Preprint typeset in JINST style - HYPER VERSION

MAUS: The MICE Analysis User Software

MICE Collaboration

NOTE: Need to finalize author list. Include from list of 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 analyze 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 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 code, including 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; Reconstruction; Simulation.

Contents

1. Introduction

1.1 The MICE experiment

The Muon Ionization Cooling Experiment (MICE) sited at the STFC Rutherford Appleton Laboratory (RAL) will deliver 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 was designed to be built and operated in a staged manner. In the first stage, the muon beamline was commissioned [3] and characterized [4]. The present configuration shown in figure 1 will be used to study the factors that determine the performance of an ionization cooling channel and to observe for the first time the reduction in transverse emittance of a muon beam.

The MICE Muon Beam line is described in detail in [3]. There are 5 different detector systems present on the beamline: time-of-flight (TOF) scintillators [5], threshold Cherenkov (CKOV) counters [6], scintillating fiber trackers [7], a sampling calorimeter (KL) [4], and the Electron Muon Ranger (EMR) – a totally active scintillating calorimeter [8]. The TOF detector system consists of three detector stations, TOF0, TOF1 and TOF2, each composed of two orthogonal layers of scintillator bars. The TOF system is 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. The KL and EMR detectors, the former using scintillating fibers embedded in lead sheets, and the latter scintillating bars, form the downstream calorimeter system.

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

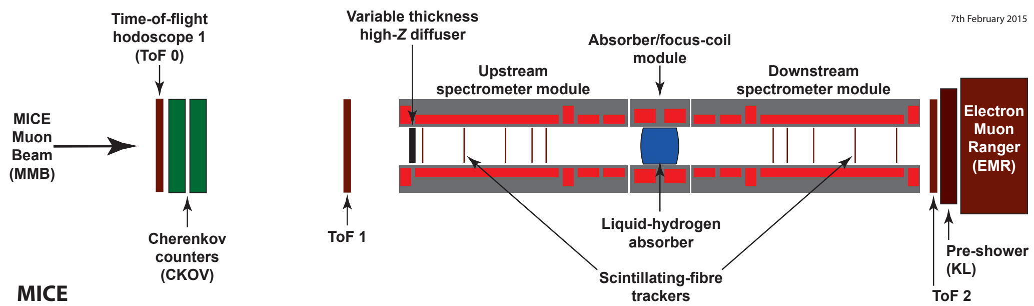

Figure 1. Schematic diagram of the configuration of the experiment. The red rectangles represent the coils of the spectrometer solenoids and focus coil. The individual coils of the spectrometer solenoids are labelled E1, C, E2, M1 and M2. The various detectors are also represented.

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 particle tracks, identifying them, and simulating the response from various detectors, while the accelerator-physics aspect includes the calculation of transfer matrices and Twiss parameters and propagating the beam envelopes. All of these require a detailed description of the beamline, the geometries of the detectors, and the magnetic fields, as well as functionality to simulate the various detectors and reconstruct the detector outputs.

Given the complexity and the time-scale of the experiment, it is essential to ensure that the software can be maintained over the long-term. 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) [9] is the experiment's simulation, reconstruction, and analysis software framework. MAUS provides a Monte Carlo (MC) simulation of the experiment, reconstruction of tracks and identification of particles from simulations and real data, and provides monitoring and diagnostics while running the experiment.

Installation is by a set of shell scripts with SCons [10] as the build tool. The codebase is maintained with the GNU Bazaar revision control system [11] and is hosted on Launchpad [12]. MAUS has a number of dependencies on standard packages such as Python, ROOT [13] and GEANT4 [14] which are built as "third party" external libraries during the installation process. The officially supported platform is Scientific Linux 6 [15] though developers successfully build on CentOS [16], Fedora [17], and Ubuntu [18] distributions.

Each of the MICE detector systems, described in section 1.1, are represented within MAUS. Their data-structures are described in section 2.2 and their simulation and reconstruction algorithms in section 4. MAUS also provides "global" reconstruction routines, which combine data from individual detector systems to identify particle species by the likelihood method and a global track fit. These algorithms are also 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 [19]. 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 back-end and code-sharing of common elements, such as error handling and data-wrangling.

The MAUS data processing model is inspired by the Map-Reduce framework [20], which forms the core of the API design. Map-Reduce, illustrated in figure 2 is a useful model for parallelizing data processing on a large scale. 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; and

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

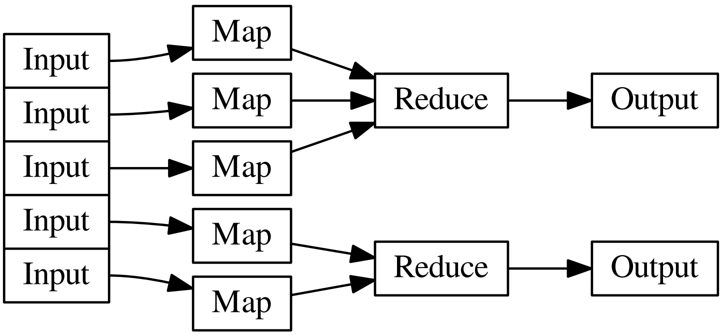

Figure 2. A Map-Reduce framework.

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, the detector geometries, magnetic fields, and calibrations are required by the reconsruction and simulation modules, and objects such as the 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 run-specific information such as electronic cabling maps, fields, and calibrations to be loaded.

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 lasts $\sim 3\,\text{ms}$ 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, an Input module creates an instance of spill data, a Map module processes the spill (simulating, reconstructing, 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.

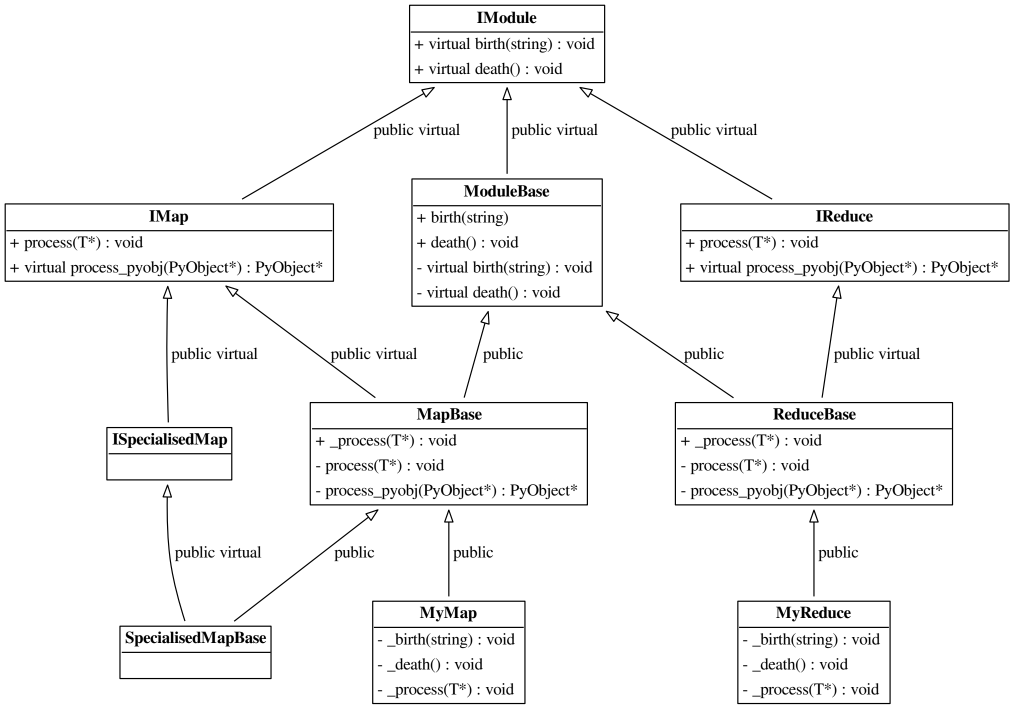

Figure 3. The MAUS API class hierarchy for Map and Reduce modules. The input and output modules follow a related design. *T* represents a templated argument. "+" indicates the introduction of a virtual void method, defining an interface, while "-" indicates a class implements that method, fulfilling 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.

Modules can exchange spill data either as C++ pointers or JSON [21] objects. In Python, the data format can be changed by using a converter module and in C++, mappers are templated to a MAUS data type and an API handles any necessary conversion to that type (see Fig. 3).

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 [13] 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 performed. Each spill is stored as a single entry in a ROOT TTree object. JSON is an ASCII data-tree format. Specific JSON parsers are available – for example, the Python *json* library, and the C++ *JsonCpp* [22] parser come prepackaged with MAUS.

In addition to storing the output from the Map modules, MAUS is also capable of storing the data by produced by the *Reducer* modules using a special *Image* class. This class 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 reconstructed data. The spill is passed between modules and written to permanent storage. The data within a spill is organized 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 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 structure 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 events containing 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 class and the global reconstruction class has several further layers of reconstruction data. This is shown in figures 7–9.

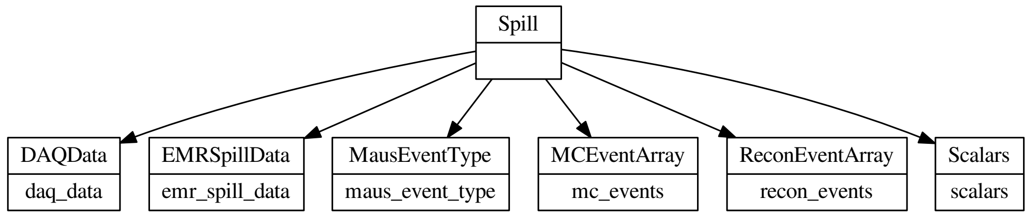

Figure 4. The MAUS output structure for a spill event. The top label in each box is the name of the C++ class and the bottom label is the json branch name.

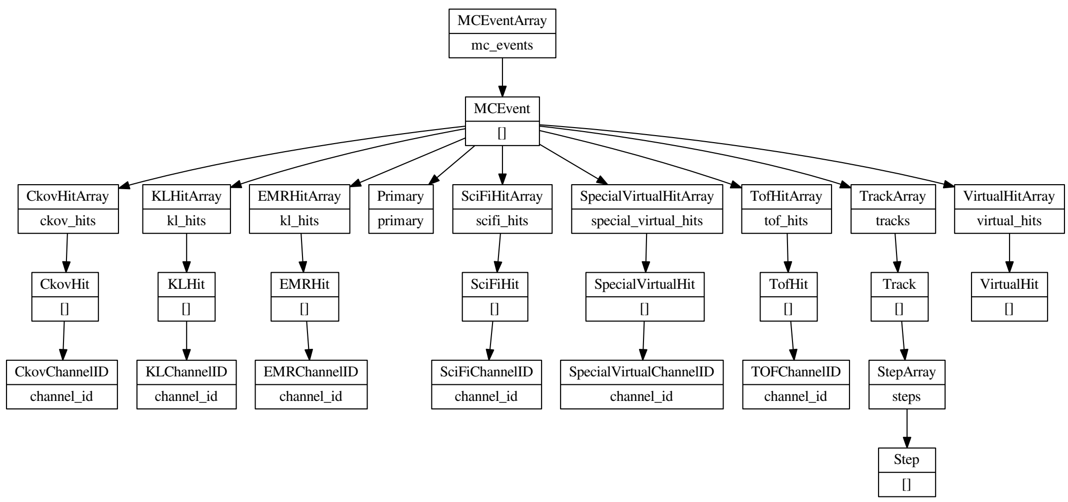

Figure 5. The MAUS data structure for MC events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.

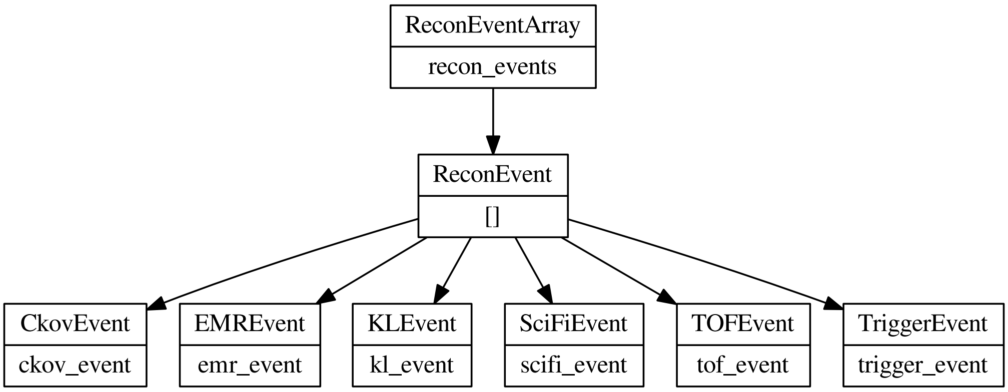

Figure 6. The MAUS data structure for reconstruction events. The top label in each box is the name of the C++ class and the bottom label is the json branch name.

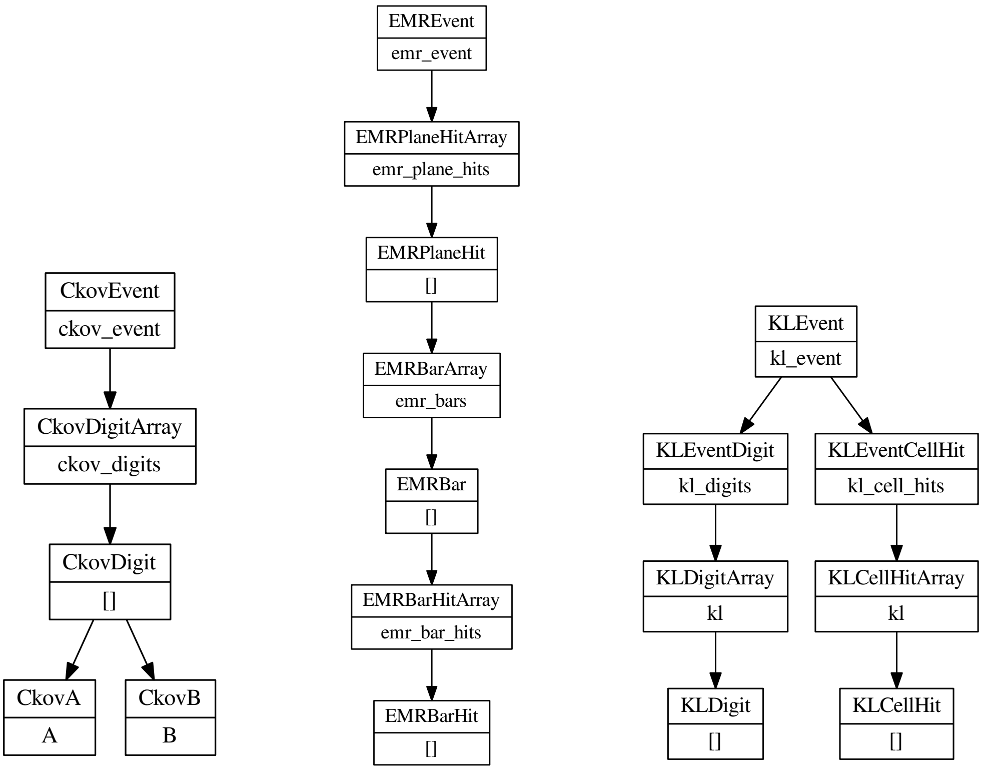

Figure 7. The MAUS data structure for CKOV (left), EMR (middle) and KL (right) reconstruction events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.

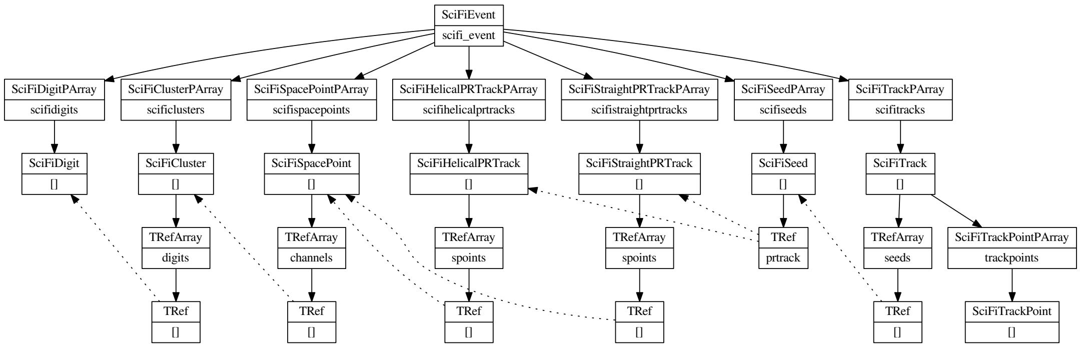

Figure 8. The MAUS data structure for the tracker. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.

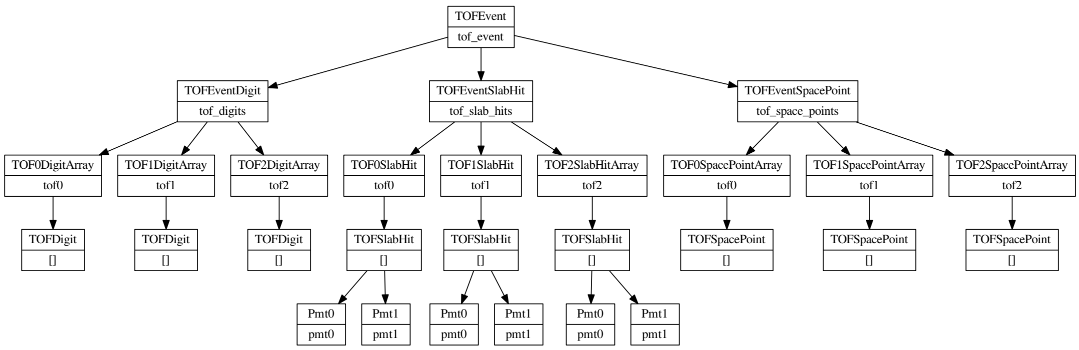

Figure 9. The MAUS data structure for the TOFs. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.

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 with 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. This 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.

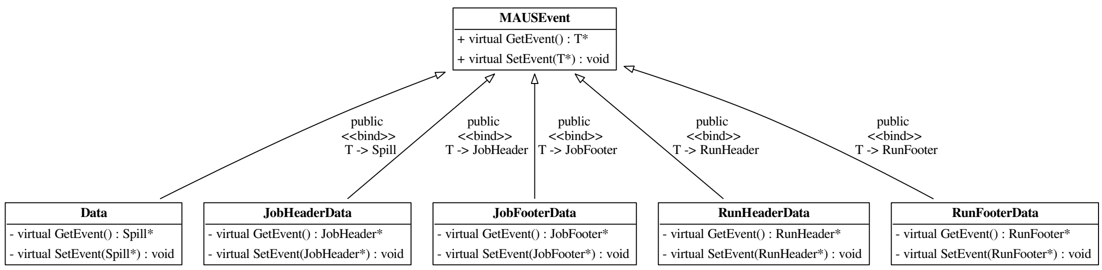

Figure 10. Class hierarchy for the wrappers and base class of the top-level classes of the MAUS data structure.

2.3 Data Flow

The MAUS data flow, showing the reconstruction chain for data originating from MC or real data, is shown in figure 11. Each item in the diagram is implemented as an individual module. The data flow is grouped into three principal areas: the simulation data flow used to generate digits (electronics signals) from particle tracking; the real data flow used to generate digits from real detector data; and the reconstruction data flow which illustrates how digits are built into higher level objects and converted to parameters of interest. The reconstruction data flow is the same for digits from real data and simulation. In the case of raw data, separate input modules are provided to read either directly from the DAQ, or from archived data stored on disk. A reducer module for each detector provides functionality to create summary histograms.

2.4 Testing

MAUS has a set of tests at the unit level and the 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 achieve a high level of code coverage and good test complexity. Integration tests

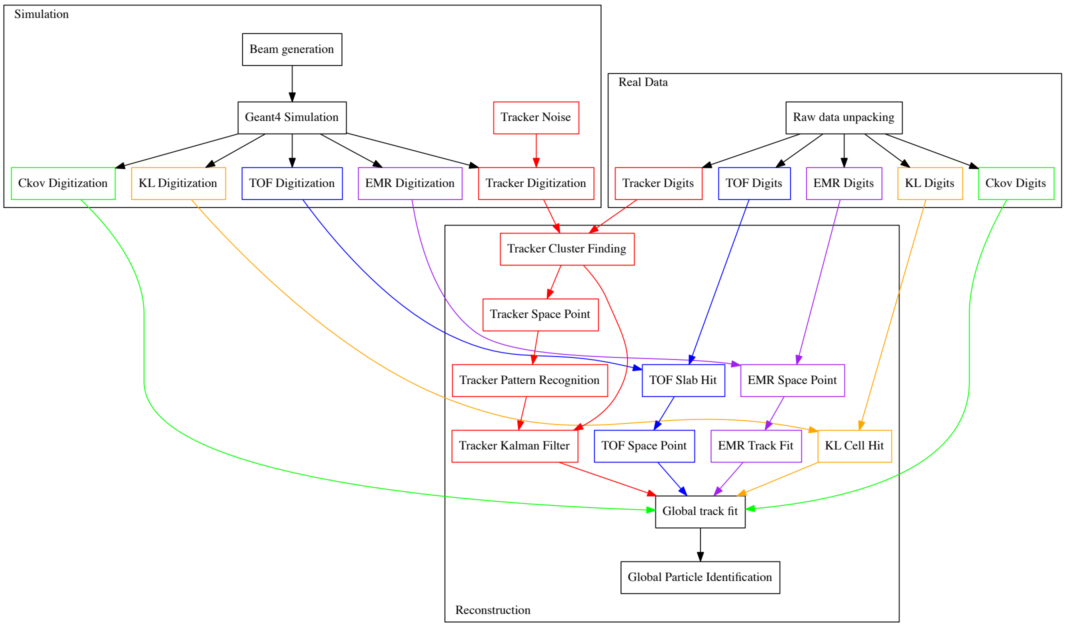

Figure 11. Data flow for the MAUS project. The data flow is color-coded by detector: Ckov - green, EMR - purple, KL - orange, TOF - blue, Tracker - red.

allow the overall performance of the code to be checked against specifications. The MAUS team provides unit test coverage that executes 70–80 % of the total code base. This level of coverage typically results in a code that performs the major workflows without any problem.

The MAUS codebase is built and tested using a Jenkins [23] continuous integration environment deployed on a cluster of servers. Builds and tests of the development branch are automatically triggered when there is a change to the codebase. Developers are asked to perform a build and test on a personal branch of the codebase using the test server before requesting a merge with the development trunk. This enables the MAUS team to make frequent clean releases. Typically MAUS works on a 4–8 week major-release cycle.

3. Monte Carlo

A 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. Routines are provided to sample particles from a multivariate gaussian distribution or generate ensembles of identical particles ("pencil" beams). In addition, 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 created by G4Beamline [24] or ICOOL [25], as well as files in 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.* magnetic field strengths and number of particles to generate, are controlled through data-cards. The magnetic 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 MICE Muon Beam line consists of a quadrupole triplet that captures pions produced when the MICE target intersects the ISIS proton beam, a pion-momentum-selection dipole, a superconducting solenoid to focus and transport the particles to a second dipole that is used to select the muon-beam momentum and a transport channel composed of a further two quadrupole triplets. The GEANT4 simulation within MAUS starts 1 m downstream of the second beamline dipole magnet (D2). 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 processes to model the interactions of particles with matter. The default process in MAUS is "*QGSP_BERT*" which causes GEANT4 to model hadron interactions using a Bertini cascade model up to 10 GeV/c. MAUS provides methods to setup the GEANT4 physical processes which allows the user to control processes with data-cards. Routines are also provided to interface the internal geometry representation in MAUS with GEANT4 descriptions. Finally, MAUS provides routines to extract particle data from the GEANT tracks at user-defined locations.

3.3 Geometry

MAUS uses an online Configurations Database to store all of its geometries. These geometries have been extracted from CAD drawings which are updated based on the most recent surveys and technical drawings available. The CAD drawings are translated to a geometry-specific subset of XML, the Geometry Description Markup Language (GDML) [26] prior to being recorded in the configuration database through the use of the FastRAD [27] 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 libraries in GEANT4 and ROOT for the purpose of independent validation. Because it is a subset of XML, the data contained in the GDML files are readily accessible through the application of the *libxml2* [28] python extension. The GDML files 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) [29].

3.4 Tracking, field maps and beam optics

MAUS tracking is performed using GEANT4. By default, MAUS uses 4^{th} order Runge-Kutta (RK4) for tracking, although other routines are available. RK4 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 in a series of overlapping regions. 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. 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. Pillbox fields can be calculated by using the Bessel functions appropriate for a TM010 cavity or by reading a cylindrically symmetric field map.

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 the tracking output and can be used to transport beam ellipses and single particles.

The accelerator modeling routines in MAUS have been validated against ICOOL and G4Beamline and have been used to model a number of beamlines and rings, including a "neutrino factory" front-end.

3.5 Detector response and digitization

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

The interaction of particles in material is modeled using 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, modeling processes such as the photo-electron yield from a scintillator bar, attenuation in light guides and the pulse shape in the electronics. The data structure of the outputs from the digitizers are designed to match the output from the unpacking of real data from the DAQ.

4. Reconstruction

The reconstruction chain takes as its input either digitized hits from the 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.

4.1 Time of flight

There are three time-of-flight detectors in MICE which serve to distinguish particle type. The detectors are made of plastic scintillator and in each station there are orthogonal x and y planes with 7 or 10 slabs in each plane.

Each GEANT4 hit in the TOF is associated with a physical scintillator slab. The energy deposited by a hit in is first converted to units of photo-electrons. The photo-electron yield from a hit accounts for the light attenuation corresponding to the distance of the hit from the photomultiplier tube (PMT) and is then smeared by the photo-electron resolution. The yields from all hits in a given slab are then summed and the resultant yield is converted to ADC counts.

The time of the hit in the slab is propagated to the PMTs at either end of the slab. The propagated time is then smeared by the PMT's time resolution and converted to TDC counts. Calibration corrections based on real data are then added to the TDC values so that, at the reconstruction stage, they can be corrected just as is done with real data.

The reconstruction proceeds in two main steps. First, the 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 fired, 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 product of the PMT charges is also calculated and stored. Secondly, 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 at the 60 ps level [5].

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 measure the emittance before and after particles pass through the absorber.

The tracker software algorithms and performance are described in detail in [30]. Digits are the most basic unit fed into the main reconstruction module, each digit representing a signal from one 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), 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.

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. The KL has been used successfully to estimate the pion contamination in the MICE muon beamline [31].

4.4 Electron-muon ranger

Hit-level reconstruction of the EMR is implemented in MAUS. The integrated ADC count 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 and momentum reconstruction. The software and performance of the detector are described in detail in [32].

4.5 Cherenkov

The CKOV reconstruction takes the raw flash-ADC data, subtracts pedestals, calculates the charge and applies calibrations to determine the photo-electron yield.

4.6 Global reconstruction

The aim of the Global Reconstruction is to take the reconstructed outputs from individual detectors and to tie them together to form a global track. A likelihood for each particle hypothesis is also calculated.

4.6.1 Global Track Matching

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 RK4 method to propagate particles between these detectors.The tracking is performed outwards from the cooling channel; the upstream tracker through TOF0; and downstream tracker through EMR. It is also available as a commissioning tool providing through-going tracks from TOF1 to EMR, in the absence of magnetic fields. 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 detector. The acceptance criterion for a hit belonging to a track is an agreement within the detector's solution with an additional 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. Hence, it is necessary to perform track matching for all particle types (muons, pions, and electrons). Tracks for all possible PID hypotheses are then passed to the Global PID algorithms.

4.6.2 Global PID

DR note: This is not used/tested in MAUS production – should this stay? Comments?

Global particle identification in MICE typically requires the combination of several detectors. The time-of-flight between TOF detectors can be used to calculate velocity, which is compared with the momentum measured in the trackers to identify the particle type. For all but very low p_t events, charge can be determined from the direction of helical motion in the trackers. Additional information can be obtained 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.

The Global PID in MAUS uses a log-likelihood method to identify the particle species of a global track. It is based upon a framework of PID variables. Simulated tracks are used to produce probability density functions (PDFs) of the PID variables. These are then compared with the PID variables for tracks in real data to obtain a set of likelihoods for the PIDs of the track.

The input to the Global PID is a number of potential tracks from global track matching. Each of these tracks was matched for a given particle hypothesis. The Global PID then takes each track and determines the most likely PID following a series of steps:

1. Each track is copied into an intermediate track;
2. For each potential PID hypothesis x, the log-likelihood is calculated using the PID variables;
3. The track is assigned an object containing the log-likelihood for each hypothesis;
4. From the log-likelhoods, the confidence level, C.L., for a track having a PID x is calculated and the PID is set to the hypothesis with the the best C.L.

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 quantities from detectors. The implementation is through a custom multi-threaded application based on a producer–consumer pattern with thread-safe FIFO buffers. Raw data produced by the DAQ is streamed through a network and consumed by individual detector mappers described in section 3. The reconstructed outputs produced by the mappers, are in turn consumed by the reducers. The mappers and reducers are distributed between the threads to balance the load. Finally, outputs from the reducers are written as histogram images. Though the framework for the online reconstruction is based on parallelized processing of spills, the reconstruction modules are the same as those used for offline processing. A lightweight tool based on Django [33] provides live web-based visualization of the histogram images as and when they are created.

Acknowledgments

The work described here was made possible by grants from Department of Energy and National Science Foundation (USA), the Instituto Nazionale di Fisica Nucleare (Italy), the Science and

Technology Facilities Council (UK), the European Community under the European Commission Framework Programme 7 (AIDA project, grant agreement no. 262025, TIARA project, grant agreement no. 261905, and EuCARD), the Japan Society for the Promotion of Science and the Swiss National Science Foundation, in the framework of the SCOPES programme. We gratefully acknowledge all sources of support. We are grateful to the support given to us by the staff of the STFC Rutherford Appleton and Daresbury Laboratories. We acknowledge the use of Grid computing resources deployed and operated by GridPP [?] in the UK.

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] R. Bertoni et al. The design and commissioning of the MICE upstream time-of-flight system. *Nucl. Instrum. Meth.*, A615:14 – 26, 2010.

[6] L. Cremaldi, D. A. Sanders, P. Sonnek, D. J. Summers, and J. Reidy, Jr. A cherenkov radiation detector with high density aerogels. *IEEE Trans. Nucl. Sci.*, 56:1475–1478, 2009.

[7] M. Ellis, P.R. Hobson, P. Kyberd, J.J. Nebrensky, A. Bross, et al. The Design, construction and performance of the MICE scintillating fibre trackers. *Nucl. Instrum. Meth.*, A659:136–153, 2011.

[8] R. Asfandiyarov et al. The design and construction of the MICE Electron-Muon Ranger. *JINST*, 11(10):T10007, 2016.

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

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

[11] https://bazaar.canonical.com.

[12] http://launchpad.net.

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

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

[15] https://scientificlinux.org.

[16] https://centos.org.

[17] https://getfedora.org.

[18] https://ubuntu.com.

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

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

[21] http://json.org.

[22] https://github.com/open-source-parsers/jsoncpp.

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

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

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

[26] J. McCormick R. Chytracek. Geometry description markup language for physics simulation and analysis applications. *IEEE Trans. Nucl. Sci.*, 53:2892–2896, 2006.

[27] http://fastrad.net.

[28] https://xmlsoft.org.

[29] https://www.w3.org/standards/xml/transformation.

[30] 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.

[31] D. Adams et al. Pion Contamination in the MICE Muon Beam. *JINST*, 11(03):P03001, 2016.

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

[33] https://www.djangoproject.com/.