Journal of Physics: Conference Series **396** (2012) 032110 doi:10.1088/1742-6596/396/3/032110

Grid administration: towards an autonomic approach

M Ubeda Garcia[1], F Stagni[1], A Tsaregorodtsev[2], P Charpentier[1], V Bernardoff[1]

[1]PH Department, CH-1211 Geneva 23 Switzerland
[2]C.P.P.M - 163, avenue de Luminy - Case 902 - 13288 Marseille cedex 09

E-mail: mario.ubeda.garcia@cern.ch

On behalf of LHCb Collaboration

Abstract.

Within the DIRAC framework in the LHCb collaboration, we deployed an autonomous policy system acting as a central status information point for grid elements.

Experts working as grid administrators have a broad and very deep knowledge about the underlying system which makes them very precious. We have attempted to formalize this knowledge in an autonomous system able to aggregate information, draw conclusions, validate them, and take actions accordingly.

The DIRAC Resource Status System (RSS) is a monitoring and generic policy system that enforces managerial and operational actions automatically. As an example, the status of a grid entity can be evaluated using a number of policies, each making assessments relative to specific monitoring information. Individual results of these policies can be combined to evaluate and propose a global status for the resource. This evaluation goes through a validation step driven by a state machine and an external validation system. Once validated, actions can be triggered accordingly.

External monitoring and testing systems such as Nagios or Hammercloud are used by policies for site commission and certification. This shows the flexibility of our system, and of what an autonomous policy system can achieve.

1. Introduction

Information availability, reliability and usability are three crucial aspects for an efficient management of the resources. In particular, a Grid infrastructure presents a wide variety of information endpoints, which turns to be spread around the monitoring tools provided by different projects. Moreover, each one of them provides different information perspectives about infrastructure and/or activities. Information given by orthogonal entities can overlap, be out of sync and/or be unrelated.

On one hand, the need of an aggregator of monitoring information is a known requirement [1]. On the other hand, the interpretation of the information will determine our knowledge of the environment.

This paper presents the evolution of the DIRAC RSS [1] driven by its usage within the LHCb collaboration as follows: section 2 gives a brief explanation of the context where the RSS was born. Section 3 formalizes the list of requirements for this project. Section 4 explains the

underlying components within the RSS, which constitutes its heart while section 5 gives an overview of the system and the status cycle. Finally, conclusions are given in section 6.

2. Context

DIRAC is a community Grid solution [2]. DIRAC, developed in python, offers powerful job submission functionalities, and a developer-friendly way to create services and agents. A DIRAC service exposes an extended Remote Process Call (RPC) and Data Transfer (DT) implementation, whereas an agent is a stateless light-weight component (comparable to a cron-job). DIRAC has been initially developed as a LHCb-specific project, and it is now a generic framework, capable to serve the distributed computing needs of a number of Virtual Organizations.

3. Requirements

The system requirements are hereafter summarized. The system must:

- collect dispersed and uncorrelated monitoring information from all the available sources;
- aggregate and expose the information in a simple way;
- use a policy system to combine information and deliver a decision. Non-trivial combinations must be allowed, to take into account the variety of resource natures and the complexity of the system;
- automate trivial and nontrivial tasks, by enforcing defined procedures;
- cope with scalability derived issues;
- be configurable enough to fit the needs of any other VOs using the DIRAC framework.

The purpose of the Resource Status System is to provide a system meeting these requirements. The architecture and the functionalities exposed within this paper have been implemented within the LHCb [3] collaboration inside the DIRAC framework.

3.1. Resources status central point of information

The RSS status database is the central point of status information for DIRAC resources. Half of the RSS system is dedicated to host statuses and to provide all necessary helpers for managing them as efficiently as possible. This includes a GUI integrated with the DIRAC Web portal and scripts that can be used by operators of the system.

3.2. Information aggregation

The proliferation of monitoring tools gave the VO administrators the opportunity to access a widespread set of information endpoints, which would be difficult to handle and keep track of individually. Thus, multi purpose monitoring platforms such as GridView [4], MyOSG [5], HappyFace [6] or Dashboard [7], among others, offer a wide set of possibilities in terms of access to monitoring information. The importance of information aggregation is crucial, and the mentioned projects are a proof of that. These tools represent very powerful instruments for local resource administrators, offering views of multi-VOs activities running at a given site. However, VO administrators require a more VO-centric view of the resources used, which RSS wants to provide.

3.3. Information interpretation
Frameworks previously cited in 3.1 are using customizable algorithms to compute the status they show. These algorithms are hosted on applications like Site Status Board (used by Dashboard) or the Availability Computing Engine (used by both Nagios and GridView). A VO-centric view would certainly require to overcome the natural limitations imposed by a general purpose tool. More specifically on the algorithms used to compute the records, as they are not being customizable enough to meet the needs for complex cases within a VO.

3.4. Automation
Once the Grid operations within a team are established with clear procedures in place, running some of the operations of the system becomes a simple routine job for an experienced Grid administrator. Taking a decision or applying a procedure should as much as possible be an automatic operation. From now on we will refer to such operations as formal policies. There is a lack of mechanism for automatically enforcing such policies. Hence a Grid administrator is required to be available in order to react manually when changes happen in the system.

4. Underlying components
Any DIRAC System makes use of databases, services, clients and agents. These are the pillars of the DIRAC architecture. In order to achieve its goal, the RSS, as a DIRAC system, makes use of a few concepts which are the heart of this system:

- *Grid ontology*;
- *State machine*;
- *Policies*;
- *Policy system*;
- *Token locker*.

In our ontology we define a set of classes that accurately model the Grid for RSS purposes. These classes are a loose reflection of the DIRAC classification of resources. A common attribute for every instance of a class is the status, which may not be unique, meaning that every instance can have more than one status assigned. Transitions between the different values for the status type of every resource are forced by a state machine. Policies are in charge of triggering the status changes of the individuals. From now, we will refer to class instances as individuals.

In the figure 1 we show an abstraction of the RSS monitoring system: an ontology with three class instances; the ExtendedPolicySystem which encompasses the PolicySystem, the State machine and the Token Locker; finally, the policies connect the PolicySystem with the monitoring information.

4.1. Grid ontology
The RSS provides by default a generic Grid Ontology which can be customized by the communities to accommodate their needs. The default ontology, showed in figure 2, proposes a schema with three main classes: Site, Resource and Node with aggregation links in between.

Each class can have a type. In the case of the Resource class we have the following default types: Computing, Storage, Catalog, FileTransfer, Database and CommunityManagement. In figure 3, we can see an example of how the Grid Ontology used in the RSS maps the internal structure of the service used in DIRAC to describe the IT resources - from now, ConfigurationService (CS). In this case there is an element of Site class (CERN.ch), which three children nodes of class Resource with types Storage, Computing and FileTransfer respectively.

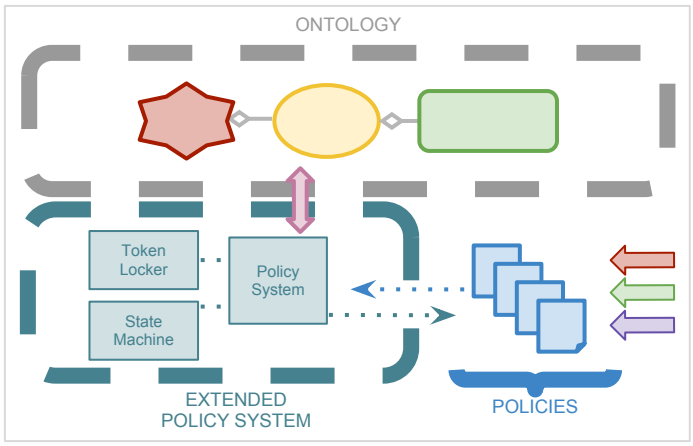

Figure 1. The RSS system at a glance: ontology, policy system, state machine, token locker and policies.

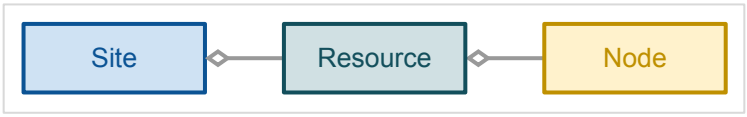

Figure 2. Ontology class diagram

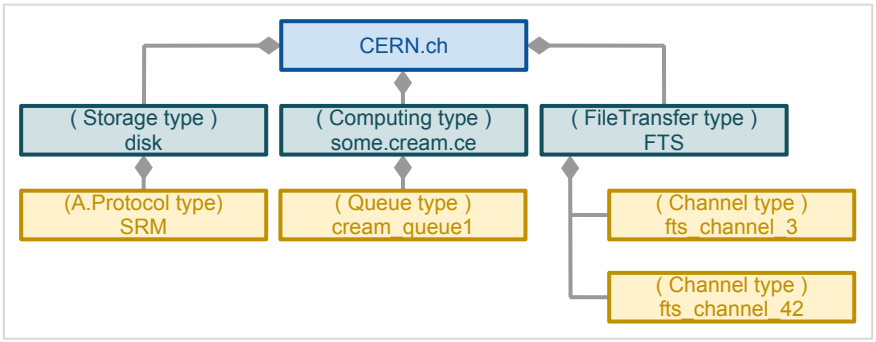

Figure 3. Ontology example

4.2. State machine

The state machine has four default states (ordered by severity): Active, Bad, Probing and Banned, but it can be extended to accommodate additional states and transitions according to the VO needs. The allowed state transitions are depicted in the figure 4.

Each of the RSS states is a result of the computation of the policies evaluated by the RSS. The authors believe that 4 states are enough for representing all the possible states of Grid resources. These states are mapped to DIRAC internal states, that determine if a resource can be used or not: Allowed or Banned

Active and Banned are self explanatory. The first one implies that the individual is in good shape and no problem/quality degradation has been reported. The second one implies the

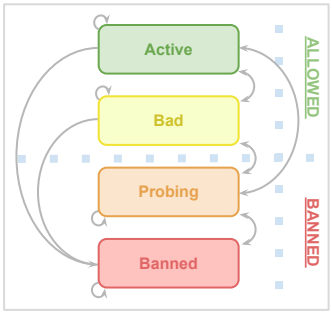

Figure 4. State machine

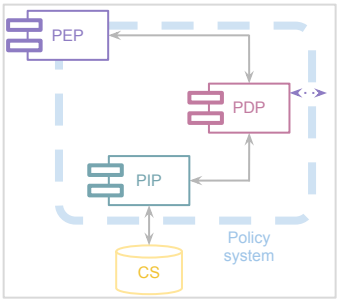

Figure 5. Policy system

individual is basically, out of order. Bad and Probing are slightly more complex states. Bad is a state that an individual gets when problems are observed but not so important to rush directly to ban. The individual in DIRAC is still Allowed, which means it does not work at 100%, but we can still get a reasonable performance out of it. Finally Probing is a state where we know there were problems with the resource serious enough to have it banned. The original problems went away but we want to test one more time the individual before unbanning it for real. The individual in DIRAC is still Banned.

Last remark, transitions to Probing are allowed from any other state. However, in the LHCbDIRAC use case, it only makes sense from the Banned state. By design, the State Machine allows such transitions, but in real life we will never reach that situation unless the policies would have been badly implemented or configured.

4.3. Policies

Without any doubt, policies are the most important part of RSS. There are many components that make RSS work, but the knowledge, the interpretation of the monitoring information spread around the different third party systems is done here. Understand them as a mapping between information from monitoring sources and the RSS states. A good setup of every single policy is crucial for RSS.

A policy can be divided into two parts, metadata and the policy itself. The policy is a set of rules which given an input, return a status.

Typically, the input is the output of a command, where command is the name given in RSS for a client that connects to a particular monitoring system and returns whatever information is stored there. Moreover, a policy does not necessarily work with only a single command, they can use different commands if needed (one at a time). So, each policy, given an input, returns a status for the given conditions.

Policy configurations are used to match policies with commands based on the metadata of the resources to be monitored. Every individual has a set of attributes. In the case of a Resource, its type is one of the attributes. One can specify which policies apply to Resources of type Computing or Storage. Taking it a little bit further, one for example can apply policies to those individuals that are Computing resources with status Banned.

4.4. Policy system

The Policy system is in charge of getting all applicable policies, running them, evaluating all the statuses proposed by the policies and select the most reasonable one. Having taken a decision, the last step is to take actions accordingly. Below there we show a simplified version of the Policy system components diagram.

Note that the terminology used follows the XACML [8] specification, where a Policy Decision Point (PDP), a Policy Enforcement Point (PEP) and a Policy Information Point (PIP) are specified and used, as shown in figure 5.

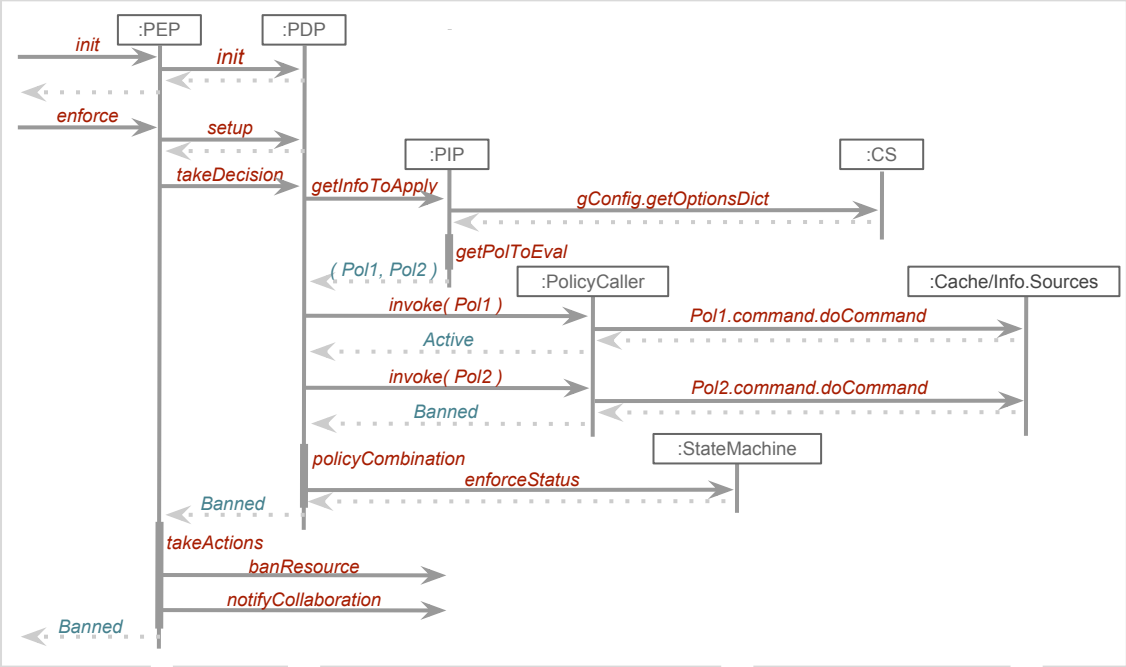

Figure 6. Policy system sequence diagram

The cycle followed by the Policy System begins and finishes on the PEP, as described in figure 6. A PEP object interrogates a PDP instance about the status of an individual. The PDP contacts the PIP, which is in charge of knowing which set of policies apply in this case and must be evaluated. Policies are evaluated, with information either from the remote source or a local cache used to reduce the network traffic.

Policies use a limited number of parameters and thresholds to evaluate a status. Together, they build the Quality of Service (QoS) requested. A formalization of such QoS, persisted in Service Level Agreement DB, is envisaged and will complement the Resource Status System.

Once all policies have been evaluated, their results go back to the PDP. When multiple policies are evaluated, the PDP combines the results of the policies, and returns a single status to the PEP.

4.5. Token locker

The token locker is a small lock owned by every individual in the grid ontology. This token locks/unlocks the access of the Policy System to the individuals. Each token has an expiration date, after which whatever value it had will be changed to the default one. The tokens are mostly used by operators when they need to perform a manual operation which was not foreseen and to keep the control of the resource out of the PolicySystem.

5. System overview

The complete RSS system makes use of a few more pieces for inspection and persistence, respectively. The first components are the agents, namely InspectorAgents and CacheAgents.

The second ones, are the databases, that store the statuses and also the cached information used by the policies. Figure 7 represents the RSS system overview.

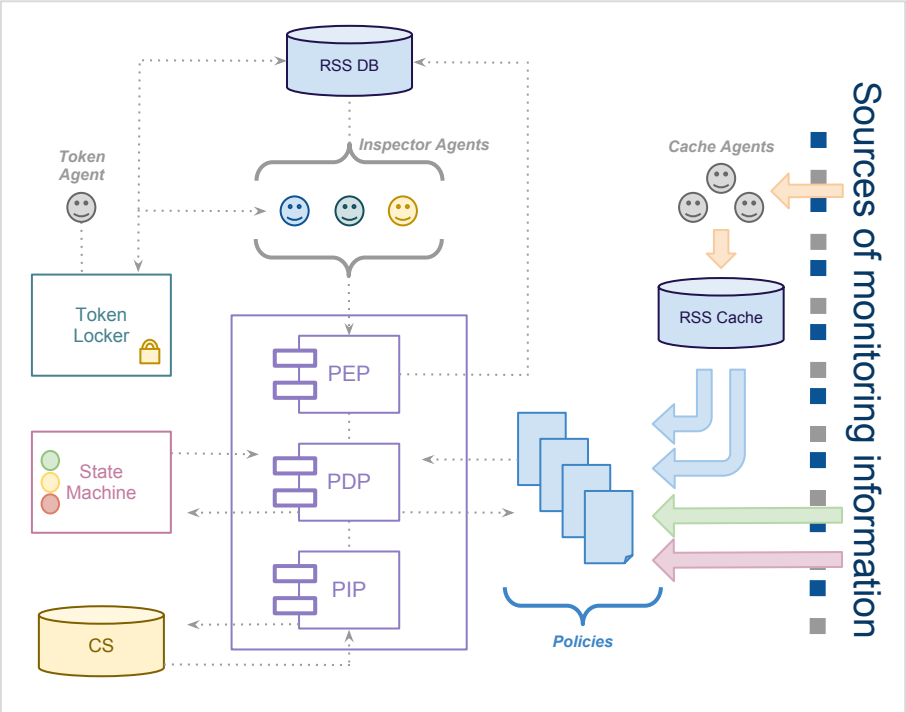

Figure 7. System overview

CacheAgents periodically query the external monitoring sources and populate the cache with up-to-date information. InspectorAgents run at a very high frequency and examine all resources registered in the system. Depending on the polling frequency and the number of resources registered in the system there can be any number of InspectorAgents running in parallel. In the LHCb case, there are three, one per resource type: Site, Resource and Node. The token agent periodically polls the Token Locker, and releases any expired user lock.

Inspector agents will read from the Resource Status Database (RSS DB in the image) which resources have not been checked lately and query the PolicySystem for a new status.

The PolicySystem will go through the PIP to find the policies to be evaluated, they then will contact the Configuration Service for the necessary information. The PDP will choose a new status if applicable and enforced to be correct according the the state machine. Back to the PEP, it will record the new status in the database along with the reasons for the status change. If needed, it will notify via email, web interface, sms, etc.. whoever may be concerned.

6. Conclusions

In this paper we presented the Resource Status System as it is used within the LHCb Computing system. It is as an information aggregation and monitoring tool which can be used by any VO using the DIRAC framework. The RSS has proven its value in a prototype developed and used by LHCbDIRAC during several months. RSS is going to be the new resource status information point for all DIRAC resources: this is major change in the DIRAC framework. RSS is also a monitoring information tool that has demonstrated the advantages of an automated policy system. The autors believe that the RSS is a step towards an autonomic resource management

tool. As it is developed as a VO independen tool, any community using the DIRAC framework can take advantage of its capabilities.

References

[1] Stagni F, Santinelli R and LHCbCollaboration 2011 *Journal of Physics: Conference Series* **331** 072067
[2] Tsaregorodtsev A, Bargiotti M, Brook N, Ramo A C, Castellani G, Charpentier P, Cioffi C, Closier J, Diaz R G, Kuznetsov G, Li Y Y, Nandakumar R, Paterson S, Santinelli R, Smith A C, Miguelez M S and Jimenez S G 2008 *Journal of Physics: Conference Series* **119** 062048 URL http://stacks.iop.org/1742-6596/119/i=6/a=062048
[3] LHCbCollaboration CERN-LHCC-2005-019 ; LHCb-TDR-11 - Geneva : CERN, 2005 - 117 p Lhcb computing : Technical design report
[4] Kalmady R, Sonvane D, Bhatt K and Chand P 2006 *Journal of Physics: Conference Series* **1** 689
[5] Gopu A, Hayashi S and Quick R Myosg: a user-centric information resource for osg infrastructure data sources.
[6] Buge V, Mauch V, Quast G, Scheurer A and Trunov A 2010 *Journal of Physics: Conference Series* **219** 062057
[7] Andreeva J, Gaidioz B, Herrala J, Maier G, Rocha R, Saiz P and Sidorova I 2009 131–139 10.1007/978-0-387-78417-5_12 URL http://dx.doi.org/10.1007/978-0-387-78417-5_12
[8] OASIS 2005 Oasis extensible access control markup language (xacml) tc http://www.oasis-open.org/committees/xacml