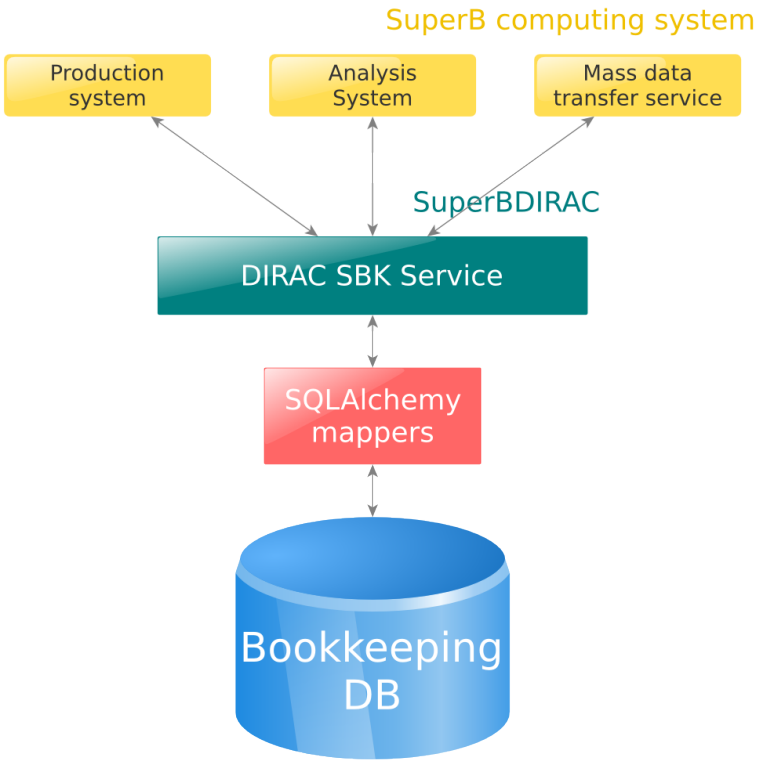
DIRAC SBKService

Overview

SBKService is a part of SuperBDIRAC extension. It is designed to handle all bookkeeping database (SBK5) operations on behalf of other parts of SuperBDIRAC (webportal, job handling, jobs, etc.).



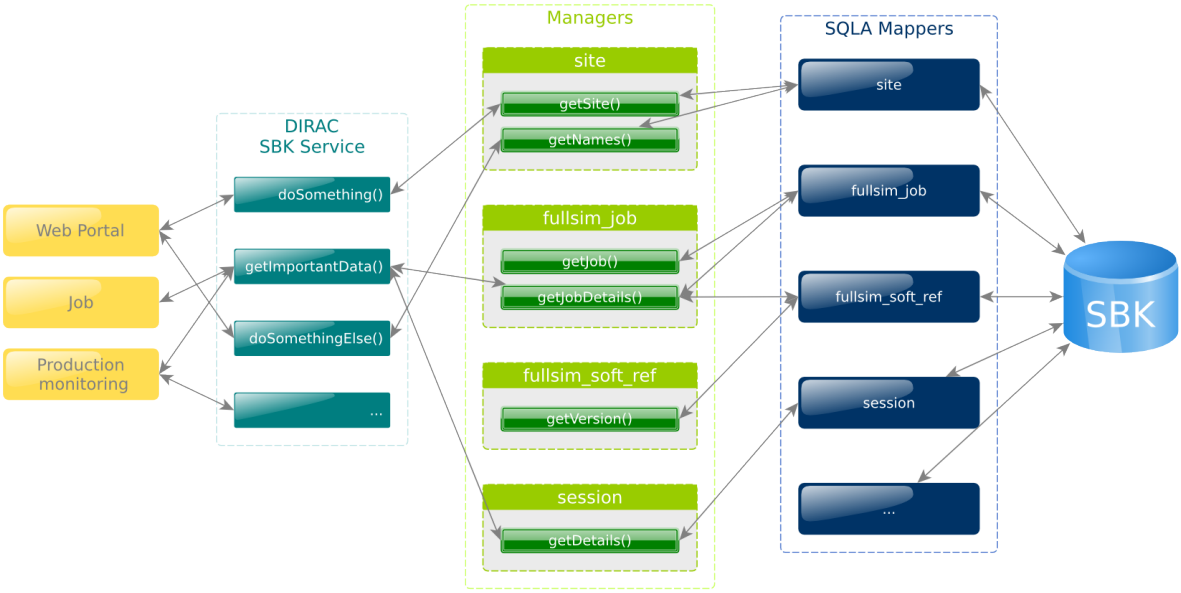
Idea

The main goal of SBKService is to gather all bookkeeping database related functions in one place, easily accessible for other parts of the system. It also gives level of abstraction: after setting of SBKService interface (functions that are exported), any changes can be made on the lower level (database engine, schema, mapping, anything) without affecting SuperBDIRAC in general.

SBKService should provide functionalities, not access to database. Set of functions should cover all scenarios of reading and writing information do database.

SuperBDIRAC doesn't have direct access to database. Database logic is on the SBKService level.

Details



SBKService is in fact semi-transparent layer on top of the SQLAlchemy based solution, which can be considered as kind of standalone - it has it's own interface containing all database related actions that are later re-exported (this time in a DIRAC way) in SBKService.

Example:
Function getSiteDetails(siteName) returns object or dictionary with site information.
Function getSiteList() returns list of objects/dictionaries with basic info of each site.

Detail level of functions should be based on usage patterns – if (just as an example) version of FastSim of a CE is massively accessed, there could be function getFastSimVersion(siteName) to avoid processing much larger siteDetails object. If not, getSiteDetails(siteName) should be enough.

Pros

DIRAC Service code is very simple and shouldn't be changed after interface is frozen.

Objects – all code is object oriented so it's very natural for Python.

Logic is on the service level – no problems with serializing connection/session objects (no RPC calls) and can be altered without requiring changes on higher levels.

Identical solution for getting access to SBK was proposed by Andrei Tsaregorodtsev during DIRAC workshop in Marseille.

Cons

Every new functionality requires implementation on all levels.

How it works on code level

SBKService.py

```
# some DIRAC imports
from sa.manager.site import Site

class SBKHandler(RequestHandler):

    types_siteDetails = [StringTypes]
    def export_siteDetails(self, sitename):
        try:
            return S_OK(Site.siteDetails(sitename))
        except:
            return S_ERROR('Error getting site details')
```

sa/manager/site.py

```
from sa.database import Session
from sa.database.entities.site import site

class Site:

    @staticmethod
    def siteDetails(siteName):
        _site = Session.query(site).filter(site.site == siteName).one()

        return [_site.site, _site.ces[0].host]
```

sa/database/entities/site.py

```
class site(object):
    pass
```

Last class is empty, it is used by SQLAlchemy to map site table, fields are added automatically, no functions or extra fields at the moment.