Table of Contents

The Future of DIRAC....1
Interface....1
File Catalog....2
Production system....2
Application interface....2
Data management....2
Production system for user....3

The Future of DIRAC

This last year of usage made several deficiencies and problems appear in different part: the interface, the file catalog, and the production system.

Interface

The interface appears to be quite cumbersome: the design makes it hard to maintain, and it's not easy for users either. The plan is to decouple the applications from the jobs, such that users have more control on what is done. Production managers lives should also be improved by this new interface.

As a recal, here is how to define a Mokka job now:

```
from ILCDIRAC.Interfaces.API.ILCJob import ILCJob
from ILCIDRAC.Interfaces.API.DiracILC import DiracILC

dirac = DiracILC(True,"repo.rep")

job = ILCJob()
job.setMokka("v0706P08","steering.steer",'input.slcio',nbevts=10,OutputFile='myfile.slcio')
job.setName("MyName")
job.setJobGroup("MyGroup")
job.setSystemConfig("x86_64-slc5-gcc43-opt")
job.setCPUTime(10000)

dirac.submit(job)
```

The definition of the setMokka is the real problem: the order of the arguments is important, it's hard to kknow which argument is what, and for the maintenance it's a pain as the code has lots of copy-paste in (and that's BAD!). So the idea is to separate the applications from the job, and have classes instead of a method:

```
from ILCDIRAC.Interfaces.API.UserJob import UserJob
from ILCIDRAC.Interfaces.API.DiracILC import DiracILC
from ILCIDRAC.Interfaces.API.Applications import Mokka

dirac = DiracILC(True,"repo.rep")

job = UserJob()
job.setName("MyName")
job.setJobGroup("MyGroup")
job.setSystemConfig("x86_64-slc5-gcc43-opt")
job.setCPUTime(10000)

mokka = Mokka()
mokka.setVersion("v0706P08")
mokka.setSteeringFile("steering.steer")
mokka.setInputFile("input.slcio")
mokka.setOuputFile("myfile.slcio")
mokka.setNbEvts(10)

job.append(mokka)

dirac.submit(job)
```

The definition becomes slightly longer, but has some advantages: more flexibility, more stability. As I know some people are not going to be happy to have to type few more lines, we will provide:

```
mokka = Mokka({"version":"v0706P08","SteeringFile":"steering.steer","InputFile":"input.slcio","Ou
```

This has the advante that the order of the items in the dictionary ios unimportant, but adds the constrain that the keys have to be the correct ones. We will make sure the keys are right (return an error otherwise).

The other advantage of the individual classes is that the

```
help(Mokka)
```

will return appropriate documentation.

Chaining of applications will be also more intuitive, as we will have a method:

```
mokka.getInputFromApp(whizard)
```

This has been put in production, and the old interface is now unusable.

File Catalog

The File catalog proved to be very useful, but some things are missing.

A feature we need to work on is a validity (or quality) flag to prevent users from using data that is bad (typically after a bug was found). We also plan on putting in place some sort of a validation procedure based on TOMATO. The idea is that each "working group" will assign the validation role to someone to check the histograms, and eventually mark the data as OK.

Production system

Producing the CDR data with the current production sytem proved to be quite easy, but could be even better.

- Whizard's processes are hard to deal with: they require the processlist.cfg to know which version of whizard contains it, and all whizard jobs need that file. Also it's hard to know which processes are available (for the moment the content of processlist.cfg is dumped on screen).
- Some software versions become outdated (when bugs are found) so should be blocked from usage: users should be prevented from using a buggy version. Plus now that we use the Shared Area, we need to install and remove the applications by hand (running special jobs). A dedicated agent/service could take care of that.
- preparing production summaries is hard: it needs to be done "by hand" more or less. The idea is that the new system would hold the relevant info and an agent would retrieve the info automatically every day and send a mail (or even if possible upload it to a web service) continaing the tables.
- Steering files are for now installed on the sites like any other application. In principle the new system could hold those steering files' templates.

Application interface

All the applications share a set of parameters, and the run time environment only depends on a few env variables. Why not having a script that produces a skeleton, both for the Interfaces part and the Workflow module part. This would make sure that the variable names used are identical, and would avoid copy-pasting from old modules. Moreover, handling of the InputData is always the same, so this would be simpler. Time to accomplish: 2-3 weeks full time

Data management

Data popularity? Decrease time of access, make sure the files are staged, etc.

How would that work? add a service behing the ReplicaManager: stores the number of times (and from where) a file is called. Use this as an estimation of usefulness of a file, and storage location. Estimate of longevity is also needed.

Production system for user

Use the same idea as for transformations, but for users' jobs. Data driven procedure of course.

Idea for control: submit N jobs, evaluate outcome, if success (N_done/M_failed >> 1) continue submitting. Evaluate continuous jobs status: if P continuous jobs fail, stop.

In case of too many failures, stop and notify user.

-- StephanePoss - 02-Aug-2011

This topic: CLIC > FutureOfDirac
Topic revision: r4 - 2012-10-05 - StephanePoss