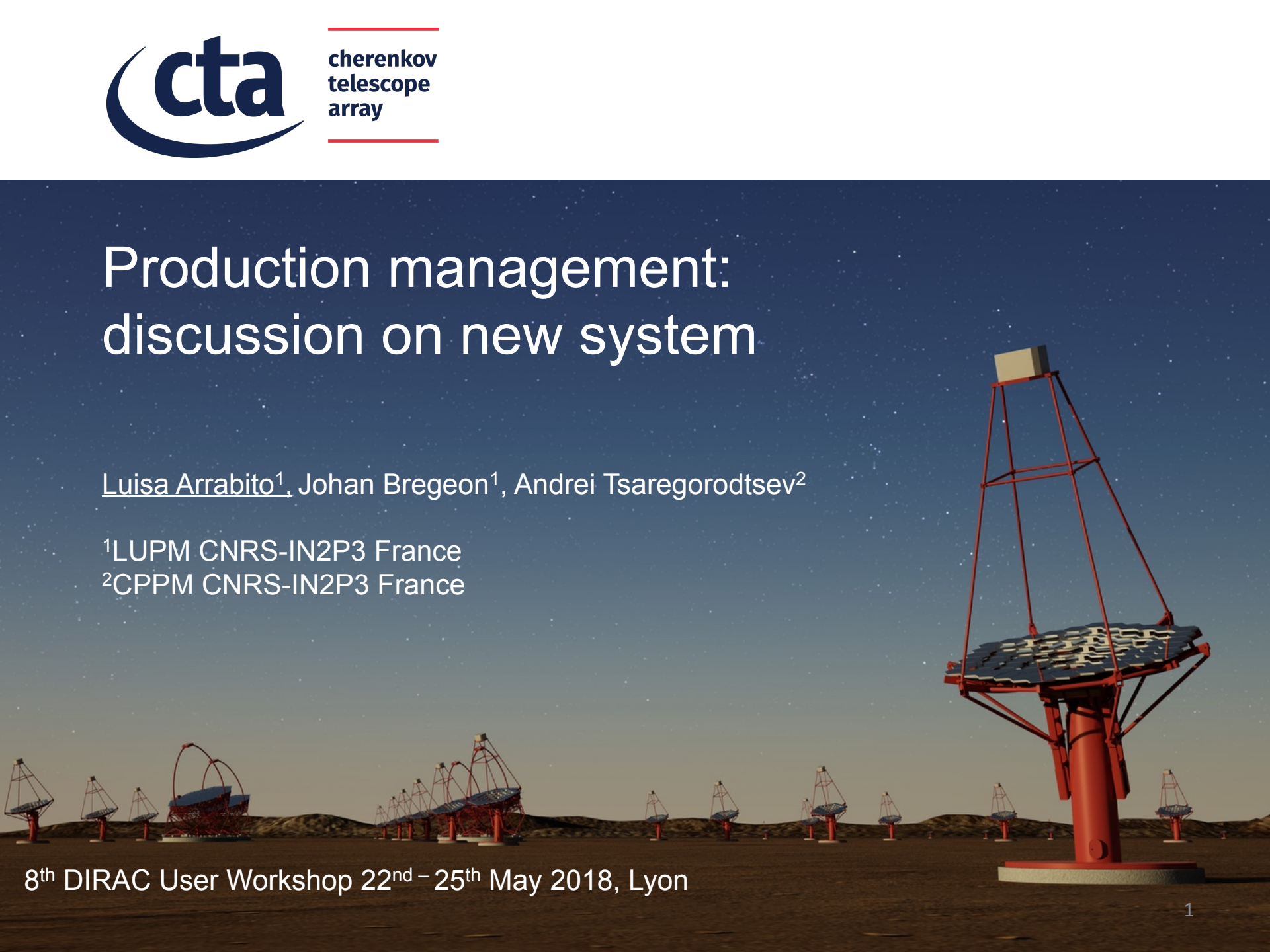
cherenkov telescope array

Production management: discussion on new system

Luisa Arrabito[1], Johan Bregeon[1], Andrei Tsaregorodtsev[2]

[1]LUPM CNRS-IN2P3 France
[2]CPPM CNRS-IN2P3 France



8th DIRAC User Workshop 22nd – 25th May 2018, Lyon

Production Management in CTA

- Production Management in CTA heavily relies on the TS
  - No higher level system or custom scripts
- Using the DFC as replica and meta-data catalog
- The meta-data are set within the job as last step together with output data upload
- Using the TS with meta-data filter for data-processing
  - Simple examples using meta-filters in the tutorial session this afternoon

Production Management in CTA

- Here we call a “production” the set of transformations needed to produce a given (DFC) dataset (intermediate datasets are also produced)
- Once a production is finished we manually create several datasets (associated to the different intermediate steps)
  - Datasets are exposed to end-users for data search/download
- Finally we report the production activity on a summary web page (transformations, datasets, data volumes, etc.)

Production Management in CTA

- Extracted from the production summary page (currently we have 250 datasets)

Transformation Id	dataset	Run	NbFiles (all sub arrays)	NbShower per file	Total Nb of Events	Target Nb of Events	Status	Query to select dataset	DataSet Size on Disk	Comments
282	Paranal_gamma_South	1-5000	~~24547~~ 21820	20000	~~0.10e9~~ 0.09e9	0.1e9	completed	Paranal_gamma_South	~~48.0~~ 42.7 TB	-
283	Paranal_gamma-diffuse_South	1-5000	~~22674~~ 19147	20000	~~0.09e9~~ 0.08e9	0.1e9	completed	Paranal_gamma-diffuse_South	~~11.1~~ 9.4 TB	-
284	Paranal_electron_South	1-5000	~~22824~~ 19417	20000	~~0.09e9~~ 0.08e	0.1e9	completed	Paranal_electron_South	~~10.5~~ 8.9 TB	-
285, 349	Paranal_proton_South	1-90000	~~421038~~ 376301	25000	~~2.11e9~~ 1.88e9	2e9	completed	Paranal_proton_South	~~151.2~~ 134.7 TB	-
287	Paranal_gamma_North	1-5500	~~25668~~ 20839	20000	~~0.1e9~~ 0.08e9	0.1e9	completed	Paranal_gamma_North	~~52.0~~ 42.2 TB	-
288	Paranal_gamma-diffuse_North	1-5000	~~24458~~ 17815	20000	~~0.10e9~~ 0.07e9	0.1e9	completed	Paranal_gamma-diffuse_North	~~12.6~~ 9.2 TB	-
289	Paranal_electron_North	1-5000	~~24544~~ 18304	20000	~~0.10e9~~ 0.07e9	0.1e9	completed	Paranal_electron_North	~~12.0~~ 8.9 TB	-
298, 340	Paranal_proton_North	20000-110000	~~415001~~ 388061	25000	~~2.08e9~~ 1.94e9	2e9	completed	Paranal_proton_North	~~151.2~~ 141.7 TB	-
-	-	-	-	-	-	-	-	-	-	-
501	Paranal_gamma_South_40deg	1-5000	23053	20000	0.09e9	0.08e9	completed	Paranal_gamma_South_40deg	82.1 TB	nb files looks suspicious
502	Paranal_gamma-diffuse_South_40deg	1-5000	23364	20000	0.09e9	0.08e9	completed	Paranal_gamma-diffuse_South_40deg	19.2 TB	-
507	Paranal_electron_South_40deg	1-5000	23260	20000	0.09e9	0.08e9	completed	Paranal_electron_South_40deg	17.9 TB	-
536, 547, 557	Paranal_proton_South_40deg	1-100, 101-5000, 20000-30000	~~122223~~ 116386	50000	~~1.22e9~~ 1.16e9	1e9	completed	Paranal_proton_South_40deg	~~152.6~~ 145.0 TB	multiple transformations
563	Paranal_gamma_North_40deg	20000-25000	~~21079~~ 19276	20000	~~0.09e9~~ 0.08e9	0.08e9	completed	Paranal_gamma_North_40deg	~~78.6~~ 71.8 TB	-
567	Paranal_gamma-diffuse_North_40deg	1-5000	23937	20000	0.10e9	0.08e9	completed	Paranal_gamma-diffuse_North_40deg	20.6 TB	-
580	Paranal_electron_North_40deg	1-5000	~~24322~~ 21186	20000	~~0.10e9~~ 0.08e9	0.08e9	completed	Paranal_electron_North_40deg	~~19.7~~ 17.2 TB	-
584	Paranal_proton_North_40deg	1-48000	207153	25000	1.04e9	1e9	completed	Paranal_proton_North_40deg	138.3 TB	-

Production Management in CTA

- Main limitations
  - Need to monitor tens of transformations at once
  - We don't have an automatic job failure recovery
  - The manual creation of several datasets and the report on the web page is time-consuming and error prone
  - Particular care is needed when defining the meta-data filter of each transformation and the output meta data in the job workflow
    - An error here compromises the validity of the whole production
- The need of a high level Production System is clear for us
- Each of you has developed its own solution
- The idea is to agree on a common system to be used by several communities

Summary from tuesday session

- LHCb
  - A production is described by a template which the PS transforms into productions, i.e. sets of transformations. It has a DB backend and a web interface. It's data-driven but it's quite specific to LHCb also because of the interaction with their bookeeping
- ILC
  - Developed high level scripts chaining transformation together based on a configuration file describing the whole workflow. It's data-driven (using DFC meta-data)
- Belle II
  - Developed a complex system (not meta-data driven). Productions are described in json format. DB backend? Monitor?
- VIP (biomed)
  - Uses their own workflow engine based on the gwendia language. It's data-driven and allows the implementation of very complex workflows. It does not rely on the TS

Toward a common production system

- ‘Main’ commonalities among the different solutions
  - Built on top of the TS
  - Data-driven
  - A ‘template’ used for production description
  - An agent treats inconsistent states (output files vs job status, etc.)
  - There is a DB backend to keep track of the productions
  - Web interface for monitoring or production creation
- Proposal of common system grouping these main features
  - We prepared a wiki page (to become an RFC) to trigger the discussion
    - https://github.com/DIRACGrid/DIRAC/wiki/Production-System

Production System goals and definition

- Execute ‘complex’ workflows composed of several steps in a automatized way
- Similar scope of the Transformation System but at higher level

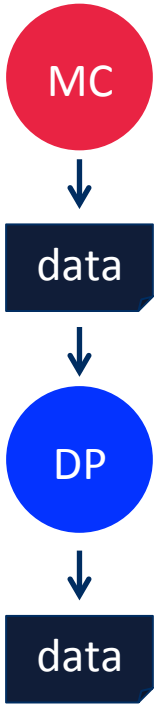
General requirements

- Support the ‘linking’ between transformations
- Support at least simple workflows (sequential, split, merge, others?)
- Support production monitoring (CLI, web)
  - progress monitoring
  - with connection to the Transformation Monitor
- Support production management actions (stop/start/delete/...)
  - Acting on the transformations belonging to the production
- Automatic recover of job and transformation failures
- Keep track of the production activity on the long term
- Want to keep the data-driven concept
- Keep it simple and general

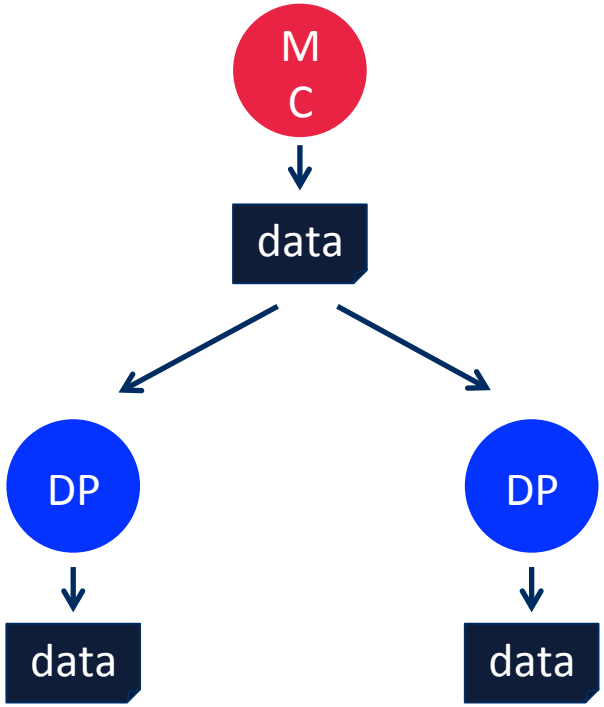
Simple workflow examples

MC = Transformation with no input Data
DP = Data Processing transformation

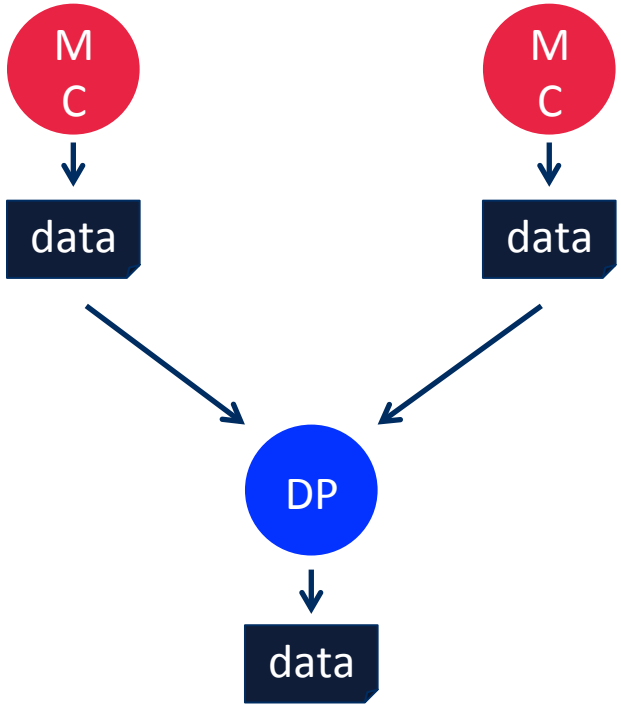
Sequential



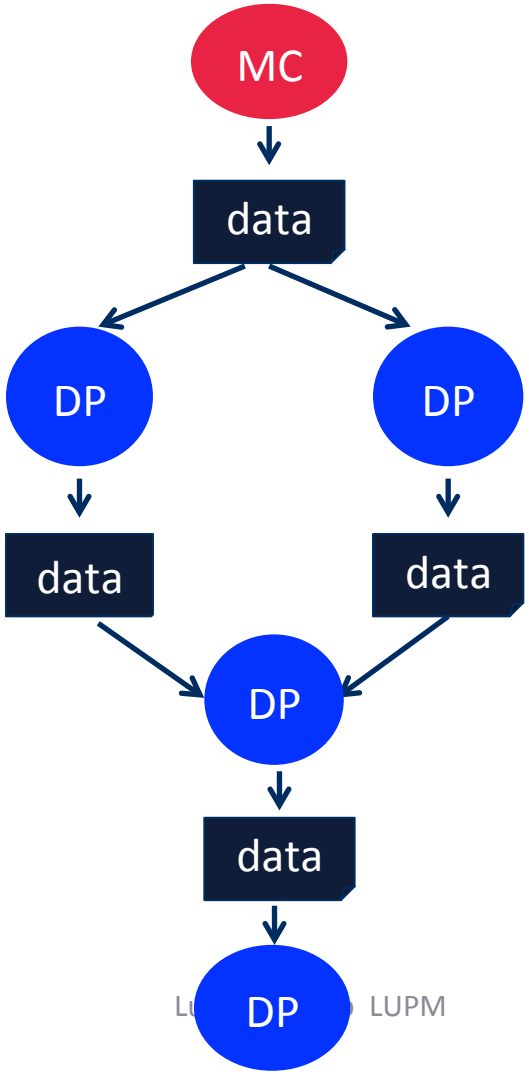
Split



Merge



All combined...



General requirements

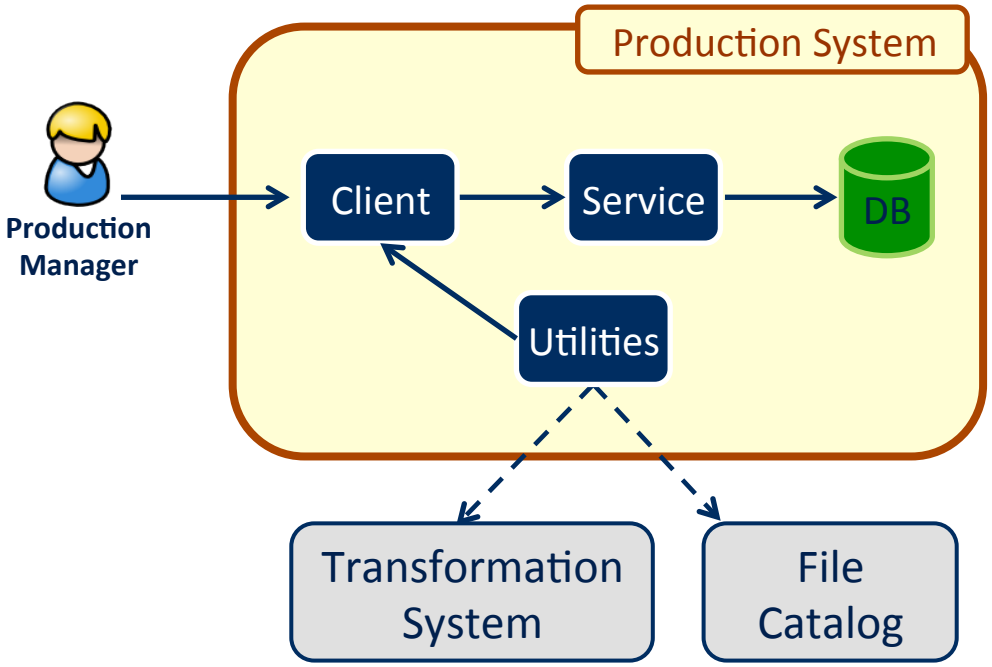
- Support the ‘linking’ between transformations
- Support at least simple workflows (sequential, split, merge, others?)
- Support production monitoring (CLI, web)
  - progress monitoring
  - with connection to the Transformation Monitor
- Support production management actions (stop/start/delete/...)
  - Acting on the transformations belonging to the production
- Automatic recover of jobs, transformations failures and of inconsistent states
- Keep track of the production activity on the long term
- Want to keep the data-driven concept
- Keep it simple and general

Design proposal

- Production definition
  - A set of 'linked' transformations -> Need to agree on what 'linked' means: ,*e.g.*:
    - 2 transformations t1, t2 are 'linked' if they verify:
      InputQuery2 logically intersects OutputQuery1
    - i.e. at least part of the files produced by t1 are potentially input of t2
  - Transformations are linked by meta-data
- Need to enhance the TS with Input/OutputMetaQuery attributes
  - InputMetaQuery: it would replace the current 'FileMask'
  - OutputMetaQuery: the output meta data with their expected range of values
- The idea is that meta-data are defined at the level of the transformation (also for the output)
- Want to have a ProductionDB backend to keep track of productions and the related transformations

Design proposal

- Architecture similar to other DIRAC Systems
  - DB, Service, Agents



Current benchmark implementation

- https://github.com/arrabito/DIRAC/tree/ProdSys_v6r19
- ProductionDB (very simple)
  - 1 table containing production definitions (Productions table)
  - 1 table containing the associations between transformations and productions (ProductionTransformations table)
- Production Manager Service
  - Exposes methods to create/delete/monitor productions (manipulating the Productions table)
  - Exposes methods to associate transformations to productions (manipulating the ProductionTransformations table)

Current benchmark implementation

- Production Client
  - Exposes all the methods to manage productions
    - addProduction, setProductionStatus, addTransformation, getProductions, getProductionTransformations
  - CLI
    - dirac-ps-add-trans-to-prod.py, dirac-ps-get-trans-from-prod.py, dirac-ps-add.py, dirac-ps-get.py, dirac-ps-start.py, dirac-ps-clean.py, dirac-ps-delete.py
- Utilities
  - ProdTransManager :
    - Uses the TS and the PS clients
    - It manages the transformations associated to the productions (start/stop/clean/ delete)
  - ProdValidator :
    - Invoked at production creation time or during production modificaton (i.e. when adding a new transformation to the production)
    - Uses the TS client and the FileCatalog
    - It validates the production definition
- Tests
  - Relatively complete unit and integration tests for different workflow scenarios (sequential, split, merge)
  - Already installed on the CTA instance and small tests done, but not used in production yet

Extracted from the integration test (sequential case)

```
### Create a production
prodName = 'SeqProd'
res = self.prodClient.addProduction( prodName )
self.assertTrue( res['OK'] )
prodID = res['Value']

### Create a MCSimulation transformation with an output meta query and output metadata
outputquery = MetaQuery( {'zenith':{'in': [20, 40]},'particle':'gamma', 'tel_sim_prog':'simtel', 'outputType':{'in': ['Data', 'Log']}}
outputquery = outputquery.getMetaQueryAsJson()
### Not used in the Production validation for the moment
outputmetadata = {'particle':'gamma', 'tel_sim_prog':'simtel', 'outputType':'Data'}

res = self.transClient.addTransformation( 'MCSim', 'description', 'longDescription', 'MCSimulation', 'Standard',
                                          'Manual', '', outputMetaQuery=outputquery, outputMetaData=json.dumps(outputmetadata) )

self.assertTrue( res['OK'] )
MCtransID = res['Value']

### Create the Analysis_step1 transformation
inputquery = MetaQuery( {'zenith': 20, 'particle':'gamma', 'tel_sim_prog':'simtel', 'outputType':'Data'} )
outputquery = MetaQuery( {'zenith': 20, 'particle':'gamma', 'analysis_prog': 'evndisp', 'data_level': 1, 'outputType':{'in': ['Data', '

inputquery = inputquery.getMetaQueryAsJson()
outputquery = outputquery.getMetaQueryAsJson()

res = self.transClient.addTransformation( 'Analysis_step1', 'description', 'longDescription', 'DataProcessing', 'Standard',
                                          'Manual', '', inputMetaQuery=inputquery, outputMetaQuery=outputquery )

self.assertTrue( res['OK'] )
analysis_step1_transID = res['Value']
```

Extracted from integration test (sequential case)

```
### Create the Analysis_step2 transformation
inputquery = MetaQuery( {'zenith': 20, 'particle':'gamma', 'analysis_prog':'evndisp', 'data_level': 1, 'outputType':'Data'} )
outputquery = MetaQuery( {'zenith': 20, 'particle':'gamma', 'analysis_prog': 'evndisp', 'data_level': 2, 'outputType':{'in': ['Data', '

inputquery = inputquery.getMetaQueryAsJson()
outputquery = outputquery.getMetaQueryAsJson()

res = self.transClient.addTransformation( 'Analysis_step2', 'description', 'longDescription', 'DataProcessing', 'Standard',
                                          'Manual', '', inputMetaQuery=inputquery, outputMetaQuery=outputquery )

self.assertTrue( res['OK'] )
analysis_step2_transID = res['Value']

### Add to the production the MCSim transformation
res = self.prodClient.addTransformationsToProduction( prodID, MCtransID, -1 )
self.assertTrue( res['OK'] )

### Add to the production Analysis_step1 transformation with the MCSim transformation as parent trans
res = self.prodClient.addTransformationsToProduction( prodID, analysis_step1_transID, MCtransID )
self.assertTrue( res['OK'] )

### Add to the production Analysis_step2 transformation with the Analysis_step1 transformation as parent trans
res = self.prodClient.addTransformationsToProduction( prodID, analysis_step2_transID, analysis_step1_transID )
self.assertTrue( res['OK'] )

### Get the transformations of the production
res = self.prodClient.getProduction( prodName )
self.assertTrue( res['OK'] )
prodID = res['Value']['ProductionID']

res = self.prodClient.getProductionTransformations( prodID )
self.assertTrue( res['OK'] )
self.assertEqual( len( res['Value'] ) , 3 )

### Delete the production
res = self.prodClient.deleteProduction( prodName)
self.assertTrue( res['OK'] )
```

Current benchmark implementation

- Some limitations of the current implementation
  - The production validation done by the ProdValidator Utility is extremely simplistic
  - Now the validation of the production is simply done when new transformations are added
    - There is no ‘final’ validation to check if all transformations are connected
  - The progress of a production is not yet defined, neither monitored
  - Job failure recovery and consistency checks should be integrated in the TS
  - The web interface is missing
  - Others?

Other questions

- Should we rely on datasets instead of meta-data queries?
- Datasets could be optionally automatically created at the end of the productions?
- We haven't talked about data-handling transformations. Should they also be included in the productions?
- Other?

Backup

Extracted from the integration test (merge case)

```
### Create a MCSimulation transformation with an output meta query and output metadata
outputquery = MetaQuery( {'zenith':20, 'particle':'gamma', 'tel_sim_prog':'simtel', 'outputType':{'in': ['Data', 'Log']}} )

### Create a second MCSimulation transformation with an output meta query and output metadata
outputquery = MetaQuery( {'zenith':40, 'particle':'gamma', 'tel_sim_prog':'simtel', 'outputType':{'in': ['Data', 'Log']}} )

### Create the Analysis_step1 transformation
inputquery = MetaQuery( {'zenith': {'in': [20, 40]}, 'particle':'gamma', 'tel_sim_prog':'simtel', 'outputType':'Data'} )

### Add to the production the MCSim transformations
res = self.prodClient.addTransformationsToProduction( prodID, [MCSimAtransID, MCSimBtransID], -1 )

### Add to the production Analysis_step1 transformation with MCaSim and MCbSim as parent trans
res = self.prodClient.addTransformationsToProduction( prodID, analysis_step1_transID, [MCSimAtransID, MCSimBtransID] )
```