Contents

1. Used modules and libraries

Technology	Description
Jinja2	It is a modern and designer-friendly templating language for Python, which we extensively used for the templating of the CMS SoftWare (CMSSW) configurations. It rapidly fills out the templates and gives the option of special functions to be fired upon obtaining a specified token in template. Much faster than the Python string.Template class.
tokenize	Module designed for obtaining structures of tokens out of some readable objects. Used by us to read the CMSSW templates and remove unnecessary comments before the Jinja2 is to be used.
cStringIO	Used together with **tokenize** for special access to string variables. It is very fast thank to being rewritten to C (hence the "c" letter at the beginning).
optparse	Module providing an easier way of managing the command-line of application
pyTotem	Python library with overloaded Python system functions and utility methods for TOTal Elastic and diffractive cross section Measurement (TOTEM) programming environment

2. Architecture

2.1. Solution description

This system is to replace the three currently used tools and so it has to work better and be as easy in usage as it's possible.
The tool has modular construction - so that it will be easier to maintain and extend it in future (as well as change some of the parts easily if it's needed). It has few ways of configuring:

- the main configuration file for the program (in which the **configuration_logic.py** module is to be imported and used). User may specify a number of options here:
  - whether he wants to run the simulation or the data reconstruction,
  - the output location for all operations,
  - computing cluster parameters,
  - path to CMSSW location,
  - workspace directory specification,
  - specialized simulation options,
    - number of events to generate per job
    - number of jobs to be used
  - specialized reconstruction options,
    - the way to split the configurations in regards of number of events, amount of files to have as an input for reduce phase

    - ⋆ paths to input data
    - ⋆ files to exclude from input

- **Parallel templates** regarding the parallel phases of the reconstruction or simulation. A person can choose the analyzer's he wants to use, the loggers and other modules specified by CMSSW framework.

- **Sequential templates**, in which user configures the final output of the chain of operations. A number of CMSSW modules can be chosen here (same as for parallel phase template) to get preferred output.
  As output the user may have:

  - ○ .root file generated out of multiple .root files
  - ○ Ntuple file as an output of merging multiple ntuples (rarely chosen as it can be easily done through ROOT environment)
  - ○ Ntuple generated through the use of TotemNTuplizer (which as input takes multiple .root files)
  - ○ one or more .root files with histograms and plots generated by the use of some of arbitrary analyzer modules (like HitsDistribution module), which as input take .root files

- simple command-line options which can be used during the launch of TOTEM Configuration Splitter Improved (TOTCSI).

2.2. Main parts

The system consists of four main parts, presented on figure 1. Most of the parts can be replaced by new ones if needed, however be careful with that for the modules of Core part extensively use the TOTCSI configuration files.

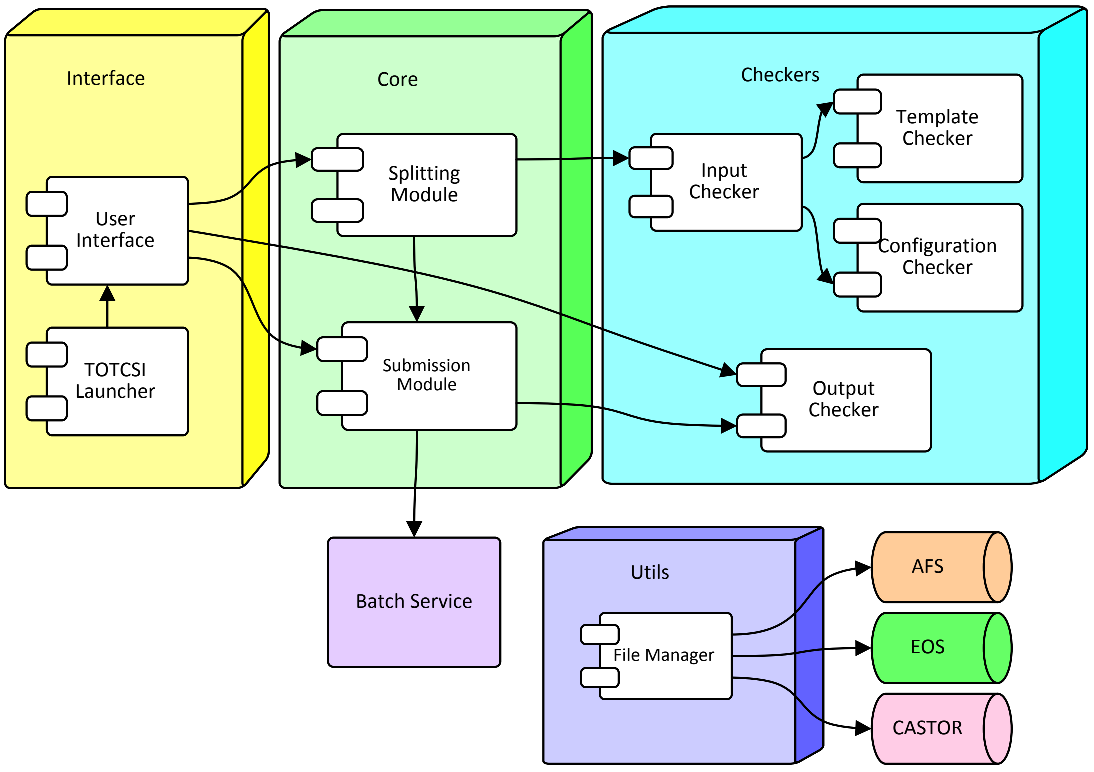

Figure 1: TOTCSI architecture overview.

2.2.1. Interface

The interface block is responsible for communication with the user and interpreting his commands. Additionally, it setups the environment for the rest of the system during its start-up and keeps the tool output messages human-readable. Most of its functions are realized by following two parts.

- *The TOTCSI Launcher* — its main purpose is to discover which version of Python is currently being used (as users could choose different ones at the CERN terminals). Afterwards, it calls the necessary set of shell commands in order to prepare the environment for following actions. This process is essential for two reasons — it ensures that the active interpreter supports all utilized language features and enables the possibility to correctly import and run the CMSSW-connected modules (which is vital in the configuration checking procedure). In the end, it starts the central UI system part. The Python module which takes care of those actions is named **totcsi_launcher.py**.

- *The User Interface* — it parses the command line and interprets the orders provided by the user, then decides which of the other modules should be called later. Also, it handles providing the help information when needed and printing messages describing the errors and exceptions, if any occur. Related module is named **main.py**.

2.2.2. Core

The application core elements manage the most important of the system tasks— splitting the CMSSW configuration templates and submitting jobs to the LXBATCH computing cluster.

- Splitting Module takes care of splitting of the templates given by user and filling them with needed additional information and saving them to workspace directories as valid CMSSW configurations. Besides that it produces the small configurations for local test-runs. Python module's name is **splitter.py**.
- Submission Module is used to produce the shell scripts for submission, resubmission and local tests of the CMSSW configurations. Functions needed for those tasks are programmed in module **submitter.py**.

2.2.3. Checkers

Checkers block consists of modules which are built for checking of the correctness of given configurations and templates, produced files and jobs' outputs. Some of the functions of the modules are called automatically during some of the activities done by TOTCSI (e.g. looking through given configurations), others (like checking of the outputs) can be called by specified command.

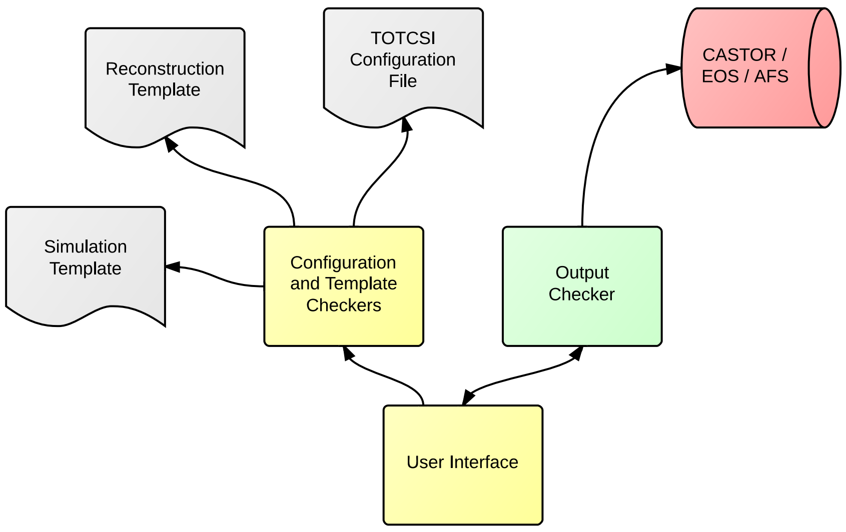

Figure 2: Architecture of the checking.

- Template Checker is used to produce smaller, computationally undemanding CMSSW configurations and shell scripts for running them locally by user. Such action let's user know if the configuration produces expected files. All produced files and configurations are stored in workspace under folders named test_directory. Both *splitter.py* and **submitter.py** modules are used in creation of those testing files.

- Configuration Checker wraps the configuration given by user and looks through paths specified by user, calls the necessary functions and produces structures needed by other modules. The file containing function for this part of system is **configuration_checker.py**.

- Output Checker checks if the output data exists and is not corrupted, if there is anything wrong, it creates lists with configurations that should be rerun which is later used during generation of resubmission scripts. All functions for this module are contained in **output_checker.py**.

2.2.4. Utils

This block consists of a number of useful modules, which provide often used functions for data access, parsing, naming and others.

- The main part of this block is File Manager, which relates to functions needed for file and directory creation, removal and access on Andrew File System (AFS), CERN Advanced STORage manager (CASTOR) and Exploration Of Storage (EOS). It's provided through the **files_manager.py** module.

- The **others.py** contains many functions that did not fit in other modules but are very needed and constantly used. It has procedures for parsing of the input data given by user in TOTCSI configuration.

- All messages (in form of Python strings) and functions regarding informing the user of what's done by TOTCSI reside in **messages.py** module.

- Exceptions which can be thrown inside TOTCSI are defined in **exception.py** module. All the exceptions thrown are caught in *main.py* module.

- Very important module, which is partly the configuration of TOTCSI (for example it contains the naming convention of the directories and files) is named **properties.py**. All of the Core modules and interface modules make use of it, as it provides important definitions and persists the name of the TOTCSI configuration which is used as the sub-directory inside the main workspace directory. It is built of few important "parts".

- Files and directories (naming and paths) for TOTCSI

```
# file storing all paths from CASTOR
CASTOR_FILE = "CASTOR_paths.txt"
# name of file containing paths to input data,
# it's later used to retrieve the paths for splitting
INPUT_PATHS_FILE = "input_files"
# same as INPUT_PATHS_FILE, but for files
# that should be excluded from input_data
EXCLUDED_PATHS_FILE = "excluded_files"
# main module file placement
MAIN_MODULE_FILE = os.path.join("totcsi", "main.py")
# where to start looking for data in CASTOR
CASTOR_BASE_DIRECTORY = "/castor/cern.ch/totem/LHCRawData"
# main part of the name of simulation directory
SIMULATION_DIRECTORY = "sim"
# main part of the name of reconstruction directory
RECONSTRUCTION_DIRECTORY = "run"
# main name of directory used to store files used for parallel part
PARALLEL_DIRECTORY = "parallel"
# name of directory used to store files used for sequential part
SEQUENTIAL_DIRECTORY = "sequential"
RENDER_DIRECTORY = "submission_files"
LOG_DIRECTORY = "LOG"
OUTPUT_TOTCSI_DIRECTORY = "TOTCSI"
MAIN_LOG_NAME = "totcsi_main.log"
EXPECTED_OUTPUT_NAME = "expected_output"
EXPECTED_OUTPUT_TYPE = "txt"
MISSING_OUTPUT_NAME = "missing_output"
CORRUPTED_OUTPUT_NAME = "corrupted_output"
RESUBMISSION_CONFIGURATIONS_NAME = "configs_for_resubmission"
TEST_DIRECTORY = "test_directory"
```

- Shell commands

```
# command for sourcing default variables for CMSSW
SOURCE_CMS_DEFAULTS_BASH = "source /afs/cern.ch/cms/cmsset_default.sh"
# command for preparing CMSSW runtime
SCRAM_CMS_RUNTIME_BASH = "eval `scram runtime -sh`"
```

- Output naming

```
FN_DEFAULT_DIRECTORY = "."
FN_DEFAULT_NAME = "totcsi"
FN_DEFAULT_NUMBER = 0
FN_NTUPLE = "ntuple"
FN_HISTOGRAM = "hist"
FN_CONFIGURATION = "config"
```

- General properties

```
# name of the configuration used
#(it's replaced in main.py when TOTCSI is used)
CONFIGURATION_NAME = None
force = False
TOTCSI_VERSION = "TOTCSI 2.0" # TOTCSI current version
INTERPRETER = "python"
SIMULATION_TASK = "Simulation"
RECONSTRUCTION_TASK = "Reconstruction"
LOAD_FROM_FILE_MARKING = "read_list_from_file_load"
LOAD_FROM_RUN_NUMBERS_MARKING = "paths_from_run_numbers_load"
```

- List of TOTCSI commands

```
READ_CASTOR_CMD = frozenset(["readCastor", "rc"])
FIND_PATHS_CMD = frozenset(["findPaths", "fp"])
GENERATE_SUBMISSION_CMD = frozenset(["generateSubmission", "gs"])
GENERATE_RESUBMISSION_CMD = frozenset(["generateResubmission", "gr"])
GENERATE_LOCAL_TESTS_CMD = frozenset(["generateTests", "gt"])
CHECK_PARALLEL1_CMD = frozenset(["checkParallel1", "cp1"])
CHECK_PARALLEL2_CMD = frozenset(["checkParallel2", "cp2"])
CHECK_PARALLEL3_CMD = frozenset(["checkParallel3", "cp3"])
CHECK_SEQUENTIAL_CMD = frozenset(["checkSequential", "cs"])
CHECK_CMD = CHECK_PARALLEL1_CMD | CHECK_PARALLEL2_CMD |
          CHECK_PARALLEL3_CMD | CHECK_SEQUENTIAL_CMD
TOTCSI_COMMANDS = READ_CASTOR_CMD | FIND_PATHS_CMD |
          GENERATE_SUBMISSION_CMD | GENERATE_RESUBMISSION_CMD |
          GENERATE_LOCAL_TESTS_CMD | CHECK_CMD
```

- Jinja2-related properties (template keywords and filter names)

```
# Jinja2 filter names
RANDOM_FILTER = "rnd"
NAME_FILTER = "name"
NTUPLE_FILTER = "ntuple_name"
HISTOGRAM_FILTER = "histogram_name"
# configuration template keywords
CTK_SEED = "seed"
CTK_EVENTS_NO = "number_of_events"
CTK_EVENTS_SKIP = "skipped_events"
CTK_OUTPUT = "output"
CTK_INPUT = "input"
```

3. Code structure

One could learn the details of application shape by browsing through API included as an attachment. Designed as an interactive website (activated by opening the index.html file in *code documentation* directory) it provides an easy way of obtaining extensive knowledge about solutions used in the system. To generate the documentation developer can use the Sphinx tool (it is already configured in developer versions of TOTCSI).

4. Map-reduce

The tool uses the Map-Reduce approach (used by the previous scripts as well), which is excellent when used in environments capable of parallelizing the jobs (such as batch services).

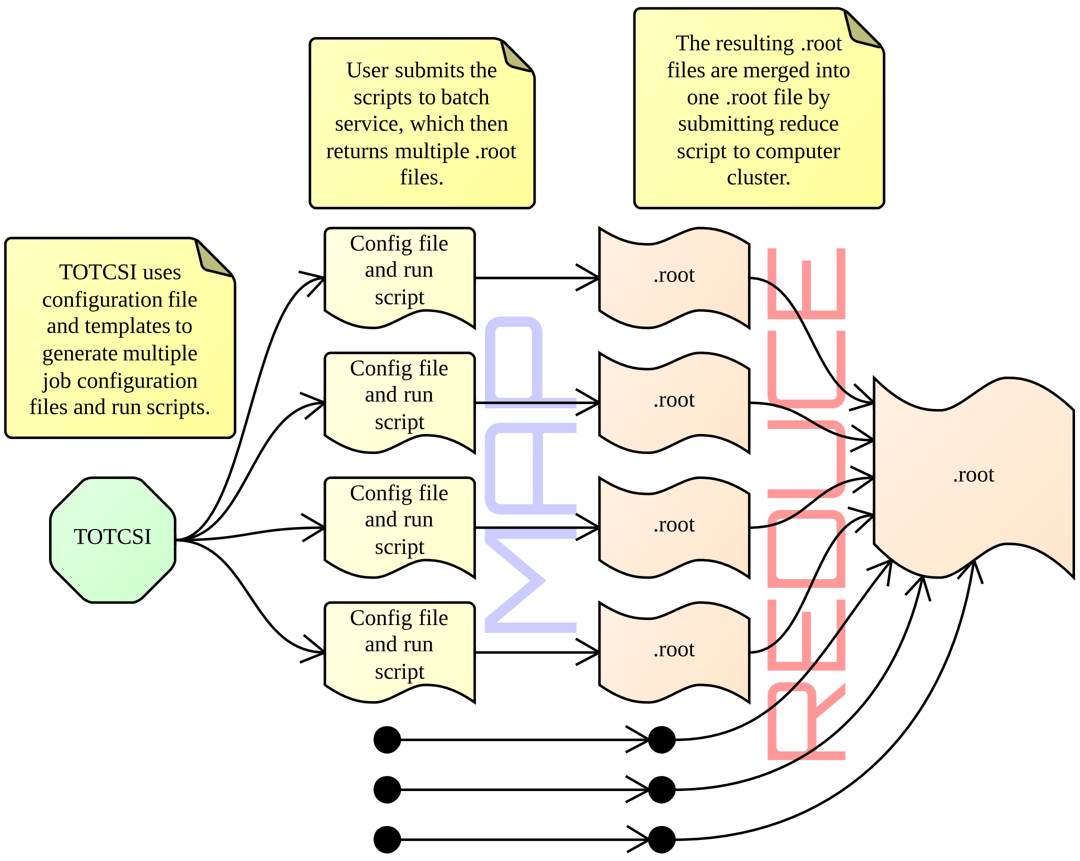
User submits the scripts to batch service, which then returns multiple .root files.
The resulting .root files are merged into one .root file by submitting reduce script to computer cluster.
TOTCSI uses configuration file and templates to generate multiple job configuration files and run scripts.
TOTCSI
Config file and run script
.root
MAP
REDUCE

Figure 3: Map-reduce simulation example.

Data can be mapped in more than one way:

- multiple raw-data files may be mapped into many .root files (one job per one file)
    - additionally raw-data files can be used to produce ntuples or other .root files with use of CMSSW analyzers
- a number of raw-data files might be put in one run and as an outcome we get one .root file - in this case the reduce operation is a part of mapping.

Reduce operation can vary:

- many .root files can be reduced to one big .root in one run
- multiple Ntuples may be merged into single Ntuple using one job
- a number of .root files might produce Ntuple with the use of TotemNTuplizer CMSSW module

5. Environment

Currently (checked on 18.01.2013) there is an issue with the environment in which TOTCSI is deployed. The Python paths change if user runs commands specified by other tools (sourcing the CMSSW framework causes the environmental variables to change). If the tools is called by

```
python totcsi
```

without user sourcing the CMSSW configurations before, the Python used is 2.4, which does not have necessary modules. On the other hand the same command will work fine after the sourcing, as the Python called will be version 2.6.4 . The command

```
python26 totcsi
```

Will give correct results when used without sourcing. After changing the environment variables this Python is stripped out of all modules, so is almost unusable.
Because of those problems we decided to do two things:

- call Python through bash launching script, which will first check if the environment was changed or not and then start TOTCSI with the right command
- in the called module (**totcsi_launcher.py**) set up the environment variables (e.g. paths to needed modules) and run the necessary bash commands.

6. Commands

User can call number of commands and options while using the TOTCSI. Apart from the full names of commands we are giving users the shorter versions for easier usage. There are few options which can be specified as well. Additionally, few of the system activities are activated by running the appropriate bash script file.

6.1. Possible commands

- **readCastor** (rc) — checks the current status of CASTOR and saves information about it into special cache file. It is accomplished by recursive search through all directories containing TOTEM experiment data and listing their contents.
- **findPaths** (fp) — searches through all known files containing raw data (obtained by readCastor command) and prepares the input file list for the splitting procedure. Whether the file should be included or not is decided on the basis of provided criteria (for example: numbers of runs that should be reconstructed).
- **generateTests** (gt) — creates a set of bash script files allowing to check the quality of provided configuration. The test computation will behave the same way the real one does, but the number of processed events would be greatly reduced (in order to quicken the calculations), the output would be saved in user directory on AFS and the job would not be send to cluster, but executed locally instead.
- **generateSubmission** (gs) — creates a set of bash files and CMSSW configurations needed to submit the user-requested jobs to LXBATCH cluster. The process is conducted in two

phases. The first one is called splitting and renders the Python modules that specify the shape of the planned computing process (by filling the configuration template). The second one prepares a hierarchy of scripts, that would be later used to submit jobs (all of them or given part, depending on chosen script).

- **checkParallel1** (cp1) — verifies whether all the predicted output files from parallel phase are present. In case of discovering a missing one it informs the user and saves the data needed for possible resubmission.
- **checkParallel2, checkParallel3** (cp2, cp3) — fulfills the same task that checkParallel1 command, but instead of the first parallel phase the second or third one output is checked.
- **checkSequential** (cs) — fulfills the same task that checkParallel1-3 command, but for sequential phase.
- **generateResubmission** (gr) — creates resubmission scripts similar to those created by generateSubmission script, which can be used by user to resubmit jobs to computer cluster. It produces only the highest and mid-level scripts in hierarchy of scripts (by reusing the configurations and scripts created by generateSubmission).

6.2. Possible options

- -h / --help — shows the possible commands with explanations
- -c / --config= — user specifies the path to TOTCSI configuration that he wants to use, it has to be used during all commands
- -q / --quiet — turns off the information of current state of work in TOTCSI
- -d / --debug — turns on the full stack trace of errors which are caught
- -f / --force — files in workspace will be overwritten if it's needed by command (for example the configuration files or shell scripts)

6.3. Using TOTCSI

TOTCSI can be used in two ways:

- as a service, as it can be deployed in one directory accessible by users and every user-specified information will be saved in the command-calling person's home folder,
- when downloaded, a user can have his own copy of TOTCSI set-up in directory of his own choosing. It can be downloaded from SVN repository.
  - project trunk
    ```
    svn co svn+ssh://svn.cern.ch/reps/totem/trunk/offline/cmssw/tools/config_splitter
    ```
  - tag for version TOTCSI 1.0
    ```
    svn co svn+ssh://svn.cern.ch/reps/totem/tags/TOTCSI/TOTCSI_1_0 config_splitter
    ```

◦ tag for version TOTCSI 1.1

```
svn co svn+ssh://svn.cern.ch/reps/totem/tags/TOTCSI/TOTCSI_1_1 config_splitter
```

◦ tag for version TOTCSI 2.0

```
svn co svn+ssh://svn.cern.ch/reps/totem/tags/TOTCSI/TOTCSI_2_0 config_splitter
```

The typical TOTCSI usage is shown on the activity diagram (figure 4).

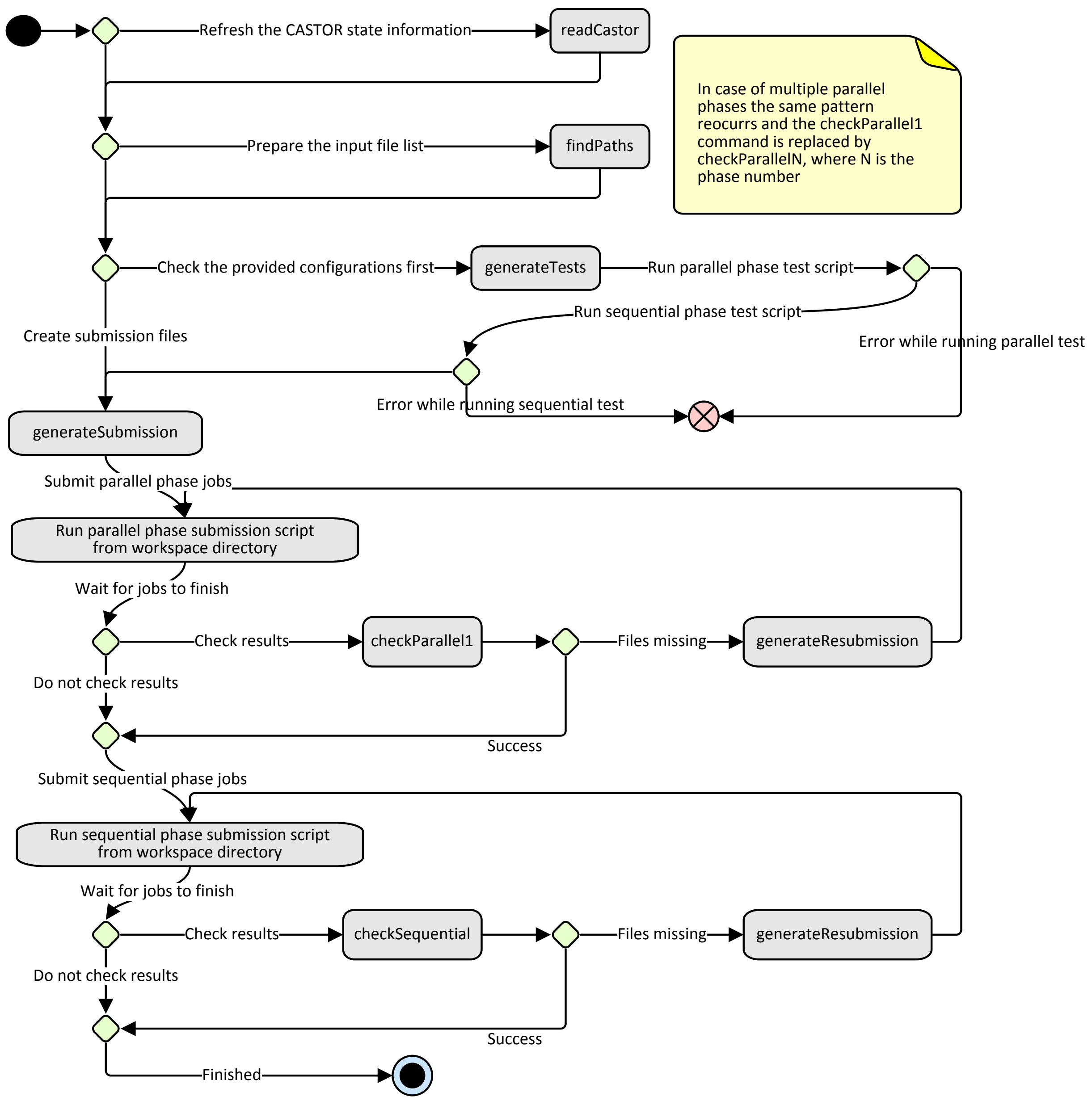
Refresh the CASTOR state information
readCastor
In case of multiple parallel phases the same pattern reocurrs and the checkParallel1 command is replaced by checkParallelN, where N is the phase number
Prepare the input file list
findPaths
Check the provided configurations first
generateTests
Run parallel phase test script
Run sequential phase test script
Error while running parallel test
Create submission files
Error while running sequential test
generateSubmission
Submit parallel phase jobs
Run parallel phase submission script from workspace directory
Wait for jobs to finish
Check results
checkParallel1
Files missing
generateResubmission
Do not check results
Success
Submit sequential phase jobs
Run sequential phase submission script from workspace directory
Wait for jobs to finish
Check results
checkSequential
Files missing
generateResubmission
Do not check results
Success
Finished

Figure 4: TOTCSI 2.0 activity diagram.

6.4. Generated scripts

- *submit_RUNNUMBER_PHASE.sh* — sends all the jobs connected with given run (or given simulation) to LXBATCH cluster. Also takes care of saving the process logs and copying the results into desired directories (if necessary).

- *main_submit_PHASE.sh* — submits every job belonging to given phase to computing cluster. The task is accomplished by running the sequence of lower level scripts (*submit_RUNNUMBER_PHASE.sh*).
- *resubmit_RUNNUMBER_PHASE.sh* — works similarly to *submit_RUNNUMBER_PHASE.sh*, although it only submits the jobs which output files are missing or corrupted (this knowledge is gained by checking the configs_for_resubmission files produced by *checkPHASE* command).
- *main_resubmit_PHASE.sh* — it is used as *main_submit_PHASE.sh* for *resubmit_RUNNUMBER_PHASE.sh* scripts.

7. Directories structure

TOTCSI defined its output and workspace directory trees, so that it can easily check the correctness of files, copy them and delete if needed. The naming conventions can be changed in **properties.py** module.

7.1. Workspace directory

Workspace main directory is specified by user in TOTCSI configuration, it is used by the tool to save all generated files (scripts, logs, state after calls). The overall structure of workspace can be seen on figure 5.
In the workspace TOTCSI creates sub-directories, which name is taken from the naming of the configuration provided by user (i.e. if user provides the configuration *my_reconstruction.py* the extracted name for sub-directory will be *my_reconstruction*). When the command **readCastor** is used, the tool generates file named *CASTOR_paths.txt*, which consists of all found path to **.vmea** files from */castor/cern.ch/totem/LHCRawData* directory tree (except the backups). As there is only one such file per workspace, it can be reused by multiple different configurations and should be updated only, when there are new files in the *LHCRawData* folder.
Structure of configuration sub-directory:

- *main_submit_PHASE.sh* and *main_resubmit_PHASE.sh* — described in Generated scripts section
- *input_files* and *excluded_files* — two files which are produced from the *config.reconstruction.input_data* and *config.reconstruction.files_to_exclude* parameters of configuration. They are generated when the **findPaths** command is called, or automatically when the user provided paths directly to the **.vmea** files; function taking main part in creation of those files can be found in **others.py** module and is named **parse_data_list(data_list, where_to_save)**
- *mainlog.log* — main log for TOTCSI operations
- *runNUMBER/simNUMBER* sub-directories — they resemble the division of tasks for TOTCSI; each NUMBER means an autonomous instance of submission, resubmission and test scripts and files; every of those folders has similar structure

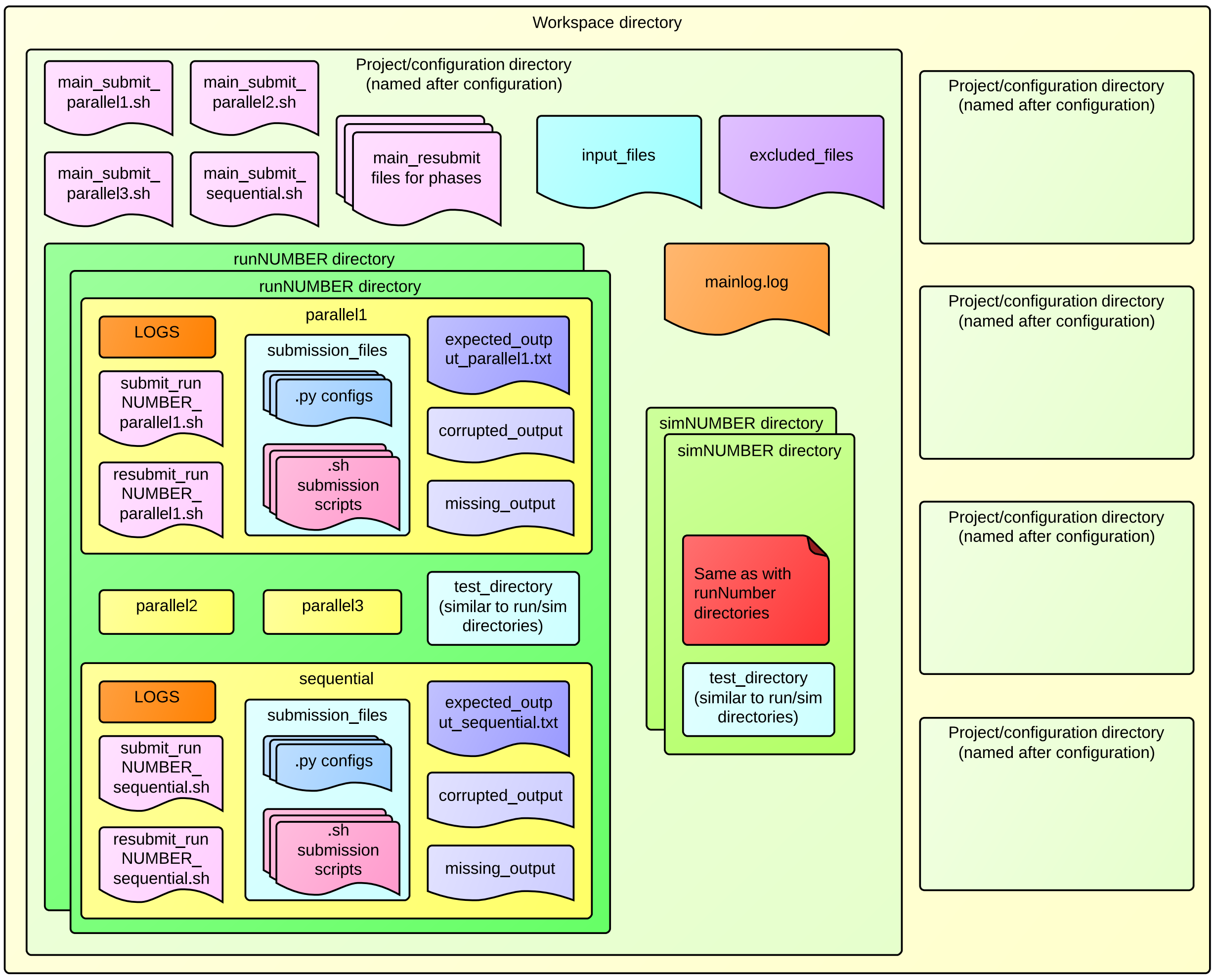
Workspace directory
Project/configuration directory (named after configuration)
main_submit_parallel1.sh
main_submit_parallel2.sh
main_submit_parallel3.sh
main_submit_sequential.sh
main_resubmit files for phases
input_files
excluded_files
mainlog.log
runNUMBER directory
runNUMBER directory
parallel1
LOGS
submit_runNUMBER_parallel1.sh
resubmit_runNUMBER_parallel1.sh
submission_files
.py configs
.sh submission scripts
expected_output_parallel1.txt
corrupted_output
missing_output
parallel2
parallel3
test_directory (similar to run/sim directories)
sequential
LOGS
submit_runNUMBER_sequential.sh
resubmit_runNUMBER_sequential.sh
submission_files
.py configs
.sh submission scripts
expected_output_sequential.txt
corrupted_output
missing_output
simNUMBER directory
simNUMBER directory
Same as with runNumber directories
test_directory (similar to run/sim directories)
Project/configuration directory (named after configuration)
Project/configuration directory (named after configuration)
Project/configuration directory (named after configuration)
Project/configuration directory (named after configuration)

Figure 5: Workspace directory structure.

- *PHASE* directory — can be either **parallel1**, **parallel2**, **parallel3** or **sequential** and contains files associated with the specified phase (logs, scripts etc.)
  - ⋆ *expected_output_PHASE.txt* — file containing information about the expected output of running the splitted jobs on this computer cluster, more information in Persistence section
  - ⋆ *submit_runNUMBER/simNUMBER_PHASE.sh* and *resubmit_runNUMBER/simNUMBER_PHASE.sh* — described in Generated scripts section
  - ⋆ *missing_output* and *corrupted_output* — files produced by **checkPHASE**; any result of the job that is in expected list but is not in the output directory will be added to *missing_output* (and if the result is found to be invalid it's added to *corrupted_output*); the file is used as informative document for user
  - ⋆ *LOGS* directory — contains logs of the submitted jobs, which are copied to the output directory as well
  - ⋆ *submission_directory* — contains both the CMSSW configurations and submission scripts for jobs (the lowest-level bash scripts in hierarchy); files contained in this

folder are produced through calling the **generateSubmission** command

- *test_directory* — produced by **generatesTests** command, it has similar structure as the runNUMBER/simNUMBER directories, however it contains special shell script for local test-run of small CMSSW configuration; the output of this run will reside in *test_directory* as well (as it's relatively light)

7.2. Output directory

The directory structure (see figure 6) for the output of all parallel and sequential phases. All relevant files needed by submission and produced by jobs will be stored there. The paths to those directories have to be specified by the user in the TOTCSI configuration file and can point to the AFS, CASTOR or EOS.

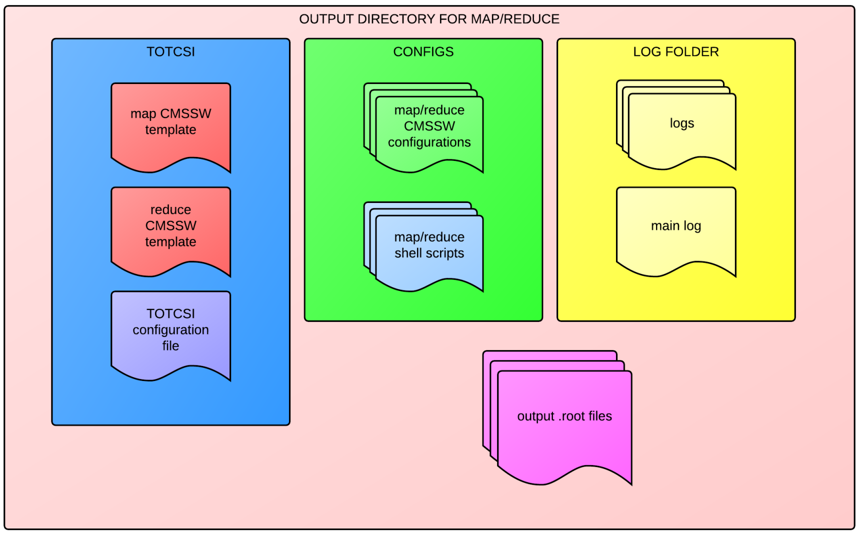

Figure 6: Output directory structure.

Explanation of contents:

- **output .root files** are the main and most important part of the output folder. They are the results of the operations carried out by the computing cluster
- **LOG folder**
  - main log has information about the CMSSW and TOTCSI version
  - logs of the standard and error output of the jobs
- **CONFIGS folder**
  - scripts used for submission
  - CMSSW configuration generated during splitting
- **TOTCSI folder**
  - TOTCSI configuration file (given at the start of the whole workflow by user)
  - the used CMSSW template created by user and passed to TOTCSI in configuration

8. Persistence

A number of TOTCSI features are impossible to implement without the ability to save some information about the system and store them between the commands execution. This requirement is fulfilled by using a set of special files, serving as the simple, but efficient and flexible persistence mechanism.
A good example of such use of persistence files is storing of the knowledge about the current CASTOR state. The process of traversing recursively through whole directory tree belonging to TOTEM experiment consumes huge amount of time and having to repeat it often would cause the usage of the software tiresome for users. Therefore, the obtained results are written into *CASTOR_paths.txt* and treated as up-to-date, until another **readCastor** command is given.
Similar reasons are the case of *input_files* and excluded_files existence. Selecting the proper path subset from data stored in *CASTOR_paths.txt* is a time consuming activity and one should avoid doing it unnecessarily. Moreover the input_data and excluded_files properties from TOTCSI configuration can be specified in many different ways, which upon calling of the **findPaths** command will end up in unified state (list of paths) in those two files.
On the other hand, *configs_for_resubmission* is used, because the need arose to connect the *check* commands with **generateResubmission**. It contains the list of the CMSSW configurations that failed to produce the declared output. And the information about the shape of the proper output itself is stored in the *expected_output* file.
Additionally, after performing the checking procedure, the files that turned out to be invalid or not present are mentioned in *missing_output* and *corrupted_output* files, that could be later utilized by the user.

Glossary

.root file format used by the ROOT software.

.vmea raw TOTEM experiment data file format.

AFS Andrew File System.

Andrew File System a distributed networked file system used by LXPLUS cluster machines, presenting a homogeneous, location-transparent file name space to all the client workstations.

batch queue a data structure maintained by job scheduler, containing jobs to run; LXBATCH offers different queues with different priorities, depending on the job CPU time requirements.

CASTOR CERN Advanced STORage manager.

CERN the European Organization for Nuclear Research.

CERN Advanced STORage manager a hierarchical storage management system developed at CERN for physics data files.

CMS Compact Muon Solenoid.

CMS SoftWare the overall collection of software built around a Framework, an Event Data Model, and Services needed by the simulation, calibration and alignment, and reconstruction modules that process event data so that physicists can perform analysis.

CMSSW CMS SoftWare.

CMSSW configuration a Python module file (.py) serving as a main input for CMSSW; it has to create a process object, describing the demanded computation chain and its properties.

EOS Exploration Of Storage.

European Organization for Nuclear Research an international research organization based in Geneva (Switzerland), mostly focused on high-energy particle physics.

Exploration Of Storage CERN project providing a fast and reliable xroot-managed disk pool for analysis-style data access.

HPC High Performance Computing.

IBM Platform LSF a powerful workload management platform for demanding, distributed HPC environments.

Jinja2 a modern and designer friendly templating language for Python, modelled after Django's templates.

Large Hadron Collider the world's largest and highest-energy particle accelerator, operating in in a circular tunnel 100 metres beneath the Swiss/French border at Geneva.

LHC Large Hadron Collider.

LSF Load Sharing Facility.

LXBATCH the CERN batch computing service consisting of around 35,000 CPU cores running Platform LSF®, providing computing power for tasks such as physics event reconstruction, data analysis and physics simulations.

LXPLUS a cluster consisting of public machines, providing the interactive logon service to Linux for all CERN users.

Monte-Carlo simulation a simulation using the Monte-Carlo random number generator.

NTuple easy to access data storage format of basic data types, allowing variables manipulation, plotting and fitting.

offline software the software used to process static data (both from simulations and detectors).

PLUS Public Login User Service.

PyTOTEM a python package developed for needs of calculation of integrated luminosity for data collected by TOTEM, containing several useful, general-purpose modules.

ROOT a system developed at CERN, providing a set of object-oriented frameworks with all the functionality needed to handle and analyze large amounts of data in a very efficient way.

Scientific Linux a Linux release put together by Fermilab and CERN, its primary purpose is to reduce duplicated effort of the labs, and to have a common install base for the various experimenters.

SL Scientific Linux.

splitting the process of creating CMSSW configurations for multiple jobs and their submission scripts from single templates.

submission script a bash script file (.sh) serving as a main LXBATCH input, describing sequence of steps for a single LSF job.

TOTal Elastic and diffractive cross section Measurement an experiment dedicated to the precise measurement of the proton-proton interaction cross section, as well as to the in-depth study of the proton structure.

TOTCSI TOTEM Configuration Splitter Improved.

TOTCSI configuration a Python module file (.py) serving as a main input for TOTCSI tool, describing desired splitting course and its details.

TOTEM TOTal Elastic and diffractive cross section Measurement.

XRootD an architecture, a communication protocol, and a set of plugins and tools aimed at giving high performance, scalable fault tolerant access to data repositories of many kinds, especially file-based ones.