Preprint typeset in JINST style - HYPER VERSION

MAUS: The MICE Analysis User Software

MAUS Developers

E-mail: durga@fnal.gov

ABSTRACT: The Muon Ionization Cooling Experiment (MICE) has developed the MICE Analysis User Software (MAUS) to simulate and analyse experimental data. It serves as the primary codebase for the experiment, providing for offline batch simulation and reconstruction as well as online data quality checks . The software provides both traditional particle physics functionalities such as track reconstruction and particle identification, and accelerator physics functions such as calculating transfer matrices and emittances. The code design is object orientated, but has a top-level structure based on the Map-Reduce model. This allows for parallelization to support fast live data reconstruction during data-taking operations. MAUS allows users to develop in either Python or C++ and provides APIs for both. Various software engineering practices from industry are also used to ensure correct and maintainable physics code, which include style, unit and integration tests, continuous integration and load testing, code reviews, and distributed version control. The software framework and the simulation and reconstruction capabilities are described.

KEYWORDS: MICE; Ionization Cooling; Software.

Contents

1. Introduction

1.1 The MICE Experiment

The Muon Ionization Cooling Experiment (MICE) sited at the Rutherford Appleton Laboratory (RAL) will be the first demonstration of muon ionization cooling – the reduction of the phase-space of muon beams. Muon beam cooling is essential for future facilities based on muon acceleration, such as the Neutrino Factory or Muon Collider [1, 2]. The experiment is designed to be built and operated in a staged manner. In the first stage (Step I), the muon beamline was commissioned [3] and characterized [4]. The present step – Step IV – will study the change in normalized transverse emittance using lithium hydride and liquid hydrogen absorbers under various optical configurations.

The MICE muon beamline is described in detail in [3]. There are 5 different detectors systems present on the beamline : 1) time-of-flight (TOF) scintillators, 2) threshold Cherenkov (Ckov) counters, 3) scintillating fiber trackers, 4) a KLOE-Lite (KL) sampling calorimeter, and 5) an electron-muon ranger (EMR). The TOF detector system consists of the three detector stations, TOF0, TOF1 and TOF2, each comprised of two orthogonal layers of scintillator bars, used to determine particle identification (PID) via the time-of-flight between the stations. Each station also provides a low resolution image of the beam profile. The Ckov system consists of two aerogel threshold Cherenkov stations, CkovA and CkovB, and privided supplementary PID, in particular for pion - muon separation. The KL and EMR detectors, the former using scintillating fibres, the latter scintillating bars, form the downstream calorimeter system, providing a final PID measurement.

The tracker system consists of two scintillating fibre detectors, one upstream of the MICE cooling channel, the other downstream, in order to measure the change in beam emittance across the cooling channel. Each detector consists of 5 stations, each station in turn having 3 fibre planes, allowing precision measurement of momentum and position on a particle-by-particle basis.

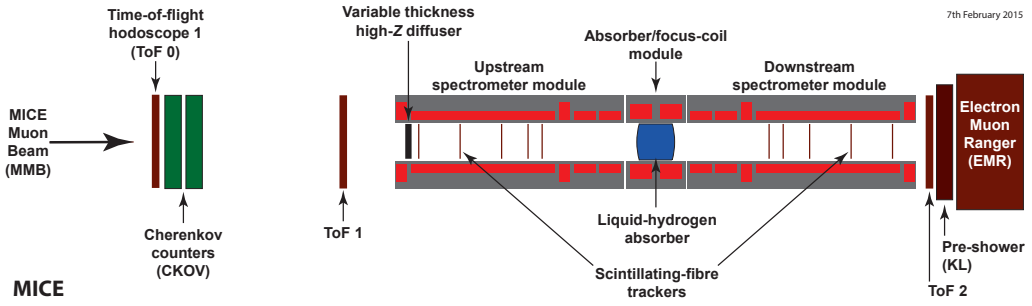

Figure 1. Rendering of the next MICE Step IV configuration

1.2 Software Requirements

The MICE software must serve both the accelerator physics and the particle physics needs of the experiment. Traditional particle physics functionality includes reconstructing tracks, identifying particles, and simulating the response from various detectors, while the accelerator physics aspect includes computing transfer matrices and Twiss parameters and propagating beam envelopes. All

of these require knowledge of the beamline, geometries of the detectors, knowledge of the magnetic fields, and functionality to reconstruct or simulate the various detectors, thus necessitating a single software scope.

Given the complexity and the inherent time-scales of experiments, the need also arises to ensure long-term correctness and maintainability of the software used for the experiment. Good performance is also important in order to ensure that the software can reconstruct data with sufficient speed to support live online monitoring of the experiment.

2. MAUS

The MICE Analysis User Software (MAUS) [5] is the experiment's simulation, reconstruction, and analysis software framework and aims to provide capabilities to 1) perform a Monte Carlo (MC) simulation of the experiment, 2) reconstruct tracks and identify particles from simulations and real data, and 3) provide monitoring and diagnostics while running the experiment.

Installation is by a set of shell scripts with SCons [6] as the build tool. The codebase is maintained with the GNU Bazaar revision control system and is hosted on Launchpad. MAUS has a number of dependencies on standard packages such as Python, ROOT [7] and GEANT4 [8] which are built as "third party" external libraries during the installation process. The officially supported platform is Scientific Linux 6 though developers successfully build on Ubuntu, CentOS and OpenSUSE Linux distributions.

Each of the MICE detector systems, described in Section 1.1, are represented within MAUS. Their simulation and reconstruction algorithms are described in Section 4 and the supporting datastructures in Section 2.2. MAUS also provides global reconstruction routines, which combine data from each detector system to provide combined PID hypotheses and a global track fit, again described in Section 4.

2.1 Code Design

MAUS is written in a mixture of Python and C++. C++ is used for complex or low level algorithms where processing time is important while Python is used for simple or high level algorithms where development time is a more stringent requirement. Developers are allowed to write in either Python or C++ and python bindings to C++ are handled through internal abstractions or SWIG [9]. In practice, all the reconstruction modules are written in C++ but support is provided for legacy modules written in Python.

MAUS has an Application Programming Interface (API) that provides a framework on which developers can hang individual routines. The MAUS API provides MAUS developers with a well-defined environment for developing reconstruction code, while allowing independent development of the backend and code-sharing of common elements like error handling and data mangling.

The MAUS data processing model is inspired by the Map-Reduce pattern [10], which forms the core of the API design. Map-Reduce is a useful model for parallelizing data processing on a large scale, including for particle physics applications. For MAUS, the API was simplified to use *transformers* in place of maps, though these modules have retained the name *map*. A map process takes a single object as an input, which remains unaltered, and returns a new object as the output,

whereas a transformer process alters the input object in place (in the case of MAUS this object is the *spill* class, see Section 2.2).

A *Module* is the basic building block of the MAUS API framework. Four types of module exist within MAUS:

1. **Inputters** generate input data either by reading data from files or sockets, or by generating an input beam.

2. **Mappers** modify the input data, for example by reconstructing signals from detectors, or tracking particles to generate MC hits

3. **Reducers** collate the mapped data and allow functionality that requires access to the entire data set

4. **Outputters** save the data either by streaming over a socket or writing data to disk.

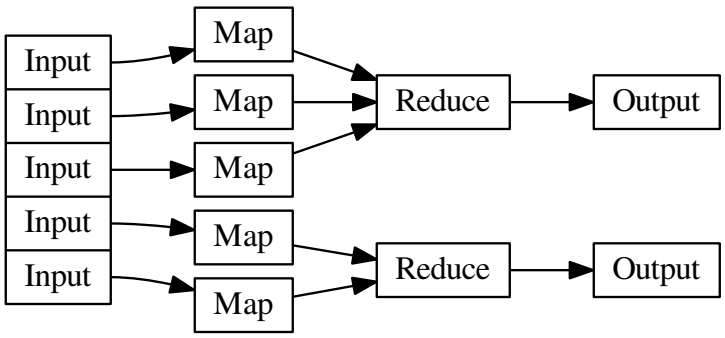

Figure 2. A Map-Reduce framework.

Each module type follows a common, extensible, object-orientated class heirarchy, shown for the case of the map and reduce modules in Figure 3.

There are some objects that sit outside the scope of this modular framework but are nevertheless required by several of the modules. For instance, knowledge of the detector geometries, magnetic fields, and calibrations are required by the reconsruction and simulation modules, and objects like electronics cabling maps are required to unpack data from the data acquisition (DAQ) source, and error handling functionality is required by all of the modules. All these objects are accessed through a static singleton *globals* class.

MAUS has two execution concepts. A *job* refers to a single execution of the code, while a *run* refers to the processing of data for a DAQ run or MC run. A job may contain many runs. Since data are typically accessed from a single source and written to a single destination, Inputters and Outputters are initialized and destroyed at the beginning and end of a job. On the other hand, Mappers and Reducers are initialized at the beginning of a run in order to allow loading run-specific information such as electronic cabling maps, fields, and calibrations.

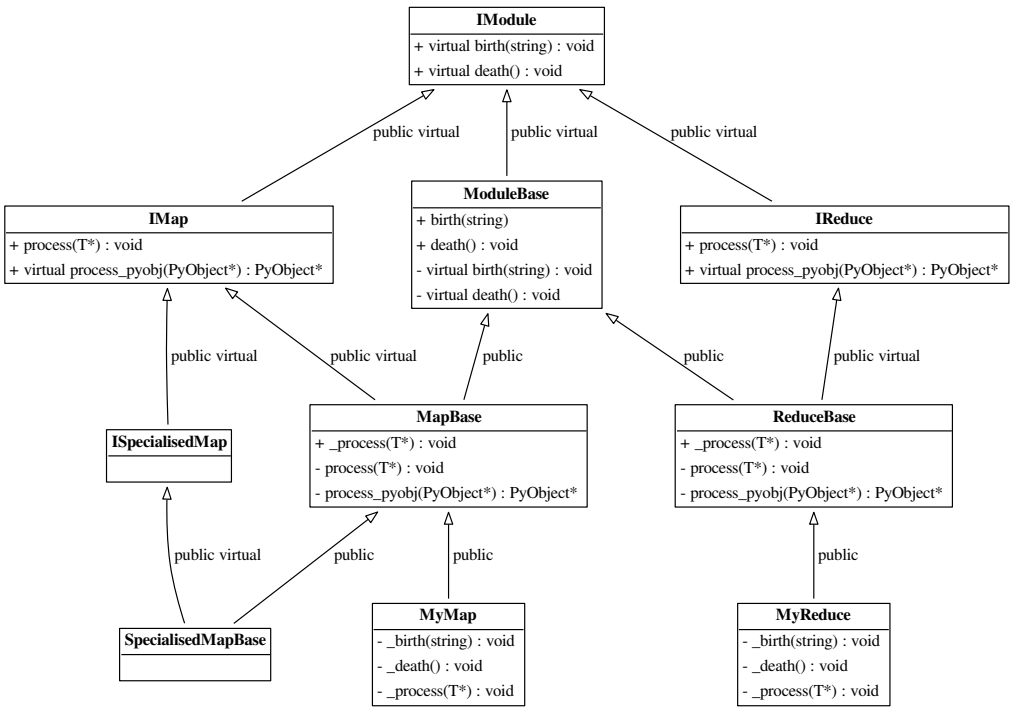

Figure 3. The MAUS API class hierarchy for Map and Reduce modules. The input and output modules follow a related design. *T* represents a templated argument. "+" indicates the introduction of a virtual void method, defining an interface, while "-" indicates a class implements that method, fulfilling that aspect of the interface. The functions *process_pyobj* are the main entry points for Python applications, *process* the entry points for C++ applications. The framework can be extended as many times as is neccessary, as exmplified by the "SpecialisedMap" classes.

The principal data type in MAUS, which is passed from module to module, is the *spill*. A single spill corresponds to data from the particle burst associated with a dip of the MICE target[3]. A spill typically lasts 3 *ms* and contains several DAQ triggers. Data from a given trigger defines a single MICE *event*. In the language of the Input-Map-Reduce-Output framework above, an Input module creates an instance of spill data, a Map module processes the spill (reconstructing, simulating, etc), a Reduce module acts on a collection of spills when all the mappers finish, and finally an Output module records the data to a given file format.

Modules can exchange spill data either as C++ pointer or JSON [11] object types. In Python, the data's format can be changed by using a converter module and in C++, mappers are templated to a MAUS data type and the API then handles any necessary conversion to that type (see Figure 3). During production deployment of the software it was found that there was a large reduction in processing speed when JSON was used for passing data internally, due to the inherent slowness in (de)serializing each spill to JSON format. Hence it was decided that modules for official reconstruction and simulations must exchange C++ objects by default, in order to minimize need for

conversion between data types. However, data can still be output in JSON format and developers find it extremely useful during debugging.

Data contained within the MAUS data structure (see Section 2.2) can be saved to permanent storage in one of two formats. The default data format is a ROOT binary and the secondary format is JSON. ROOT is a standard high energy physics analysis package, distributed with MAUS, through which many of the analyses on MICE are conducted. Each spill is stored as a single entry in a ROOT TTree object [7]. JSON is an ASCII data-tree format readable with any text editor. Specific JSON parsers are available - for example, the Python *json* module is available and comes prepackaged with MAUS.

Besides saving the full data structure, MAUS is also capable outputting the reduced data produced by Reducer modules. These data are not part of the spill, but have their own special data class, known as *Image*. Image is used by Reducers to store images of monitoring histograms, efficiency plots, etc. Image data may only be saved in JSON format.

2.2 Data Structure

2.2.1 Physics Data

At the top of the MAUS data structure is the spill class which contains all the data from the simulation, raw real data and the reconstruction. As mentioned earlier, the spill is passed between modules and written to permanent storage. The data within a spill is organised into arrays of three possible event types: a *MCEvent* contains data which represents the simulation of a single particle traversing the experiment and the simulated detector responses; a *DAQEvent* corresponds to the real detector readout data for a single trigger; and a *ReconEvent* corresponds to the data reconstructed for a single particle event (either arising from a MC particle or a real data trigger). These different branches of the MAUS data struture are shown diagrammatically in Figures 4 - 9.

The sub-structure of the the MC event class is shown in Figure 5. The class is subdivided into sensitive detector hits (energy deposited, position, momentum) for each of the MICE detectors (see Section 1.1). The event also contains information about the primary particle that created the hits in the detectors.

The sub-structure of the the reconstruction event class is shown in Figure 6. The class is again subdivided into events representing each of the MICE detectors, together with the data from the trigger, and data for the global event reconstruction. Each detector and the global reconstruction class has several further layers of reconstruction data, shown in Figures 7 - 9.

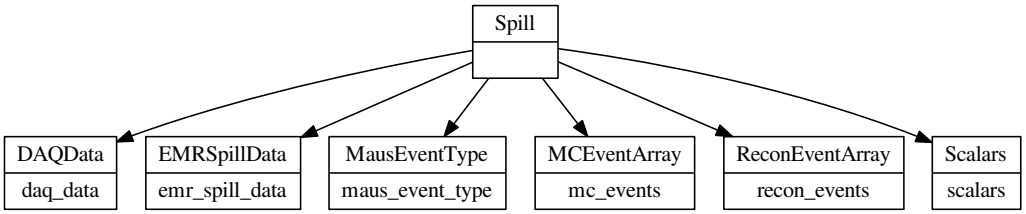

Figure 4. The MAUS output structure for a spill event. The top label in each box is the name of the C++ class and the bottom label is the json branch name. If a [] is shown, this indicates that child objects are array items.

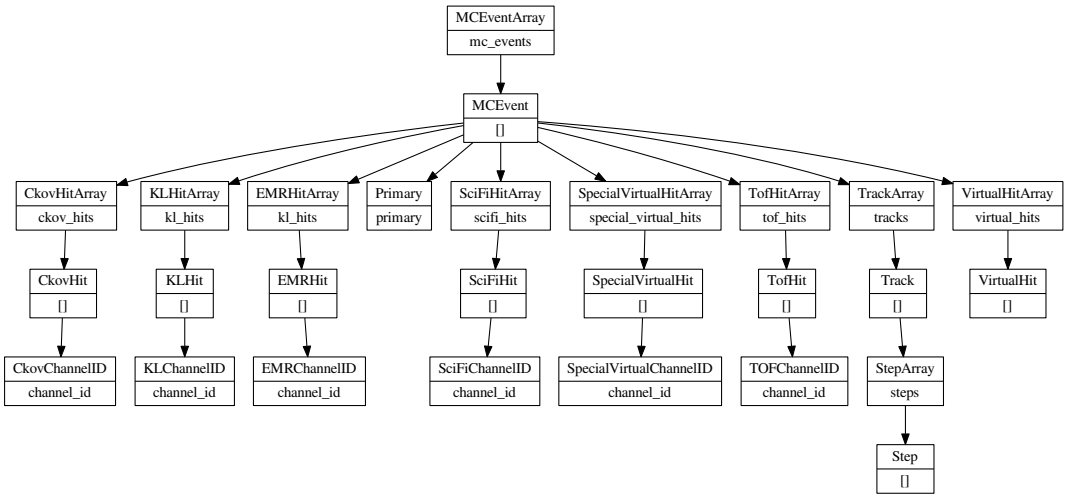

Figure 5. The MAUS data structure for MC events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. If a [] is shown, this indicates that child objects are array items.

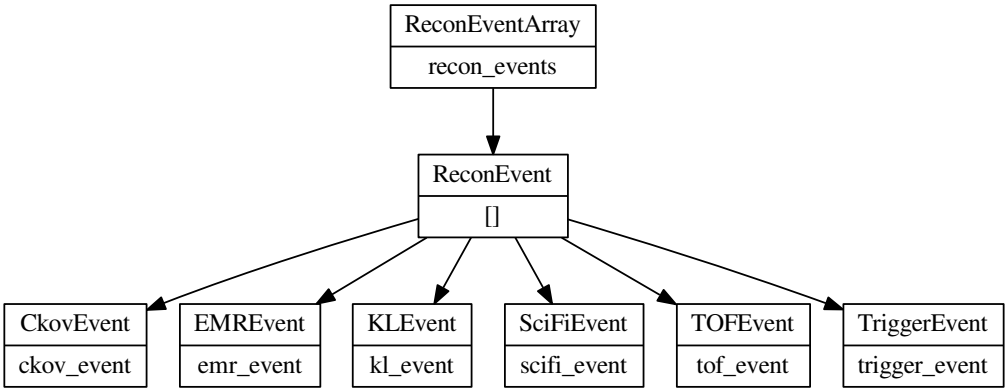

Figure 6. The MAUS data structure for reconstruction events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. If a [] is shown, this indicates that child objects are array items.

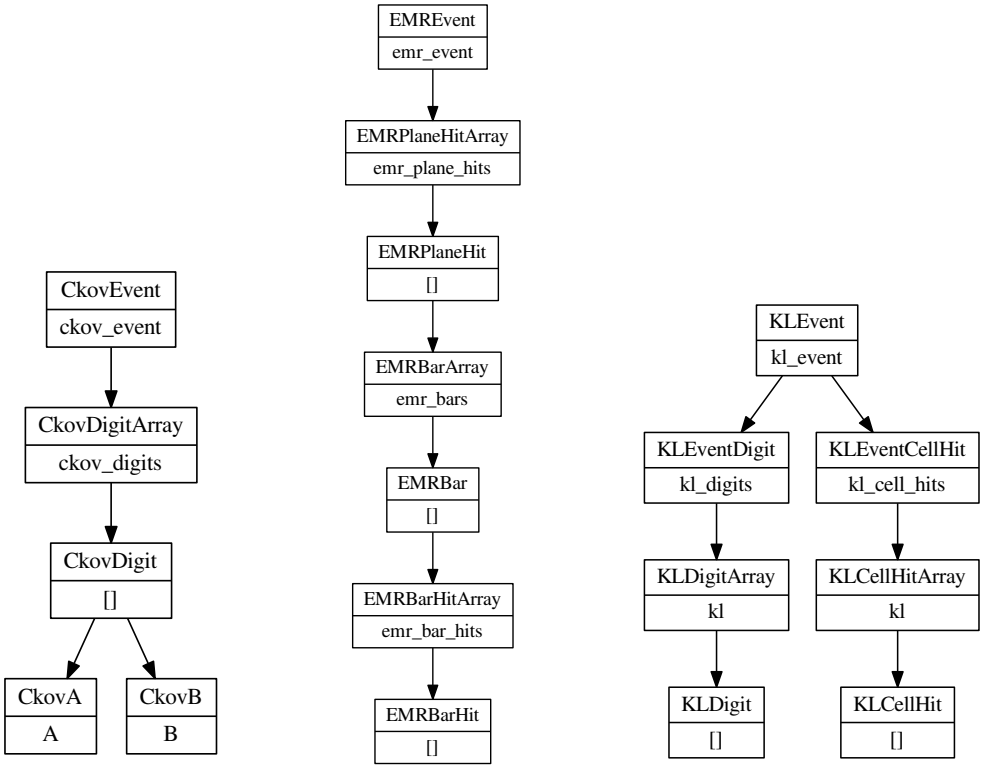

Figure 7. The MAUS data structure for CKOV (left), EMR (middle) and KL (right) reconstruction events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. If a [] is shown, this indicates that child objects are array items.

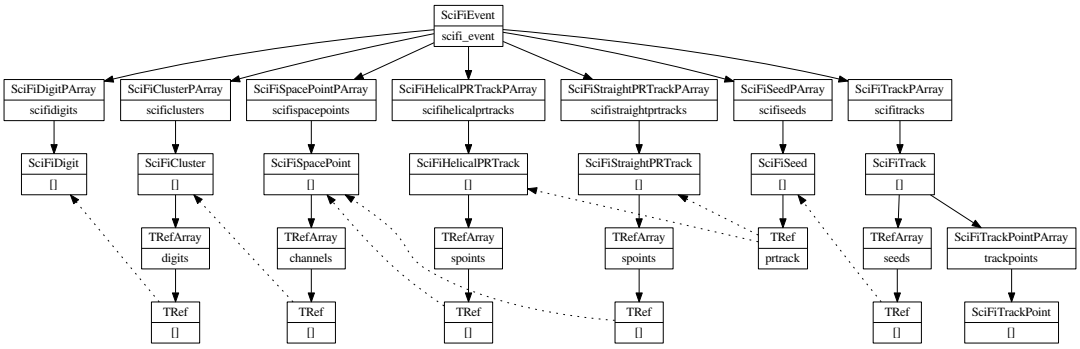

Figure 8. The MAUS data structure for the tracker. The top label in each box is the name of the C++ class and the bottom label is the json branch name. If a [] is shown, this indicates that child objects are array items.

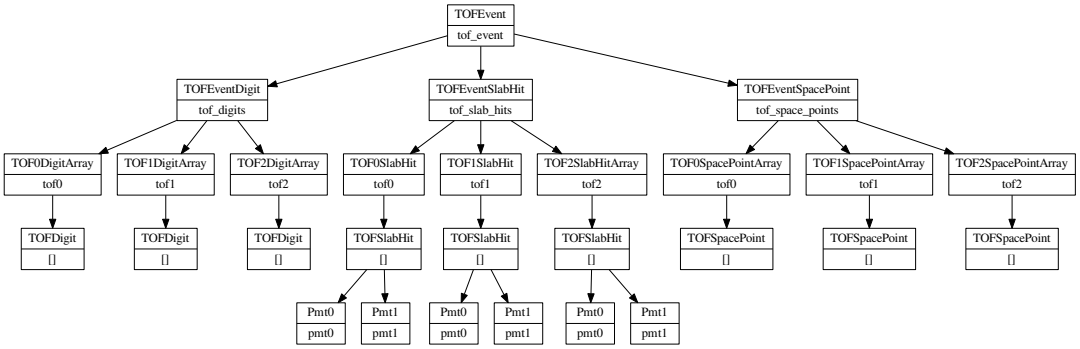

Figure 9. The MAUS data structure for the TOFs. The top label in each box is the name of the C++ class and the bottom label is the json branch name. If a [] is shown, this indicates that child objects are array items.

2.2.2 Top Level Data Organisation

In addition to the spill data, MAUS also contains structures for storing supplementary information for each run and job. These are referred to as *JobHeader* and *JobFooter*, and *RunHeader* and *RunFooter*. The former represents data from the start and end of a job, such as the MAUS release version used to create it, and the latter data from the start and end of a run, such as the geometry ID used for the data processing. This may be saved to permanent storage along the spill.

In order to interface with ROOT, particularly in order to save data in the ROOT format, thin wrappers for each of the top level classes, and a templated base class, were introduced. The allows the ROOT TTree, in which the output data is stored (see Section 2.2.1), to be given a single memory address to read from. The wrapper for Spill is called *Data*, while for each of RunHeader, RunFooter, JobHeader and JobFooter, the respective wrapper class is just given the original class name with "Data" appended e.g. *RunHeaderData*. The base class for each of the wrappers is called *MAUSEvent*. The class hierarchy is illustrated in Figure 10.

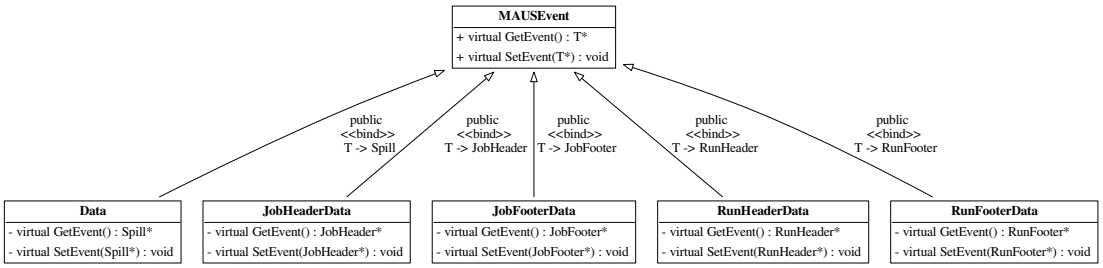

Figure 10. Class hierarchy for the wrappers and base class of the top-level classes of the MAUS data structure.

2.3 Data Flow

The MAUS data flow, showing the reconstruction chain for data originating from MC or real data, is shown in Fig. 11. The data flow is grouped into three principal areas: the simulation data flow is used to generate digits (electronics signals) from particle tracking; the real data flow is used to generate digits from real detector data; and the reconstruction data flow illustrates how digits are built into higher level objects and converted to physics parameters of interest. The reconstruction data flow proceeds identically whether the digits originate from simulation or real data.

2.4 Testing

MAUS has a set of tests at the unit level and integration level, together with code style tests for both Python and C++. Unit tests are implemented against a single function, while integration tests operate against a complete workflow. Unit tests check that each function operates as intended by the developer and can achieve a high level of code coverage and good test complexity. Integration tests allow checking the overall performance of the code meets the specifications laid out.

The MAUS team aims to provide unit test coverage that executes 70–80 % of the total code base. This level of test coverage typically results in a code that performs the major workflows with-

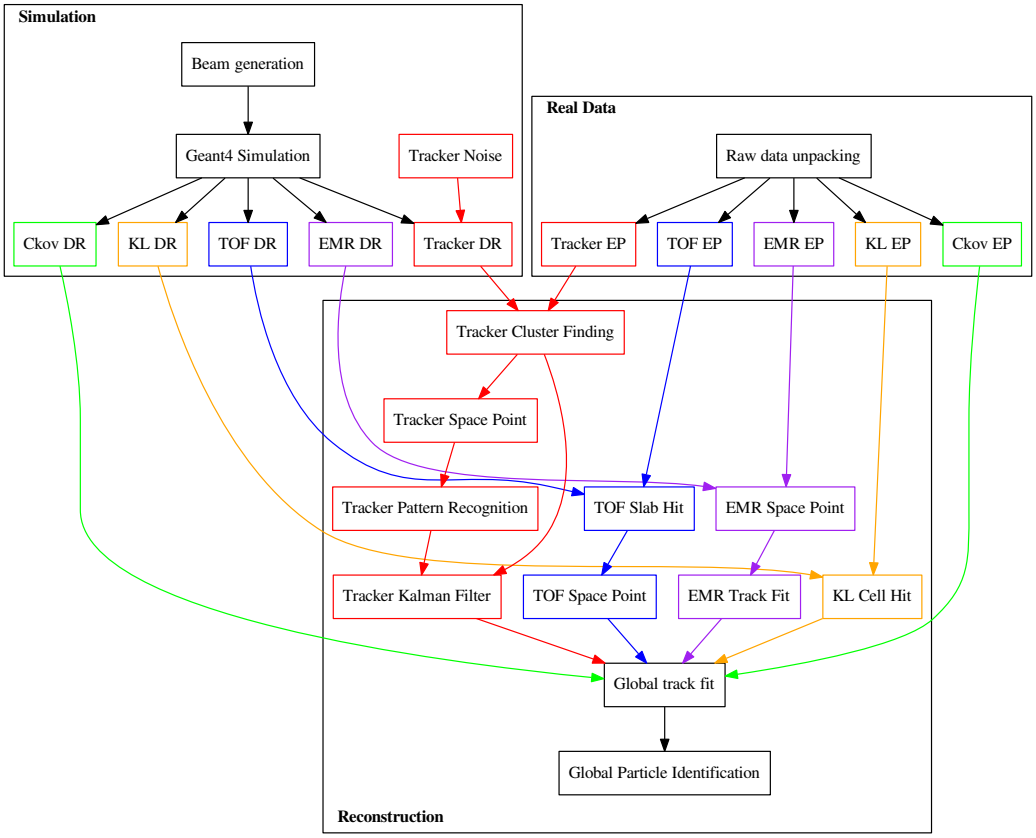

Figure 11. Data flow for the MAUS project. The data flow is color-coded by detector: Ckov - green, EMR - purple, KL - orange, TOF - blue, Tracker - red.

out any problem, but has errors in some of the less well-used functions and can behave ungracefully following user error. At the most recent release, MAUS test coverage was 68 % for python code, 78 % for non-legacy C++ code and 36 % for legacy (pre-2011) C++ code.

MAUS operates a continuous integration stack using a suite of test servers. Code building and testing is driven by the Jenkins test environment [12]. Developers are asked to perform a build and test on a personal code branch, using the test server, before requesting a merge with the development trunk. This enables MAUS to make frequent clean releases. Typically MAUS works on a 4 - 8 week major release cycle.

3. Monte Carlo

An MC simulation of MICE encompasses beam generation, geometrical description of detectors and fields, tracking of particles through detectors, and digitization of the detectors' response to particle interactions.

3.1 Beam generation

Several options are provided to generate an incident beam. 1) Routines are provided to sample particles from a multivariate gaussian distribution or generate ensembles of identical particles ("pencil" beams). Additionally it is possible to produce time distributions that are either rectangular or triangular in time to give a simplistic representation of the MICE time distribution. Parameters, controlled by datacards, are available to control random seed generation, relative weighting of particle species, and the transverse-longitudinal coupling in the beam. MAUS also allows the generation of a polarized beam by generating a spin vector from beam distributions.

Beam particles can also be read in from an external file G4Beamline [13], ICOOL [14], and user-defined formats. In order to generate beams which are more realistic taking into account the geometry and fields of the actual MICE beamline, we use G4Beamline to model the MICE beam line from the target to a point upstream of the second quad triplet (upstream of Q4). The beam line settings *e.g.* magnet field strengths and number of particles to generate, are controlled through datacards. The magnet field strengths have been tuned to produce beams that are reasonably accurate descriptions of the real beam. Scripts to install G4Beamline are shipped with MAUS.

Once the beam is generated, the tracking and interactions of particles as they traverse the rest of the beamline and the MICE detectors is performed using GEANT4

3.2 GEANT4

The GEANT4 simulation within MAUS starts $1m$ downstream of the second beamline dipole (D2) magnet. GEANT4 bindings are encoded in the Simulation module. GEANT4 groups particles by run, event and track. A GEANT4 run maps to a MICE spill; a GEANT4 event maps to a single inbound particle from the beamline; and a GEANT4 track corresponds to a single particle in the experiment.

GEANT provides a variety of reference physics lists to model the interactions of particles with matter. The default list in MAUS is *QGSP_BERT* which models hadron interactions via Bertini cascade model up to 10 GeV/c, a low energy parametrized model between beyond 10 and a quark gluon string model beyond 25 GeV/c. MAUS provides methods to set up the GEANT4 physical processes allows the user to control processes with datacard settings. Routines are also provided to the interface the internal geometry representation in MAUS with GEANT4 descriptions. Finally, MAUS provides routines to extract particle data from the GEANT tracks independent of the GEANT geometry with virtual planes.

3.3 Geometry

MAUS uses an on-line Configuration Database to store all of its geometries. These geometries have been extracted from CAD drawings which are continually updated based on the latest surveys and technical drawings available. The CAD drawings are translated to a geometry specific subset of XML, the Geometry Description Markup Language (GDML) prior to being recorded in the configuration database through the use of the Fast-RAD commercial software package.

The GDML formatted description contains the beam-line elements and the positions of the detector survey points. Beam-line elements are described using Tessellated solids to define the shapes of the physical volumes. The detectors themselves are described using an independently generated

set of GDML files using GEANT4 standard volumes. An additional XML file is appended to the geometry description that assigns magnetic fields and associates the detectors to their locations in the GDML files. This file is initially written by the geometry maintainers and formatted to contain run specific information during download.

The GDML format has a number of benefits. The files can be read via a number of already existing libraries in GEANT4 and ROOT for the purpose of independent verification and validation. Because it is a subset of XML, the data contained in the GDML files are readily accessible through the application of the "libxml2" python extension. The GDML are in turn translated into the MAUS readable geometry files either by directly accessing the data using the python extension (which is the method applied to the detector objects) or through the use of EXtensible Stylesheet Language Transformations (XSLT).

3.4 Tracking, Field Maps and Beam Optics

MAUS tracking is performed by GEANT4 [8]. By default, MAUS uses 4^{th} order Runge-Kutta for tracking, although other routines are available. 4th order Runge-Kutta has been shown to have very good precision relative to the MICE detector resolutions, even for step sizes of several cm.

Magnetic field maps are implemented as a series of overlapping regions, each of which contains a field. On each tracking step, MAUS iterates over the list of fields, transforms to the local coordinate system of the field map, and calculates the field. The field values are transformed back into the global coordinate system, summed and passed to GEANT4.

Numerous field types have been implemented within the MAUS framework. Solenoid fields can be calculated numerically from cylindrically symmetric 2D field maps, by taking derivatives of an on-axis solenoidal field or by using the sum of fields from a set of cylindrical current sheets. Pillbox fields can be calculated by using the Bessel functions appropriate for a TM010 cavity or by reading a cylindrically symmetric field map. Multipole fields can be calculated from a 3D field map, or by taking derivatives from the usual multipole expansion formulae. Linear, quadratic and cubic interpolation routines have been implemented for field maps.

Matrix transport routines for propagating particles and beams through these field maps have been implemented. Transport matrices are calculated by taking the numerical derivative of tracking output. These can be used to transport beam ellipses and single particles, for example enabling optics work, beam matching and so forth. Higher order transport routines are also available.

The accelerator modelling routines in MAUS have been validated against ICOOL and G4Beamline. The routines have been used to model a number of beamlines and rings, including the Neutrino Factory front end.

3.5 Detector response and digitization

The modelling of the detector response and electronics enables MAUS to provide test data for reconstruction algorithms and estimate the errors introduced by the detector and its readout.

The interaction of particles in material is also performed by GEANT4. A "sensitive detector" class for each detector processes GEANT4 hits in active detector volumes and stores hit information such as the volume that was hit, the energy deposited and the time of the hit. Each detector's digitization routine then simulates the electronics response to these hits, modelling processes such

as the photon yield of scintillator light, attenuation in light guides and the pulse shape in the electronics. The data structure of the outputs from the digitizers are designed to mock the output from the unpacking of the data from the DAQ.

4. Reconstruction

The reconstruction chain takes as its input either digitized hits from MC or DAQ digits from real data. Regardless, the detector reconstruction algorithms, by requirement and design, operate the same way on both MC and real data.

??? TODO: something about aims of the reconstruction – tracking, particle identification for purposes of physics goals [mention physics goals somewhere up top ?] ??? TODO: some intro/description of detectors with refs to pubs ??

4.1 Time of flight

There are three time-of-flight detectors in MICE and they serve to distinguish particles based on their times of flight. The detectors are made of plastic scintillator and in each station there are x and y planes with 6 or 10 slabs in each plane.

Each GEANT hit in the TOF is associated with a physical scintillator slab based on the geometry. The energy deposited by a hit in is first converted to units of photoelectrons. The photoelectron yield from a hit is attenuated by the distance from the hit to the PMT, then smeared by the photoelectron resolution. The yields from all hits in a given slab are then added and the summed photoelectron yield is converted to ADC counts.

The hit time is propagated to the PMTs at either end of the slab. The propogated time is then smeared by the PMT time resolution and converted to TDC counts. After converting the energy deposit to ADC and the time to TDC, the TDC values are "uncalibrated" so that at the reconstruction stage they can be corrected just as is done with real data.

The reconstruction proceeds in two main steps: 1) Slab-hit-reconstruction takes individual PMT digits and associates them to reconstruct the hit in the slab. If there are multiple hits associated with a PMT, the hit which is earliest in time is taken to be the real hit. Then, if both PMTs on a slab have hits, the slab is considered to have a valid hit. The TDC values are converted to time and the hit time and charge associated with the slab hit are taken to be the average of the two PMT times and charges respectively. In addition, the charge product of the PMT charges is also formed. 2) Finally, individual slab hits are used to form space-points. A space point in the TOF is a combination of x and y slab hits. All combinations of x and y slab hits in a given station are treated as space point candidates. Calibration corrections, stored in the Configurations Database, are applied to these hit times and if the reconstructed space-point is consistent with the resolution of the detector, the combination is said to be a valid space point.

The TOF has been shown to provide good time resolutions, with errors at the 50 ps level. Improvements are being made to the calibration algorithms to improve the reconstructed resolutions and examine some suspected systematic effects.

??? TODO: Plots of resolution, time of flight?? [do performance plots belong in this paper??]

4.2 Scintillating fiber trackers

The scintillating fiber trackers are the central piece of the reconstruction. As mentioned in Section 1.1, there are two trackers, one upsteam and the other downstream of an absorber, situated within solenoidal magnetic fields. The trackers are responsible for measuring the emittance before and after particles pass through the absorber.

The tracker software algorithms and performance are described in detail in [15]. Digits are the most basic unit fed into the main reconstruction module, each digit representing a signal from one tracker channel. Digits from adjacent channels are assumed to come from the same particle and are grouped to form clusters. Clusters from channels which intersect each other, in at least two planes from the same station, are used to form space-points, giving x and y positions where a particle intersected a station. Once space-points have been found, they are associated with individual tracks through pattern recognition (PR) (described in section 4), giving straight or helical PR tracks. These tracks, and the space-points associated with them, are then sent to the final track fit. To avoid biases that may come from space-point reconstruction, the Kalman filter uses only reconstructed clusters as input.

??? TODO: Plots of spatial, momentum resolutions?? [do performance plots belong in this paper??]

4.3 KL calorimeter

Hit-level reconstruction of the KL is implemented in MAUS. Individual PMT hits are unpacked from the DAQ or simulated from MC and the reconstruction associates them to identify the slabs that were hit and calculates the charge and charge-product corresponding to each slab hit. [ACTION: Update - MB/JN]

??? TODO: What plots?? [do performance plots belong in this paper??]

4.4 Electron-muon ranger

Hit-level reconstruction of the EMR is now implemented in MAUS; the integrated ADC and time over threshold are calculated for each bar that was hit. The EMR reconstructs a wide range of variables that can be used for particle identification, as well as momentum reconstruction. The software and performance of the detector are described in detail in [16].

??? TODO: Ref to EMR paper ??? TODO: Plots??

4.5 Cherenkov

The Ckov reconstruction takes the raw flash-ADC data, subtracts pedestals, calculates the charge and applies calibrations to determine the photoelectron yield. [ACTION: Update - DR/LC]

??? TODO: expand ??? TODO: plots?? [do performance plots belong in this paper??]

4.6 Global reconstruction

The aim of the Global Reconstruction is to take the reconstructed outputs from individual detectors and tie them together to form a global track including a likelihood for various particle hypotheses.

Global track matching is performed by collating particle hits (TOFs 0, 1 and 2, KL and Ckov) and tracks (Trackers and EMR) from each detector using their individual reconstruction and combining them using a 4th order Runge-Kutta (RK4) method to propagate the particles' between these detectors.

Particle identification in MICE typically requires the combination of several detectors. Principally the time-of-flight between TOF detectors can be used to calculate velocity, which is compared with the momentum measured in the trackers to calculate particle mass and hence particle type. For all but very low pT events, charge can be determined from the direction of helical motion in the Trackers. Additional information can be gleaned from the Ckov, KL and EMR detectors. The global particle identification framework is designed to tie this disparate information into a set of hypotheses of particle types, with an estimate of the likelihood of each hypothesis.

Global Reconstruction is performed outwards from the cooling channel; upstream of Tracker 1 (tracker_0 in MAUS) through TOF0; and downstream of Tracker 2 (tracker_1 in MAUS) through EMR. It is also available as a commissioning tool providing through-going tracks from TOF1 to EMR, in the absence of magnetic fields. During later Step IV analysis, time of flight between TOF1 and TOF2 will be included to improve the performance of the global reconstruction.

Efficiency and purity are returned for global track matching and PID along with global reconstruction efficiency (combined efficiency of the global reconstruction not including individual detector recon efficiencies) and global efficiency (combined efficiency of the global reconstruction,including the result of detector recon efficiencies) values.

4.6.1 Global Track Matching

Track points are matched to form tracks using a RK4 method. Initially this is done independently for the upstream and downstream (i.e. either side of the absorber) sections of the beamline. As the Trackers provide the most accurate position reconstruction, they are used as starting points for track matching, propagating hits outwards into the other detectors and then comparing the propagated position to the measured hit in the respective detector. The acceptance criteria for a hit belonging to a track is an agreement within the fundamental limit (i.e. half the detector granularity) plus an allowance for multiple scattering. Track matching is currently performed for all TOFs, KL and EMR.

The RK4 propagation requires the mass and charge of the particle to be known, therefore, it is necessary to perform track matching for all realistically possible particle types (muons, pions, and electrons), though typically the Tracker reconstruction will provide a charge hypothesis to exclude a charge-sign beforehand. Tracks for all possible PID hypotheses are then passed to the PID algorithms.

Efficiency and purity of the track matching are evaluated on a per-detector basis by comparing MC tracks that could have been matched—which requires MC hits existing in both the respective tracker (upstream or downstream) and the detector for which the efficiency is being calculated. In order for an event to be used in this evaluation, the tracker hits have to have been reconstructed correctly or in such a way so as to not alter the path of the particle in the tracker, as otherwise track matching can't reliably be performed. Checks are then performed to determine whether the correct hit has been matched.

For use during the alignment run, track matching between upstream and downstream tracks in a no-absorber no-field scenario has been implemented using a cut on the TOF1-TOF2 time-of-flight as the principal matching criteria ($z/c < dt < z/ \sim 0.6c$).

4.6.2 Global PID

The Global PID in MAUS uses a log-likelihood method to determine the particle identification (PID) of a global track. It is based upon a framework of PID variables; quantities that can be used to distinguish between different particle species. These variables are used with Monte Carlo tracks to produce probability density functions (PDFs) of their values. PID variables for a real data track are then compared to the corresponding PDFs, in order to obtain a set of likelihoods for the potential PIDs of the track.

Track reconstruction will pass to the PID a number of potential tracks, each reconstructed for a given particle hypothesis. The PID will take each of these tracks in turn, and determine the most likely PID of that track. This process follows a series of steps:

1. Each track is copied into an intermediate track, and the PID of the copy is set to zero (undefined).

2. For each potential PID hypothesis x, the log-likelihood LL_x is calculated using the PID variables.

3. The track is assigned an object containing the log-likelihood for each PID hypothesis (for reference).

4. From the log-likelhoods, the confidence level for a track having a PID x (CL_x) is calculated by $CL_x = (LL_x)/(\sigma_i LL_i)$

5. The PID of the track is set to the particle hypothesis that gave the best CL.

If the PID determined above matches the PID that the track reconstruction had assigned to the track, then that track (with it's PID) is considered the correct track, to then be passed back to track reconstruction for final fitting. Otherwise, the PID is unable to assign a PID to the particle and does not return a final track.

PID will be used for both global alignment during commissioning, and running of Step IV. However, the different requirements of these phases of the experiment necessitate two different sets of PID variables. During Step IV, the requirement for independence between upstream and downstream track reconstruction means that the PID variables used are also separated into two sets, one using the upstream detectors, the other the downstream detectors.

4.7 Online reconstruction

During data taking, it is essential to visualize a detector's performance and have diagnostic tools to identify and debug unexpected behavior. This is accomplished through summary histograms of high and low-level reconstructions from detectors. These are available for the TOF, Cherenkov, KL, EMR and Trackers.

For online reconstruction, MAUS uses a distributed processing model to enable a scalable reconstruction of the MICE dataset. Raw data is passed to a networked message queue for multiprocessing across multiple CPUs and servers. Reconstructed data is handed to another message queue. Histogramming routines pick data from this second message queue and collate it into histograms, which are written to disk. A web-based visualisation tool enables viewing of the histograms. A production version of the online reconstruction is available. Further development work is underway on the user interface and some of the backend infrastructure. Though the framework for the online reconstruction is based on distributed processing of spills, the reconstruction modules are the same as those used for offline processing.

An event display summarising global reconstruction data will be implemented when the global reconstruction is complete. This will enable visualisation of the phase space distribution of the beam at various points along the beamline, together with a comparison of the nominal beam envelope propagation. The event display is intended to enable online validation of the behaviour of accelerator components, by comparing propagation of the beam envelope with the detected beam parameters.

??? TODO: Redo re new Yordan framework

5. Summary

??? TODO:

Acknowledgments

Acknowledgments.

References

[1] The IDS-NF collaboration. International design study for the neutrino factory: Interim design report, 2011. IDS-NF-020, www.ids-nf.org/wiki/FrontPage/Documentation.

[2] Steve Geer. Muon Colliders and Neutrino Factories. *Annual Review of Nuclear and Particle Science*, 59:345 – 367, 2009.

[3] M. Bogomilov et al. The MICE Muon Beam on ISIS and the beam-line instrumentation of the Muon Ionization Cooling Experiment. *JINST*, 7:P05009, 2012.

[4] D. Adams et al. Characterisation of the muon beams for the Muon Ionisation Cooling Experiment. *Eur. Phys. J.*, C73(10):2582, 2013.

[5] C. Tunnell and C Rogers. MAUS: MICE Analysis User Software. In *Proc. 2011 International Particle Accelerator Conference, San Sebastian*, 2011. MOPZ013.

[6] http://scons.org/.

[7] R. Brun and F. Rademakers. Root - an object oriented data analysis framework. *Nucl. Instrum. Meth.*, A389:81 – 86, 1997. http://root.cern.ch/.

[8] S Agnostinelli et al. Geant4 - a simulation toolkit. *Nucl. Instrum. Meth.*, A506:250 – 303, 2003.

[9] http://www.swig.org/.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of OSDI04*, 2004. http://research.google.com/archive/mapreduce.html.

[11] http://json.org.

[12] http://jenkins-ci.org.

[13] Thomas J. Roberts and Daniel M. Kaplan. G4BeamLine programme for matter dominated beamlines. In *Proc. 2007 Particle Accelerator Conference, Albuquerque*, 2007. THPAN103.

[14] R. C. Fernow. Icool: A simulation code for ionization cooling of muon beams. In *Proc. 1999 Particle Accelerator Conference, New York*, 1999.

[15] A. Dobbs, C. Hunt, K. Long, E. Santos, M.A. Uchida, P. Kyberd, C. Heidt, S. Blot, and E. Overton. The reconstruction software for the mice scintillating fibre trackers. *JINST*, 11(12):T12001, 2016.

[16] D. Adams et al. Electron-muon ranger: performance in the MICE muon beam. *JINST*, 10(12):P12012, 2015.