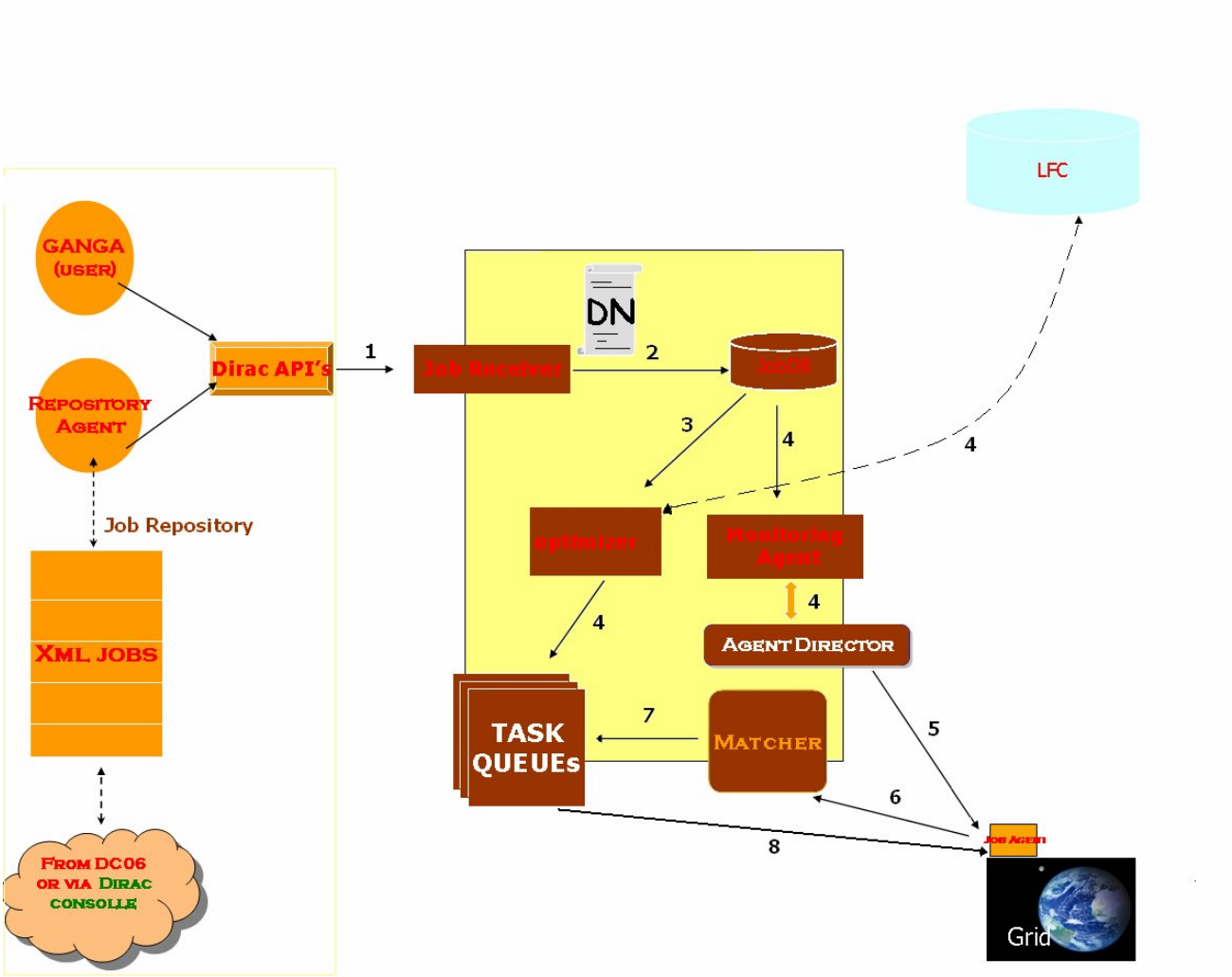
LHCb Pilot Agent Model

With this write-up we aim to describe the pilot job model that is currently implemented by the computing infrastructure of LHCb experiment DIRAC (Distributed Infrastructure with Remote Agent Control). The pilot model embraces the grid pull paradigm and represents an extremely powerful and flexible system for running jobs independently on the back end (LCG, Dirac, LRMS, standalone machine). By reminding to other reading for further information, this is rather a technical note that should help to understand how really this mechanism works (mechanism too often not fully understood).
In figure is schematized the whole mechanism.



1. A Dirac user submits a job. By Dirac User we mean either a "human" user (GANGA, Dirac CLI) or a special agent called ***Repository Agent***. In the former case a user (either through GANGA or through Command Lines) submits its own Dirac JDL with options and list of input files collected via the ***Bookkeeping*** service […]; in the latter the Repository Agent allows to submit to the Dirac

WMS a well defined number of (Dirac) jobs that it takes from a area called ***Job Repository*.** In turn, jobs are inserted into the Job Repository either via a special Graphical Interface (*Dirac Console)* by the on-shift production managers or via other agents and mechanisms (***Transformation Agents**,* to be contextualized in a more general view -see DC06 Infrastructure document)

2. The Dirac job is shipped (together with the Grid User Proxy) via SSL layer, to the ***Job Receiver*** that authenticates and validates the user credentials and puts Proxy and JDL in the ***JobDB*** by appending in a generated ID (form now on: **Dirac ID**)
3. Once the job has been stored in the DB the ***Optimizer*** polls the DB and performs several checks (against the File Catalog and/or other modules for prioritization) and puts the job in the corresponding ***TaskQueue*** (depending on the user DN, on the site and on the CPU length) with also a priority set.
4. As soon as the job has been stored in the JobDB another agent polls the data base: the ***Monitoring Agent***. This agent checks for the status of the job and (in case it has not yet been picked up) lets the *Agent Director* to submit an empty job (Pilot) to the backend infrastructure.
5. The ***Agent Director*** submits the Pilot Agent using the User Proxy. Dirac stores just one single proxy per user (the latest one for expiration time). It builds the right JDL and registers the EDG-ID of the pilot under the corresponding Dirac-ID job entry. Once the Pilot arrives at the resource, it installs the Dirac environment and performs few other checks on the hardware available (memory, disk, cpu). After that, it starts another agent: the ***job agent***. The job agent is a "volatile" agent (its life is nearly equal to the CPUTime set on the local batch system) that in LCG just replaces the permanent agent running on Dirac; it represents the decentralized intelligence of the Dirac system. The monitor Agent will keep querying the JobDB until the status of the Dirac-ID informs it has been picked up. After a (customizable) time, if the job is not yet running it asks the Agent Director to submit another Pilot whose EDG-ID is also registered and appended in the JobDB entry. This multiple submission (that reminds "shallow resubmission" mechanism in gLite) continues until the Dirac-ID job starts running.
6. 7. 8. The job agent on the WN asks the ***Matcher*** for a task to be executed by presenting some information like CPU, Memory, Disk available, site, user's DN. The Matcher, on top of this information, checks the availability of tasks in the corresponding task queue; it informs then the job agent about the ID of the task to be executed and the XML description of the job is then sent to the WN which starts the execution. The monitor agent updates the JobDB accordingly. It is worth to highlight the following:
    - **a.** Pilot jobs that have failed (either at the LCG level or at Dirac level) to run the *job agent* on the site (e.g. Installation failure, communication failure) will imply a resubmission of another Pilot for the same task. **Statistically it has been evaluated a ratio of 1.4 LCG (pilots) job per 1 Dirac job.**
    - **b.** The Pilot submitted for a given task may pick up another (previous) job sitting in the same task queue where the original Dirac job was also queued and then satisfying the same requirements. This is simply a way for optimizing the grid usage and for allowing centrally managed prioritization strategies. **Back in the days, when the Dirac-ID was also**

presented by the Pilot so that it was univocally taken, that wasn't possible.

c. **<u>It is not TRUE</u> that for each Dirac tasks, the Dirac WMS submits a priori tens of Pilot (**that die then gracefully once the job is honored**)** and then infests the LCG resources**.** Each pilot - getting a slot - will use the resource if there are tasks available. Full exploiting of the resources improves further by running more than one task per Pilot. This is also under investigation by knowing the remaining CPU time on the WN.

d. **<u>It is not TRUE</u>** that pilots running on behalf of a certain DN are allowed (by the Dirac WMS) to run tasks belonging to someone else. **The User DN of the Pilot is always checked against the Dirac owner of the task.** Only when ***glexec*** will be available on the WN this control will be by-passed.

e. In the very next future Job Throttling mechanism is also in the scope of their investigation.