MAUS Work Specification Document

Change History

Version	Changes	Author	Date
1	First draft	C.Rogers	22/12/2011
2	Add MAUSRunIO and MAUSJobIO objects which reside on master node and handle data structure stuff. Expand other items as appropriate. Add implementation section.	C. Rogers	03/01/2012
3	Update following suggestions by Tunnell	C. Rogers	18/01/2012

Project Summary

Project Title	MAUS_G4BL_Interface
Main Issue	#990
Subtask Issues	
Project Lead	M. Leonova
Project Supervisor(s)	C. Rogers
Associated Manpower	S. Chong Lo, J. Christensen

Table of Contents

Overview of Work

Motivation and Overview

MICE takes it's beam from a target dipping into the ISIS proton synchrotron. Pions are produced in the target and propagated through a series of quadrupoles and dipoles. Particle type and momentum selection is made by a pair of dipoles known as D1 and D2. The region upstream of D2 is referred to as the upstream beamline, as this region contains a high flux, impure beam.

The region downstream of D2 is referred to as the downstream beamline as this region can contain a low flux, pure beam suitable for ionisation cooling measurement. Several detectors sit in the downstream beamline, namely TOF0, Ckov and TOF1. These detectors form a part of the main diagnostic system of the ionisation cooling channel, providing information essential for particle type identification and timing information for analysis of a pulsed beam.

In MICE, modelling of the upstream and downstream beamline is performed by G4Beamline, a software code developed primarily at Fermilab for tracking particles through accelerator lattices. Modelling of the sensitive detectors and cooling channel is performed by MAUS, a custom software code developed by the MICE collaboration.

It is desirable that the MAUS software be able to read in a beam with a realistic distribution of particles characteristic of the MICE beamline. Two basic scenarios are foreseen:

- Data is taken. Set current values for magnets and proton absorber settings are stored in the MICE Configuration Database (CDB). Subsequently, it is desired to run a Monte Carlo simulation of the cooling channel using a realistic model of the upstream beamline. Set currents should be read from the Configuration Database and fed into the G4Beamline model for the upstream beamline. The upstream beamline should be tracked and an input beam for MAUS provided.
- A user chooses to run a Monte Carlo simulation using some user-defined settings. Magnet currents and other settings should be read from MAUS Configuration datacards and fed into the G4Beamline model for the upstream beamline. The upstream beamline should be tracked and an input beam for MAUS provided.

The interface point should be on the upstream edge of D2. It is foreseen that the field in D2 may be used as part of the reconstruction routine, so this should be modelled in MAUS.

Implementation

A new Map module should be developed that will generate a beam file at the upstream edge of D2 from G4Beamline suitable for input into MAUS. The script should be developed as a Map module in the MAUS framework. It should be possible to control the script using either MAUS cards or CDB settings. The beam should generate the correct transverse and momentum distribution and also a reasonable time distribution.

Task Breakdown

An update should be made to

G4Beamline Driver script

The first task is to make a driver script. In MAUS, scripts should be written in python. The script

should call the subprocess module to execute G4Beamline, feeding the correct parameters into the G4Beamline command line. The following variables will need to be set:

- Q1, Q2, Q3 currents
- D1, D2 and decay solenoid currents
- Proton absorber thickness
- Number of particles produced at the target

At this stage they should be stored as hard-coded global variables.

Please note the python coding style guidelines and testing guidelines outlined below. Every function should have an associated test function; code should not raise any pylint errors; every function should have a docstring detailing what the function does, what it returns and the meaning of any input parameters.

Installation of MAUS

MAUS should be installed as per the instructions at

http://micewww.pp.rl.ac.uk/projects/maus/wiki/Install

A new development branch should be created as per the instructions at

http://micewww.pp.rl.ac.uk/projects/maus/wiki/Bzr_usage

Implementation as MAUS Mapper

The G4Beamline script should be modified to function as a MAUS Map module called MapPyBeamlineSimulation, placed in file

$MAUS_ROOT_DIR/src/map/MapPyBeamlineSimulation/MapPyBeamlineSimulation.py

Map modules require

- **__init__** function that initialises data
- **birth** function that reads in MAUS configuration datacards and returns True on success (initially just return true here)
- **process** function that generates particle data for a spill
- **death** function that deletes any variables initialised in **birth** and then exits

For an example mapper, see *$MAUS_ROOT_DIR/src/map/MapPyDoNothing*. The MapPyBeamline module should put data into the spill every time the process function is called. The tests should be implemented as TestMapPyBeamMaker. Note: the tests should skip if no version of G4Beamline is available.

Interface Using MAUS Configuration Datacards

The birth function receives a copy of the Configuration datacards (encoded as a json string). These should be parsed into a python dictionary and then used in place of the global variables used initially.

Interface using CDB controls

An extra option “beamline_simulation_settings_from_cdb” should be added to the configuration datacards. When set to true, MapPyBeamlineSimulation should interrogate the configuration database for

- Q1, Q2, Q3 currents
- D1, D2 and decay solenoid currents
- Proton absorber thickness

Documentation

A new latex file should be added to the documentation in *$MAUS_ROOT_DIR/doc/doc_src* detailing in brief the function of the MapPyBeamlineSimulation module, together with a table explaining the relevant control variables. This should be complementary to more detailed documentation in the mapper class itself.

G4Beamline Third Party Package

Two bash script should be added to the third party download scripts,

- *$MAUS_ROOT_DIR/third_party/bash/54beamline_geometry.bash*
- *$MAUS_ROOT_DIR/third_party/bash/55g4beamline.bash*

The first should download the latest copy of the MICE beamline geometry files. The second should download the latest copy of the G4beamline code.

Integration Test

An integration test should be written that models the input beamline using some known beamline settings and checks that the outgoing beamline distribution follows some well defined behaviour. Typical quantities to check against might be

- Particle rates for different particle species
- Transverse and momentum distributions

Time Distribution from User Input

Additional control variables should be made available to modify the input time distribution. Users should be able to simulate either a gaussian, flat top or sawtooth time distribution.

Time Distribution from Reconstructed Data

Additional control variables should be made available to modify the input time distribution based on a pre-analysis of the time distribution of input triggers from reconstructed data and performing some fitting to the reconstructed triggers.

MAUS Overview and Requirements

MAUS is an analysis tool designed for the reconstruction and analysis of the Muon Ionisation Cooling Experiment.

The MAUS project uses a number of tools to manage code and the development process. Issue trackers are used to monitor development of new features and log any issues arising such as bugs. Code is stored using Version Control System (VCS). Communication between the development team is achieved using mailing lists, regular phone conferences etc. Code quality is assured by a set of style requirements, testing requirements, documentation of code and regular code reviews.

These tools are documented here for convenience, but additional documentation can be found on the MAUS website http://micewww.pp.rl.ac.uk/projects/maus/wiki/MAUSDevs. Where there is a conflict, the MAUS website takes precedence as this is more likely to be updated.

Issue Tracking

The redmine issue tracking system is used to mointor development of new features and log any issues arising such as bugs. Any information pertaining to the features in this document shall be stored in the relevant issues on that system. Redmine issues can be found by following the *Issues* link on the MAUS website.

Code Version Control

Code is controlled using the *bazaar* Distributed Version Control System. **Before** writing code, developers shall checkout a copy of the maus trunk (*lp:maus*). Developers shall then make their own development branch on launchpad. Code shall be pushed regularly to this development branch (typically nightly). Every release, the MAUS trunk shall be merged with the development branch to ensure that development is performed against the latest MAUS version.

Further guidance on bzr usage can be found on the MAUS wiki.

Communication

Communication is performed using mailing lists and fortnightly phone meetings. All developers should be subscribed to the following mailing lists:

- mice-software
- maus-devs
- maus-users

Where possible, developers shall attend the fortnightly phone meetings, as announced on the mice-software mailing lists.

Code Quality

Code quality is ensured by a number of tools. Code must be commented and conform to certain style requirements. All code must have appropriate regression tests. Documentation must be created or updated as appropriate. Code may be reviewed by one or more developers before submission.

The workflow that ensures code quality is controlled by the workflow field in the issue tracker. As code moves through the workflow, this field should be updated.

Coding Style and Comments

Code shall be written in the appropriate style. For C++, we follow the google style guide; for python we enforce the standard python style guide. Scripts that automatically check for code style are provided and are executed upon execution of the unit tests.

All public functions shall be commented. C++ functions should have a description in the header file. Python functions should be commented using python standard docstrings. Comments should include the purpose of the function, the definition of any input parameters and the values that can be returned. Python comments should specify possible types for all input and output parameters. Comments should be formatted for the Doxygen tool.

Testing

Unit tests shall be provided by the developer for all code they develop at the 90% line coverage level. We aim to provide about 90% line coverage for our unit tests. Tests for modules (maps, reduces, inputs, outputs) should be written in python and kept in the folder for that module. Tests for common code (src/common_cpp, src/common_py) should be written in the same language as the code to be tested and kept in tests/cpp_unit or tests/py_unit respectively. All C++ code should be compatible with the google testing framework. All python code should inherit from the python unittest framework.

The MAUS test server is used to replicate environments that the code is required to be run in such as the MICE control room and standard high energy physics clusters. All tests must pass not only in the developers local environment but also in the test server environment.

Documentation

High level documentation should be written in latex and placed in the doc/doc_src area. All user interfaces should be documented (either input or output) in latex.

Code Review

All code by developers new to the project shall be reviewed by the project supervisors. Additional reviewers, or code reviews, may be requested by the project supervisor. The code should be reviewed by the project supervisor when it is ready to merge and all tests pass on the test server.

Code review is a way of:

- finding bugs
- spreading knowledge of the code to other developers
- ensuring high quality code by:
    - checking that code is adequately tested
    - checking that the code is documented properly
    - checking that the code has the correct style

When code is ready for review, please set the workflow field on the issue tracker to the appropriate value.

Effort and Timescale

Effort Available

Major Milestones