WEEK, 3.10.2005

"The AliEn 2-ROOT-API"

Andreas-Joachim Peters CERN

Enabling Grids for E-science in Europe

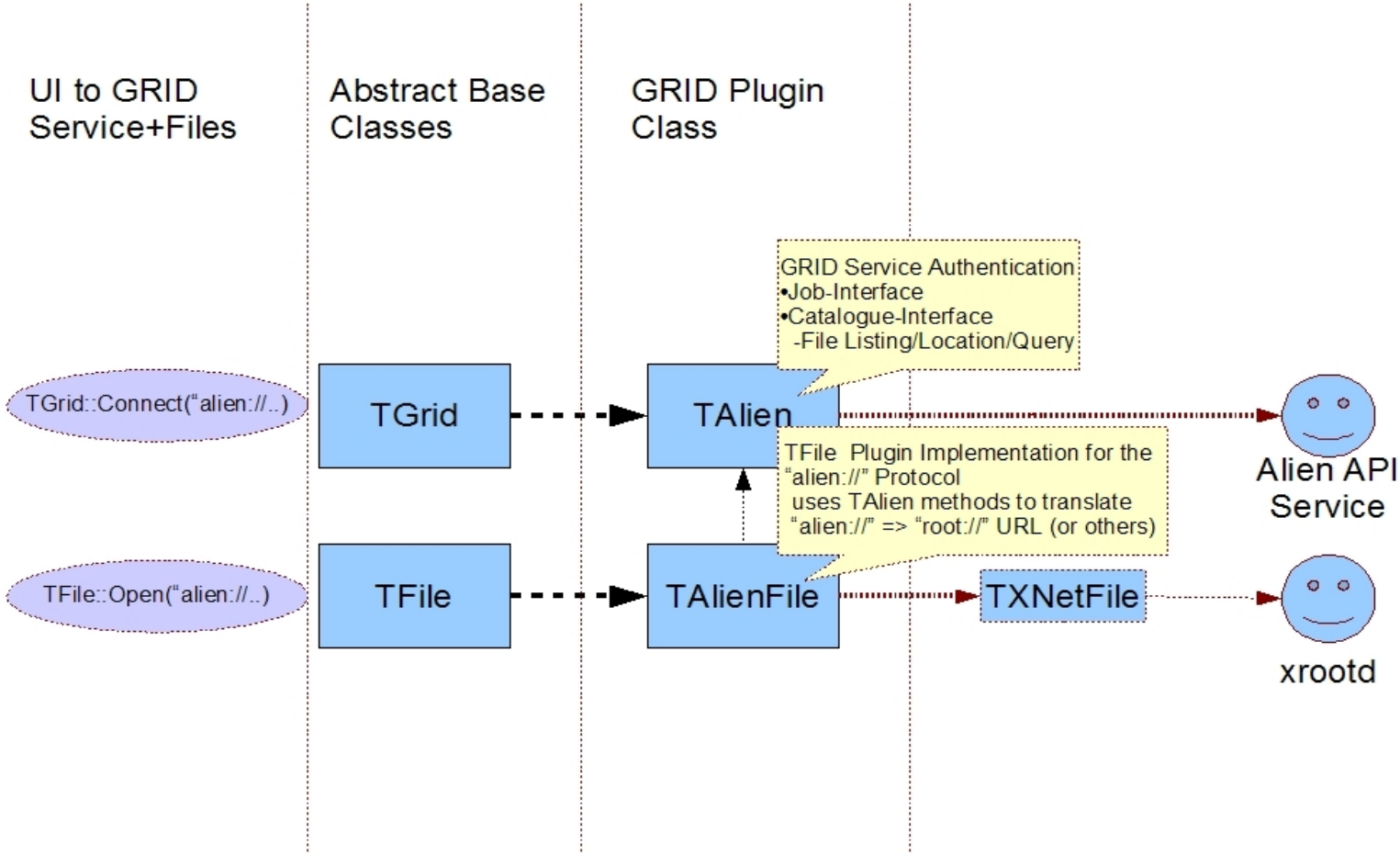
www.eu-egee.org

cern.ch/lcg

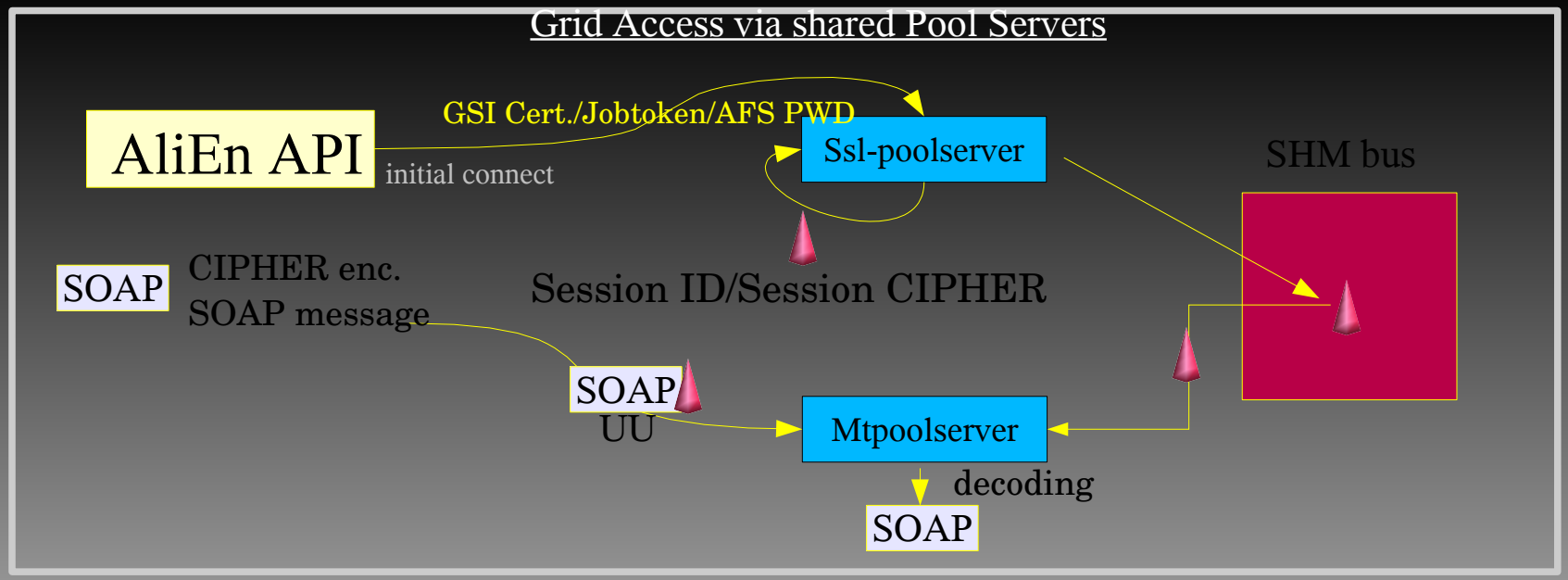
http://cern.ch/arda

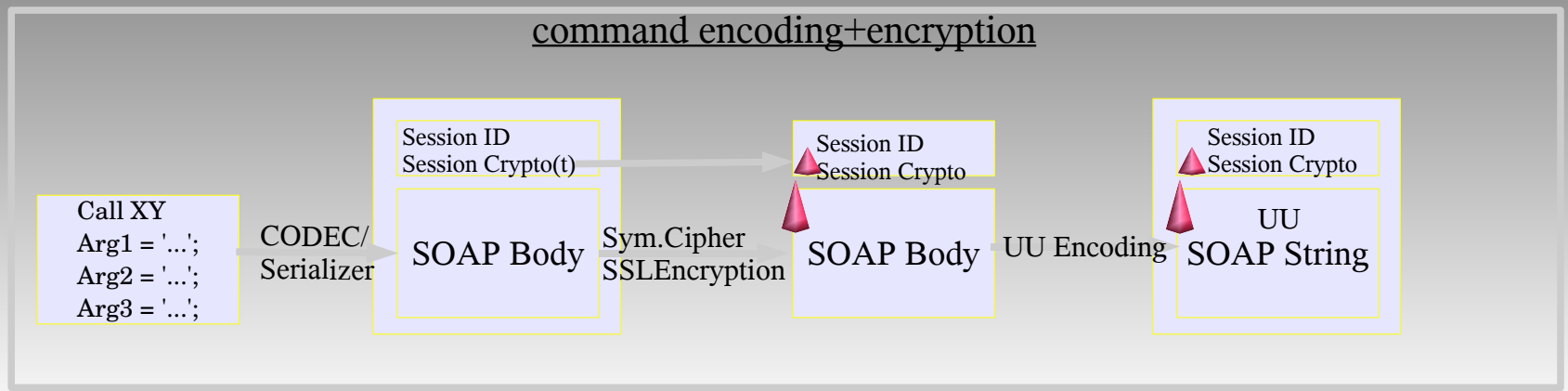
EGEE is a project funded by the European Union under contract IST-2003-508833

The ROOT– AliEn Integration: TGrid + TFile AliEn Plugins



AliEn API Service - UI Authentication and Message Encoding





TGrid API - Connect

- Session Setup
  - TGrid::Connect("alien://"); // sets gGrid global variable
    - Establish a session with the API service using a memory token
    - not thread-safe – don't use with threads
  - TGrid::Connect("alien://",0,0,"t");
    - Establish a session using a token stored in the file system
    - thread-safe – token is locked every time it is used
    - token is shared with the session in the
  - Authentication:
    - AFS Password
    - Grid Proxy – Let your Certificate be registered in the AliEn LDAP server
      - the API distributed with AliEn comes automatically with GLOBUS support
      - recompile your ROOT with GLOBUS support, the GLOBUS libraries can be found in /opt/alien/globus/lib
  - successful Authentication:
    - you see MOTD and gGrid != 0
  - If you run in AliEn, TGrid::Connect(.) authenticates with the Jobtoken!

TGrid::Command
Low-level command execution

- Basis for all TGrid functionality
- Every AliEn shell command can be executed via

  TGridResult *TAlien::Command(const char *command, bool interactive, UInt_t stream)
  - <command> is any AliEn shell command
  - <interactive>=true prints the stdout of the command
  - <stream> selects the API stream which is stored into the TGridResult
    - can be
      - kSTDOUT : store the stdout
      - kSTDERR : store the stderr
      - kOUTPUT : store the array of hash values returned by the command
      - kENVIR : store the environment variables returned by the command

- This is a low-level method, which is not needed to be used in the general use cases. In case you miss a high-level abstract method in the TGrid interface, you can use this low-level function: Experts Only!

TGrid API – Catalogue Interface

- const char* gGrid->Pwd(Bool_t verbose)
  - returns the current working directory
  - if verbose=true, the path is printed on stdout
- Bool_t gGrid->Cd(const char* ldn, Bool_t verbose)
  - changes the working directory to the logical directory name <ldn>
  - returns true, if the working directory could be changed
  - if verbose=true, errors are printed out on stdout
- TGridResult* result = gGrid->Ls(const char* ldn, const char *options)
  - lists the logical directory name <ldn>
  - if successful, returns a TGridResult, otherwise 0!
  - don't forget to delete the TGridResult afterwards!
  - options can be “” or “-la” or “-b” to retrieve more file information

TGrid API – Catalogue Interface TGridResult and gGrid->Ls()

- TGridResult contains abstract methods to retrieve File Listing results:
  - const char* TGridResult::GetFileName(TGridResult::GetKeyint index)
    - returns only the file name without directory path

```
Int_t i=0;
while (result->GetFileName(i))
      printf("File %s\n",result->GetFileName(i++));
```

  - const char* TGridResult::GetFileNamePath(int index)
    - returns the complete file path (directory+ filename)
  - const char* TGridResult::GetFilePath(int index)
    - returns only the directory where the file is found
  - In conjunction with gGrid->Ls(<dir>,"-la") you can use a generic function const char* TGridResult::GetKey(int index,const char* key)
    where key is "group","permission","date","name","user","path","size"
  - in conjunction with gGrid->Ls(<dir>,"-b") you can use TGridResult::GetKey key is "guid" to retrieve the GUID of a file

    ⇒ We should provide more abstract functions like GetGroup(<lfn>), GetDate(<lfn>), GetUser(<lfn>), GetSiz(<lfn>) + GetGUID(<lfn>) in the next release

TGrid API – Catalogue Interface Mkdir, Rmdir, Register, Rm

- Other catalogue functions:
  - Bool_t Mkdir(const char *ldn="", Option_t *options="", Bool_t verbose=kFALSE);
    - create directory
    - use “-p” to create recursive directories
  - Bool_t Rmdir(const char *ldn="", Option_t *options="", Bool_t verbose=kFALSE);
    - delete directory
    - use “-r” to delete recursive directories
  - Bool_t Register(const char *lfn, const char *turl, Long_t size=-1,const char *se=0, const char *guid=0, Bool_t verbose=kFALSE);
    - create a catalogue entry – no file copy
  - Bool_t Rm(const char *lfn, Option_t *option="", Bool_t verbose=kFALSE);
    - delete a catalogue entry – does not work for directories

TGrid API – Catalogue Interface Query

- To run a “find” like command use
  - TGridResult *TAlien::Query(const char *path, const char *pattern, const char *conditions, const char *options)
    - <path> is the dir where to start the search, must have the “alien://” prefix
    - <pattern> is the match pattern f.e. “galice.root” or “*.root”
    - <conditions> are meta data queries
      - meta data situation in AliEn not clear to me
      - options supported are f.t. moment:
        - “-l <maxentries>” - limit the query to return <maxentries> from each queried DB
  - Example:

```
 TGridResult* result =
gGrid>Query("/alice/cern.ch/user/p/peters/analysis/miniesd/",
"*.root","","");
result->Print();
```

TGrid API – Catalogue Interface Query Result Converions

- A TGridResult as returned by the Query functions contains the following Keys (use TGridResult::GetKey):
  - “type”,”dir”,”lfn”,”perm”,”owner”,”turl”,”ctime”,”md5”,”seStringlist”,”aclId”,”guid”,”expiretime”,”size”,”replicated”,”entryId”,”selist”
  - in normal use cases, users don't need to deal with this low-level representation and to use the GetKey method!
- Convert a TGridResult into the new List of TFileInfo Objects:
  - TList* fileinfolist = result->GetFileInfoList()
    - file info list objects store lists of turls, md5, size and entry information for ROOT trees. Users can attach any additional meta data to this object
- Convert a List of TFileInfo objects into a TChain:
  - Int_t TChain::AddFileInfoList(TFileInfoList* list, int maxentries)
    - Adds all the elements or <maxentries> to a Chain

TGrid API – TFile (TAlienFile) Interface

- Open an ROOT File in AliEn via:
  TFile::Open("alien://<lfn>");
- Open a non-ROOT File
  TFile::Open("alien://<lfn>?filetype=raw");
  - you can use the SysSeek,SysRead,SysWrite functions to implement raw file access in ROOT
- Open a ROOT File on a specific storage element
  TFile::open("alien://<lfn>?se=<sename>");
- Warning:
  - Not all methods, which are available in TFile are automatically working in TAlienFile. If you have problems, check TAlienFile.h if your function is listed there!
  - If you delete a TAlienFile, gFile does not move automatically to the file opened before the TAlienFile!
  - There is some implementation problem here to solve with the Plugin-Mechanism in TFile ....

TGrid API – TFile (TAlienFile) Interface

- TFile Options:
  - "" or "READ" - read support fully supported
  - "CREATE" or "New" - write support fully supported, requires, that the entry does not exist in the catalogue
  - "RECREATE" - creates a new version of a file, if it was already existing, if you don't want this, you have to delete the file beforehand!
  - "UPDATE" - supported on the client, but not yet implemented in the xrootd server
    - the file has a new version/guid in the catalogue
    - the xrootd server has to make a copy of the existing file before you can modify it (not implemented yet!)

- If you write files without specifying an SE, it will be the default one of the API service, where you are connected
  - better specify your default one with "export alien_CLOSE_SE=<X>"

TGrid API – TAlienCollection

- class to represent list of files or list of list of files, i.e. to define the input data for analysis (locally or as a batch job)
- not yet virtual in TGrid because very specific to AliEn
- TAlienCollections are build with Query commands like used in TGrid or via XML files (static functions Query or Open)
  - in batch jobs, AliEn can produce from the InputData section in a JDL an XML file, which an anaylsis batch job can read into a TAlienCollection object
  - we should implement the Query command also for local files ....
  - the batch job then has to loop over the collection files:
    - TAlienCollection::Reset()
    - TAlienCollection::Next();
    - TAlienCollection::GetTURL("Kinematics.root");
- the integration into the AliEn batch analysis is tested at present ....

AliEn File Copying from ROOT TFileMerger Class

- To copy local files to AliEn and vice-versa, use the TFileMerger Class from libProof.so
  - gSystem->Load("libProof.so");
  - TFileMerge merger;
  - merger.Cp(<url1>,<url2>);
  - <url1,2> can be any local or remote URL:
    - "file:/..."
    - "root:/..."
    - "alien://..."
  - copy local file /tmp/test.root to alien:
    - merger.Cp("file:/tmp/test.root","alien:///alice/cern.ch/user/t/test/test.root");
  - copy alien file to local:
    - merger.Cp("alien:///alice/cern.ch/user/t/test/test.root","file:/tmp/test.root");

TGrid API – Job Submission Interace

- There is also a complete API for the WMS of AliEn:
  - TGrid::Submit - submit a job to AliEn
  - TAlienJDL - class to produce a JDL
  - TAlienJob - class describing an AliEn Job
  - TAlienMasterJob - class describing an AliEn MasterJob
- in many cases, there is no need to use this methods as a user directly.
- the WMS part in ROOT is not yet extensivly tested, therefore I postpone a detailed description ....

TGrid API – Documentation and Questions

- Documentation is integrated into ROOT
- Don't hesitate to send questions to me:
  - Andreas.Joachim.Peters@cern.ch
- If you need a more abstract functions, send me a request to integrate it into the ROOT interface like f.e.

Good Luck with the API!