Transformation System report

Luisa Arrabito[1], Federico Stagni[2]

1) LUPM CNRS/IN2P3, France

2) CERN

6th DIRAC User Workshop 23rd – 25th May 2016, Montpellier

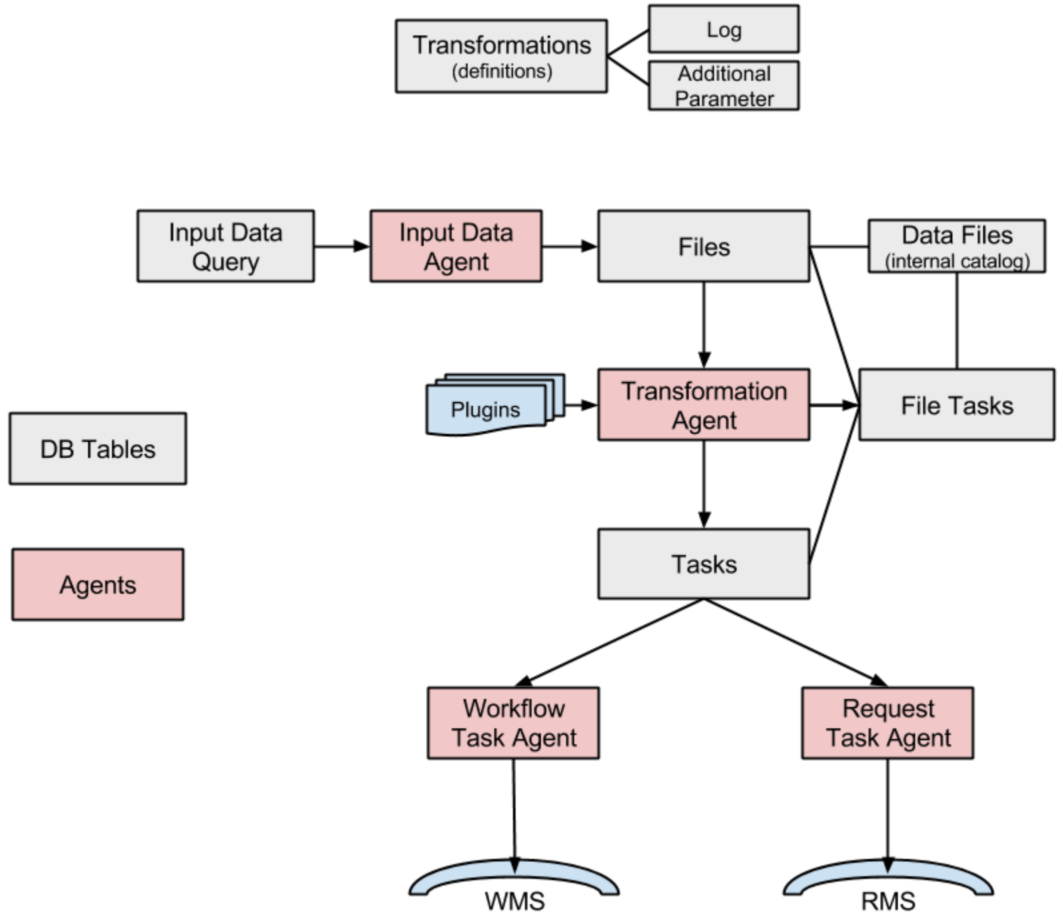
Plan

- What's the Transformation System?
- Evolutions since last year
- Future plans

What's the Transformation System?

- A DIRAC System as usually comprising:
  - MySQL DB, Services, Agents, Clients, Scripts and *Plugins*
- A system for handling "repetitive work", i.e. many identical tasks with a varying parameter
- 2 main usages:
  - Productions: the "same" job – i.e. the same workflow - is executed
    - Client for the Workload Management System
  - Data handling: replications, removal
    - Client for the Request Management System
- It handles input datasets (if present)
  - It interacts with Replica and Metadata catalogs (*e.g.* DFC or external catalogs)
  - Use of 'Plugins' to group tasks input files and set tasks destinations
- LHCb 'Production System' is built on top of it. Also CTA, ILC and Belle II use it for their productions

Transformation System architecture



- Production Manager
  defines the transformations
- TransformationAgent
  processes the transformations and creates tasks given a Transformation Plugin
- InputDataAgent
  queries the Catalog to obtain files to be 'transformed'
- WorkflowTaskAgent
  transforms tasks into job workflows, given a TaskManager Plugin
- RequestTaskAgent
  transforms tasks into requests

Plugins

Transformation Plugins

- Group input files of the tasks according to different criteria
  - Standard
    - ☐ Group files according to replica location
  - BySize
    - ☐ Group files until they reach a certain size (input size in Gb)
  - ByShare
    - ☐ Groups files given the share (specified in the CS) and location

For replication

  - Broadcast
    - ☐ Take files at the source SE and broadcast to a given number of locations

How it works in practice (I)?

- See documentation at:
  - http://diracgrid.org/files/docs/AdministratorGuide/Systems/Transformation/index.html

- Installation
  - Need to have the Transformation System installed and running. The minimum is:
    - **Service:** TransformationManagerHandler
    - **Database:** TransformationDB
    - **Agents:**
      - TransformationAgent
      - WorkflowTaskAgent
      - RequestTaskAgent
      - InputDataAgent
      - TransformationCleaningAgent

How it works in practice (II)?

- Configuration
  - Add the transformation types in the Operations/[VO]/Transformations section, *e.g.*:

```
Transformations
{
  DataProcessing = MCSimulation
  DataProcessing += Merge
  DataProcessing += Analysis
  DataProcessing += DataReprocessing
  DataManipulation = Removal
  DataManipulation += Replication
}
```

2 classes of Transformations

  - Eventually configure the WorkflowTaskAgent and the RequestTaskAgent to treat a particular transformation type

Use cases examples (I)

- MC Simulation
  - You want to generate many identical jobs with a varying parameter (and no input files)
  - The varying parameter should be built from *@{JOB_ID}*, which corresponds to the *TaskID*, and it's used in the job workflow, *e.g.*:

```
job.setExecutable( './dirac_prod3_corsika', arguments = '@{JOB_ID}' )
```

  - Create a MC transformation

```
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.Interfaces.API.Job import Job
j = myJob()
...
t = Transformation( )
t.setTransformationName("MCProd") # This must
t.setTransformationGroup("Group1")
t.setType("MCSimulation")
t.setDescription("MC prod example")
t.setLongDescription( "This is the long description of my production" ) #mandatory
t.setBody ( j.workflow.toXML() )
t.addTransformation() #transformation is created here
t.setStatus("Active")
t.setAgentType("Automatic")
```

set Type

Use cases examples (II)

- Data analysis, i.e. process a large number of files with the same program
  - You want to create many identical jobs with varying input files
  - Create a transformation with a valid type (see slide on TS configuration), *e.g.*:
    - setType(”Analysis")
  - Add files to the transformation using the TransformationClient
    - Add a list of existing files
      - addFilesToTransformation(transID,infileList)
    - Add files which are the result of a DFC query
      - createTransformationInputDataQuery(transID, {’site': ’Paranal’,'particle': 'proton',’analysis_prog=evndisp’})
      - In this way files are added as soon as they are registered in the Catalog (InputDataAgent)
      - They are most likely the result of another on-going transformation
    - Set the number of input files per job, *e.g.*:
      - setGroupSize(10)
    - Define how files should be grouped, *e.g.*:
      - setPlugin("Standard”)

Use cases examples (III)

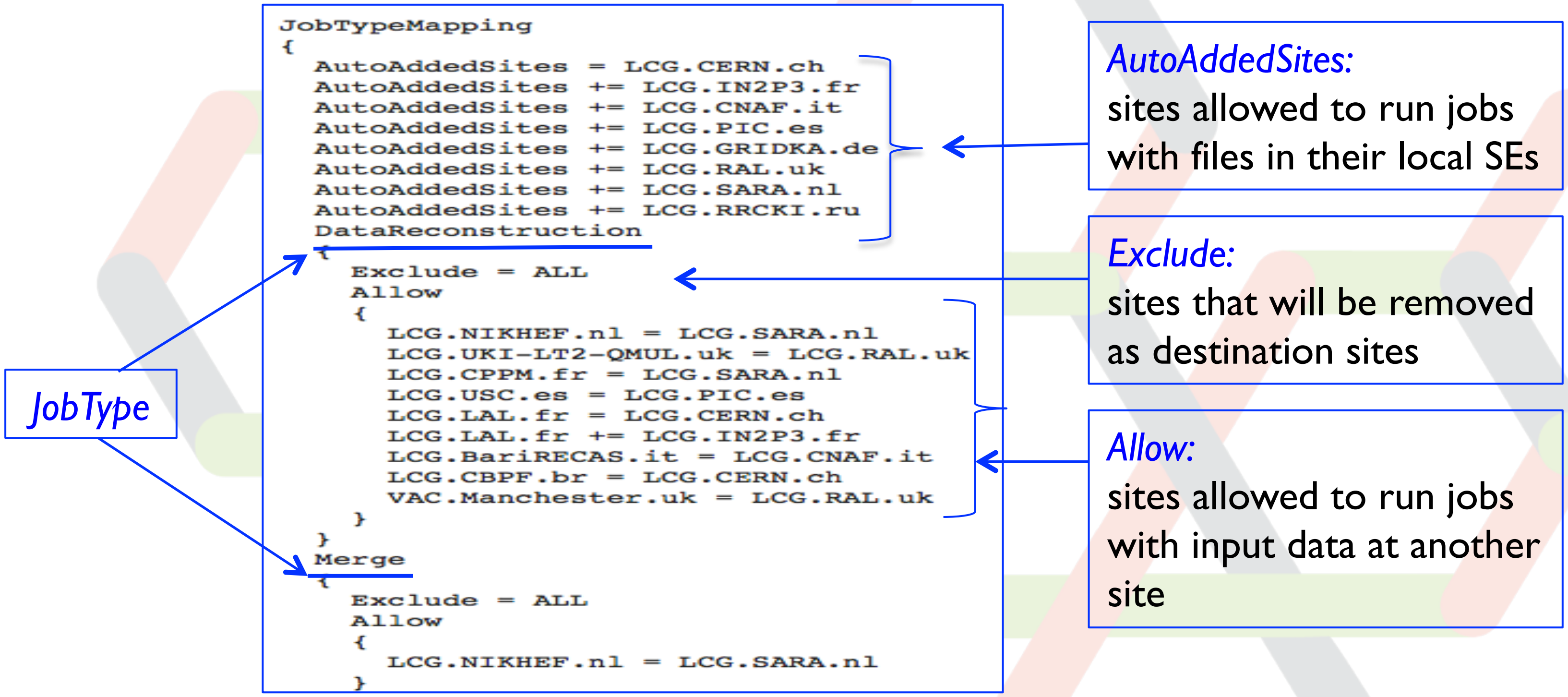
- Data Management Transformations
  - Bulk data replication, i.e. replicate many files to a list of Target SEs
    - You want to create many identical replication requests with varying input files
    - Create a Replication transformation
      - Define the type of requests to be executed
        - setBody('ReplicateAndRegister')
      - Set a valid type (see slide on TS configuration)
        - setType("Replication”)
      - Set the source and the target SEs for replication
        - setSourceSE(['CYF-STORM-Disk','DESY-ZN-Disk'])
        - setTargetSE(['CEA-Disk'])
        - setPlugin("Broadcast")
  - Bulk data removal (see details in documentation)

Evolutions since last year

- Support for parametric jobs (in v6r15)
  - Improvement of job submission
  - TaskManager prepares and submit a bunch of jobs in one go
  - It’s activated by Transformations/BulkSubmission flag in CS
- Introduction of new TaskManager Plugins (already in v6r13)
  - Used to specify tasks destination
    - BySE
      - Default plugin
      - Set jobs destination depending on the input data location
    - ByJobType
      - It allows to implement any distributed computing model by simple configuration in the CS
        - By default, all sites are allowed to run every job
        - Different rules for site destination can be specified in the CS for each JobType

JobByType Plugin: how it works?

- Configuration
  - Set Operations/Transformations/DestinationPlugin = ByJobType
  - Define the rules for each JobType in Operation/JobTypeMapping, *e.g.*:



```
JobTypeMapping
{
  AutoAddedSites = LCG.CERN.ch
  AutoAddedSites += LCG.IN2P3.fr
  AutoAddedSites += LCG.CNAF.it
  AutoAddedSites += LCG.PIC.es
  AutoAddedSites += LCG.GRIDKA.de
  AutoAddedSites += LCG.RAL.uk
  AutoAddedSites += LCG.SARA.nl
  AutoAddedSites += LCG.RRCKI.ru
  DataReconstruction
  {
    Exclude = ALL
    Allow
    {
      LCG.NIKHEF.nl = LCG.SARA.nl
      LCG.UKI-LT2-QMUL.uk = LCG.RAL.uk
      LCG.CPPM.fr = LCG.SARA.nl
      LCG.USC.es = LCG.PIC.es
      LCG.LAL.fr = LCG.CERN.ch
      LCG.LAL.fr += LCG.IN2P3.fr
      LCG.BariRECAS.it = LCG.CNAF.it
      LCG.CBPF.br = LCG.CERN.ch
      VAC.Manchester.uk = LCG.RAL.uk
    }
  }
  Merge
  {
    Exclude = ALL
    Allow
    {
      LCG.NIKHEF.nl = LCG.SARA.nl
    }
```

AutoAddedSites:
sites allowed to run jobs with files in their local SEs

Exclude:
sites that will be removed as destination sites

Allow:
sites allowed to run jobs with input data at another site

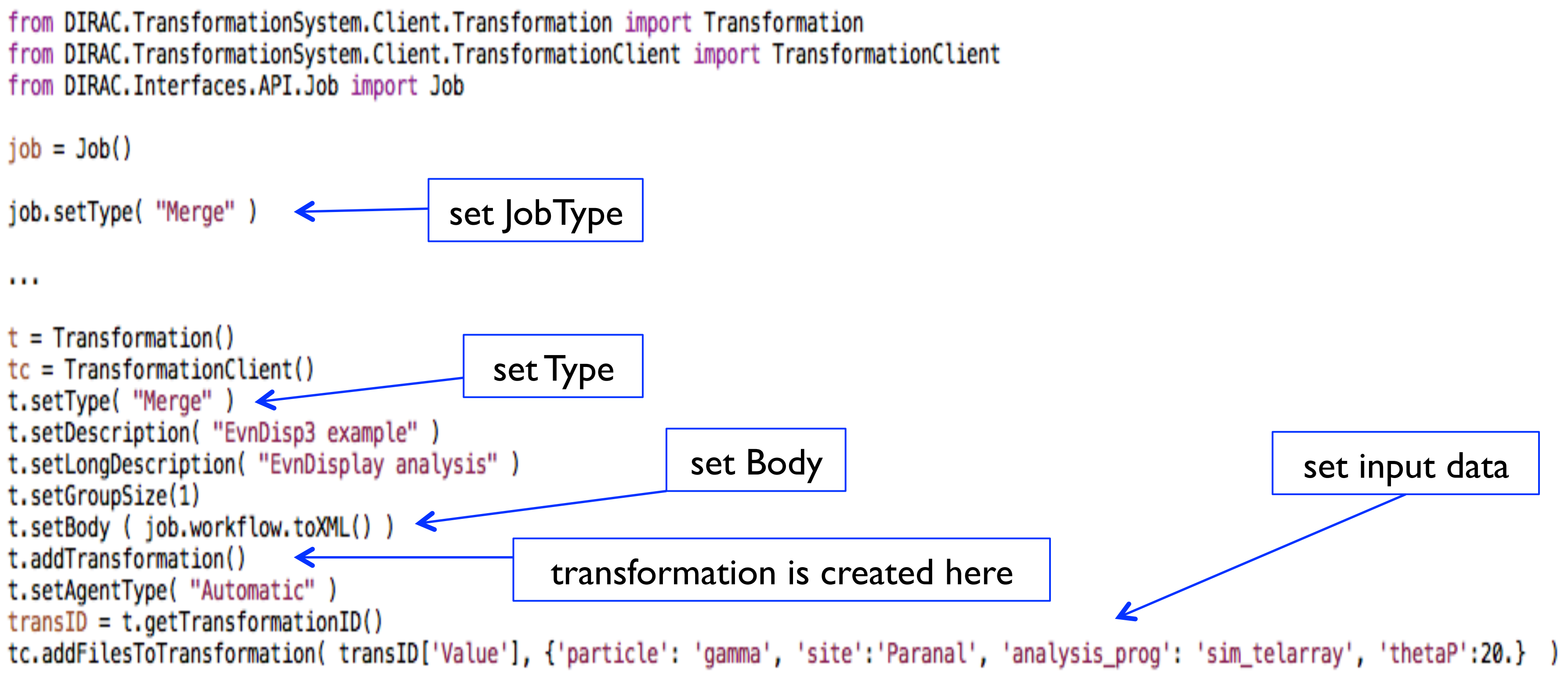
JobType

- Here 'Merge' jobs having input data at LCG.SARA.nl can run both at LCG.SARA.nl and LCG.NIKHEF.nl

DIRAC THE INTERWARE

JobByType Plugin: how it works?

- Create your transformation
  - Set JobType in the job workflow, *e.g.*:



```
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.TransformationSystem.Client.TransformationClient import TransformationClient
from DIRAC.Interfaces.API.Job import Job

job = Job()

job.setType( "Merge" )

...

t = Transformation()
tc = TransformationClient()
t.setType( "Merge" )
t.setDescription( "EvnDisp3 example" )
t.setLongDescription( "EvnDisplay analysis" )
t.setGroupSize(1)
t.setBody ( job.workflow.toXML() )
t.addTransformation()
t.setAgentType( "Automatic" )
transID = t.getTransformationID()
tc.addFilesToTransformation( transID['Value'], {'particle': 'gamma', 'site':'Paranal', 'analysis_prog': 'sim_telarray', 'thetaP':20.}  )
```

Future plans

- Already discussed last year, see RFC #21:
  - https://indico.cern.ch/event/372717/contributions/1793972/attachments/741943/1017819/PrsentationTS.pdf
  - https://github.com/DIRACGrid/DIRAC/wiki/Transformation-System-evolution
- Motivations for improvement:
  - Large catalog queries may be a bottleneck (experience from LHCb)
    - ☐ Proposal to make the TS fully 'data-driven' by implemeting 'meta-filters'
    - ☐ Work already started
  - Need to support 'chained transformations'
    - ☐ Example: in LHCb chained transformations, *e.g.* Re-processing -> Merging -> Removal, are handled by a dedicated Production System
    - ☐ Proposal to extend the TS to support chained transformations as basis for each community to build its own 'Production System'
  - Agents in the TS work in 'polling' mode
    - ☐ Proposal to use a Message Queuing System complementary to polling