Transformation System report

Luisa Arrabito[1], Federico Stagni[2]

1) LUPM CNRS/IN2P3, France

2) CERN

5th DIRAC User Workshop 27th – 29th May 2015, Ferrara

DIRAC
THE INTERWARE

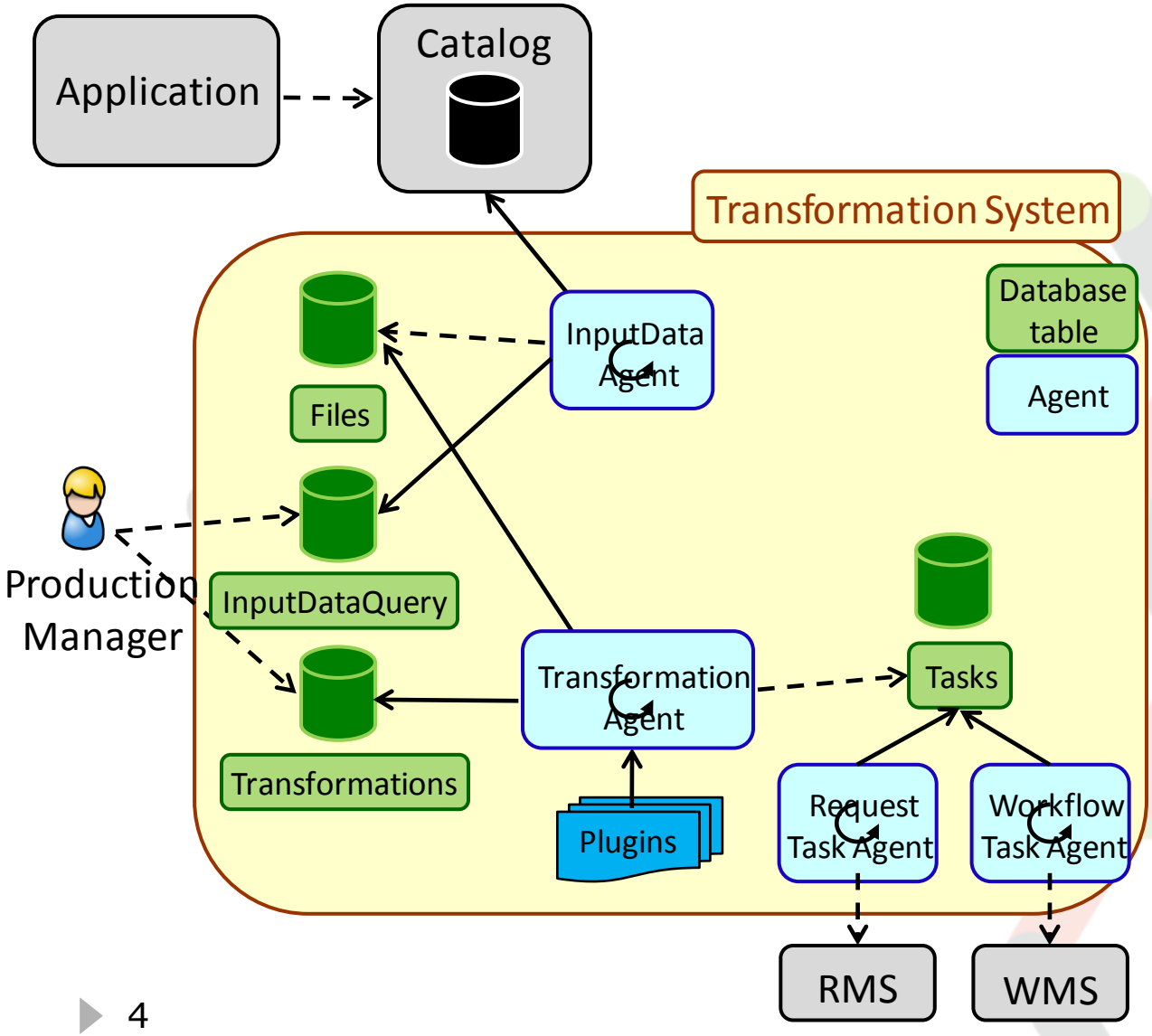
Plan

- What's the Transformation System?
- Transformation System architecture
- How it works in practice?
- Proposal for a new design

What's the Transformation System?

- A DIRAC System as usually comprising:
  - MySQL tables, 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 Request Management System
- Handles input datasets (if present)
  - It interacts with Replica and Metadata catalogs (*e.g.* DFC or external catalogs)
  - Plugins are grouping input files into tasks according to various criteria
- LHCb 'Production System' is built on top of it and CTA is going to do same

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

Transformation Plugins

- Standard():
  - Group files by replicas (tasks created based on the file location)
- BySize():
  - Group files until they reach a certain size (Input size in Gb)
- ByShare()
  - Group files given the share (specified in the CS) and location

For replication:

- Broadcast()
  - Takes files at the source SE and broadcast to a given number of locations

TaskManager Plugins (from v6r13)

- BySE():
  - Default plugin
  - Set jobs destination depending from the location of its input data
- ByJobType():
  - By default, all sites are allowed to do every job
  - The actual rules are freely specified in the CS Operation JobTypeMapping section

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
}
```

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

How it works in practice (III)?

- Create your transformation defining:
  - Type (*e.g.*: MCSimulation, DataReprocessing, Replication)
  - Body (the job workflow to execute, or the request type to execute)
  - Plugin (*e.g.:* ByReplica, BySize, Broadcast, default is Standard)
- Example for a “processing” transformation:

```
from DIRAC.TransformationSystem.Client.Transformation import Transformation
from DIRAC.TransformationSystem.Client.TransformationClient import TransformationClient
from DIRAC.Interfaces.API.Job import Job
j = myJob()
...
t = Transformation( )
tc = TransformationClient( )
t.setTransformationName("Reprocessing 1") # This must be unique
t.setType("DataReprocessing")
t.setDescription("repro example")
t.setLongDescription( "This is the long des reprocessing" ) #mandatory
t.setBody ( j.workflow.toXML() )
t.addTransformation()
t.setStatus("Active")
t.setAgentType("Automatic")
transID = t.getTransformationID()
#tc.addFilesToTransformation(transID['Value'],infileList)
tc.creteTransformationInputDataQuery(transID['Value'], {'particle': 'proton','zenith':20.,'outputType':'corsikaData'})
```

set Type

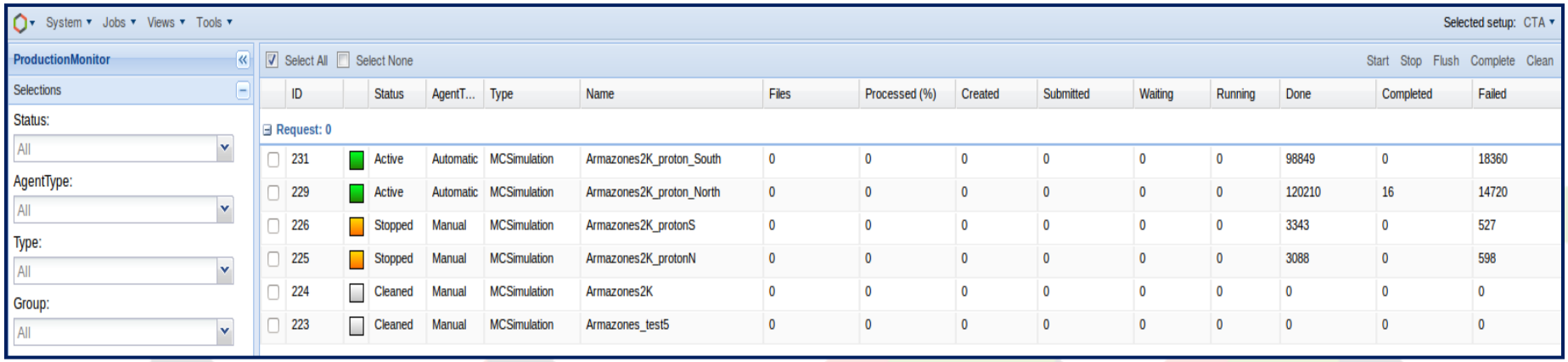
set Body

transformation is created here

set Inputdata

How it works in practice (IV)?

- Monitor (and manage) your transformation

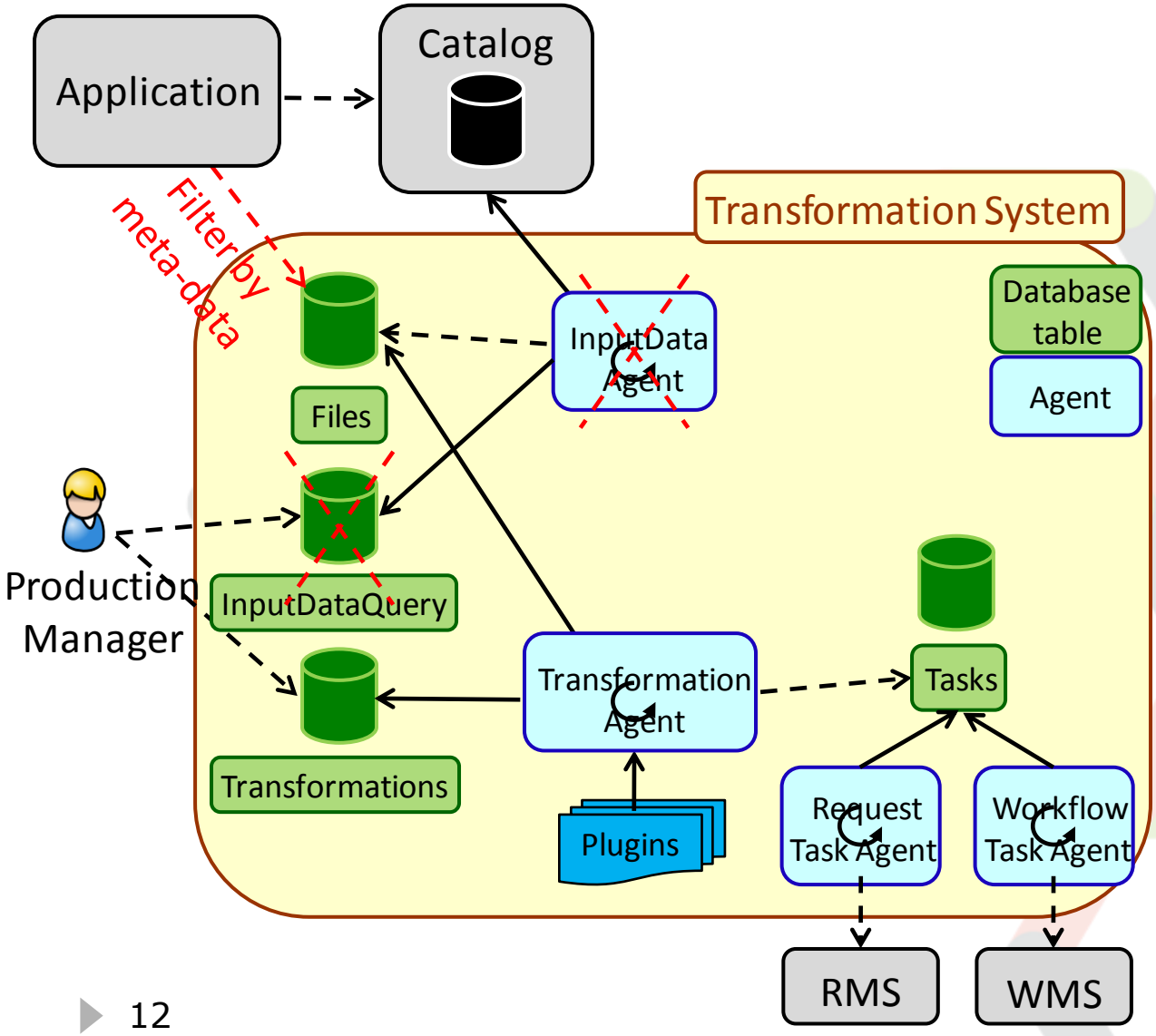


ID	Status	AgentT...	Type	Name	Files	Processed (%)	Created	Submitted	Waiting	Running	Done	Completed	Failed
231	Active	Automatic	MCSimulation	Armazones2K_proton_South	0	0	0	0	0	0	98849	0	18360
229	Active	Automatic	MCSimulation	Armazones2K_proton_North	0	0	0	0	0	0	120210	16	14720
226	Stopped	Manual	MCSimulation	Armazones2K_protonS	0	0	0	0	0	0	3343	0	527
225	Stopped	Manual	MCSimulation	Armazones2K_protonN	0	0	0	0	0	0	3088	0	598
224	Cleaned	Manual	MCSimulation	Armazones2K	0	0	0	0	0	0	0	0	0
223	Cleaned	Manual	MCSimulation	Armazones_test5	0	0	0	0	0	0	0	0	0

Proposal for a new design (I)

- See RFC #21:
  - 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' (see next slide)
  - Job submission could be improved using bulk submission as done for 'parametric jobs'
  - 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 Queueing System complementary to polling

Proposal for a new design (II)



- Use the Catalog interface of the TS
- When new files are registered, a filter based on meta-data is applied
- No need anymore to perform large Catalog queries

Conclusions

- The Transformation System allows to handle massive ‘production’ operations (large number of jobs or requests)
  - Successfully used by LHCb, ILC, CTA…
- LHCb experience shows some scalability problem, essentially due to large queries on the catalog
- Development work has started to make the TS fully ‘data-driven’
- RFC #21 waits for your comments!

BACKUP

Job Workflows

- Job description format
- Enables running "complex" jobs
  - e.g. multiple applications, linked together via input/output data
  - I/O chaining
- description in different formats: XML, JDL, python
  - JDL executable: dirac-jobexec
  - Argument: jobDescription.xml (which is in the Input Sandbox)
- A workflow is composed of steps
  - that are made of modules
  - workflow modules are instantiated by python modules
    - that do the real job
  - parameters at any level

Job description

Application Step 1
Application Step 2
Finalization Step

(for users AND production jobs)

Transformation

Task_1 → Job_1
Task_2 → Job_2
…
Task_n → Job_n

Job workflow

App Step 1
App Step 2
Finalization Step