

GENIE Validation at Fermilab

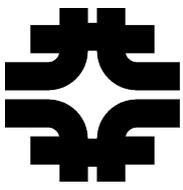
Gabriel N. Perdue
Fermilab



GENIE

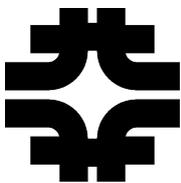
- **Generates Events for Neutrino Interaction Experiments.**
- <http://genie.hepforge.org>
- Well-engineered C++ software framework built on sound OO-principles and design patterns. (The Gang of Four is omnipresent.)
- Propagates a flux of neutrinos (specified by function, histogram, or ntuple) through a geometry (Geant4-compatibility is an option) and simulates the initial interaction and propagation of hard vertex products through the nuclear medium. Geant4 takes over when particles leave the nucleus.
- ROOT provides many core utilities. GENIE also heavily leverages other HEP and FOS software – LHAPDF, GSL, Pythia, log4cpp, etc.

Andreopoulos, C. and Bell, A. and Bhattacharya, D. and Cavanna, F. and Dobson, J. and others.
"The GENIE Neutrino Monte Carlo Generator". Nucl.Instrum.Meth. A614. 87-104. 2010.



GENIE

- Created to be the “universal event generator” (covering low energy reactor experiments, solar, supernova, meson decay at rest, accelerator-based experiments, and all the way through PeV+ cosmic experiments) requested during the NuInt conference series.
- It is the most widely-adopted neutrino event generator. Competitors are brittle FORTRAN projects or lack comprehensive features like a flux driver, highly flexible configuration, re-weighting machinery, geometry drivers, charged lepton and hadron interaction drivers, etc.
- Good separation of different levels of abstraction – event handling is decoupled from physics routines, physics routines use visitor and chain of responsibility patterns to allow for fairly arbitrary algorithm stacks.
- Cross-sections are pre-computed and stored in configuration XML (ROOT and XML are both heavily used to store computation results, physics output, and configuration options).

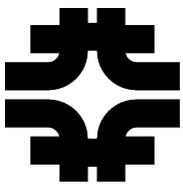


```
> ./cloc-1.60.pl R-2_8_0/  
 3285 text files.  
 3200 unique files.  
 7197 files ignored.
```

```
http://cloc.sourceforge.net v 1.60 T=113.14 s (11.3 files/s, 4119.1 lines/s)
```

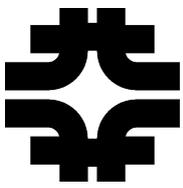
Language	files	blank	comment	code
C++	525	30478	37587	176349
XML	125	21895	2144	147176
C/C++ Header	504	9052	8118	22282
Perl	28	456	1469	3620
make	47	514	485	1651
Bourne Shell	34	157	334	1059
Bourne Again Shell	2	145	127	727
SQL	12	37	0	117
ASP.Net	1	0	0	39
SUM:	1278	62734	50264	353020

There is a lot of configuration XML and experimental data packaged for the validation framework.



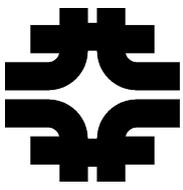
Challenges

- GENIE's framework is good, but the physics models implemented lag the state of the art.
- Limited manpower (1.5 FTE of active labor as of Summer 2013 when FNAL joined) has meant slow release schedules and new feature implementation.
- **We must be able to issue releases faster.**
- We should be able to produce experiment / target / energy-regime specific tunes.



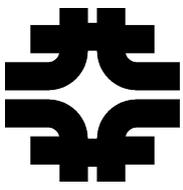
Our Goals

- C. Andreopoulos (GENIE spokesperson) articulated a clear purpose for GENIE validation:
 - Reduce cycle time.
 - Cycle time is the time required to implement a one-line bug-fix and prove there were no unintended consequences anywhere in the software.
- **The GENIE collaboration takes total package integrity VERY seriously – the validation process is the gatekeeper to issue a release.**
- This is why the validation is important. **Fast, stable validation means a rapid release cycle becomes possible.**
- GENIE has a good validation framework for a HEP software project, but cycle time is currently ~2 months.
 - Our goal is one week.



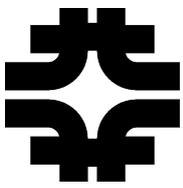
Basic Validation

- One week is a practical lower bound – it takes long enough to create the required samples and enough space to store them that nightly validation is not practical (although nightly *builds* and time-scale appropriate integration tests are required). The process:
 - Check out the code and build it. Requires a couple of hours and less than one GB of space.
 - Compute total cross section splines (large files, hundreds of MB per target (free nucleons, carbon, argon, etc.)). Requires ~dozen hours per target and 10's of GB of space (total). These are input for the following step.
 - Compute validation sample, e.g. muon neutrinos over the NuMI flux to replicate final state muon energies and angles for Deep Inelastic Scattering events in MINOS. Samples need to have tens of thousands of events so run long (~dozen hours) and take significant space (several GB per app, possibly many dozens of apps).
 - Compare the validation sample and experimental data. Run GENIE apps (compiled C++) over the outputs from the previous step. These are typically much faster (less than an hour) and produce analysis-style histograms (but may produce many of them, so they could easily require hundreds of MB per app).



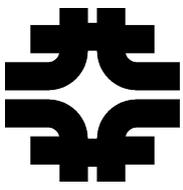
Validation

- Many levels:
 - Compile. Does the code build? Where does the build fail?
 - Run one event. Can the program run without crashing?
 - Run one million events. Is the program robust across the whole phase space?
 - Physically sensible. Is charge conserved? Momentum? Unitarity?
 - Agreement with experimental data. Do the predictions of the generator agree with experimental data within uncertainties? Are available experimental uncertainties accounted for correctly in figures of merit?
 - Confidence of predictions. Are uncertainty bands attached to predictions of the generator where appropriate?
 - Usability. Can experimenters quickly understand the differences between releases and understand the regions of validity for the generator?
- We need to be able to address every level eventually (although not all applications will access every level).



Validation

- Two components:
 - Validation applications: often specific to experimental results (to account for the different sets of uncertainties provided), but sometimes general. These programs compare predictions of the generator to data.
 - The validation framework: infrastructure to automate the production of MC samples and uncertainties, run the applications, store and summarize the results, present summary statistics and histograms. We would like to be able to run the validation weekly using an automated framework. It should present easily consumable summaries and detailed plot books for a thorough reference.



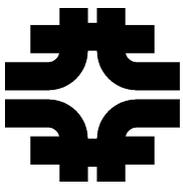
Important Caveats

- This presentation is meant to express a set of ideas that should spark discussion and debate in the formation of requirements, timelines, and milestones.
- The "official" definitions should be specified in the FNAL–GENIE MOU and subsidiary documents.
- This presentation will focus on the validation framework, but we should not fail to give the appropriate emphasis to the validation applications. Physicist judgement is required to make progress on the validation applications and GENIE leadership has stressed their importance in a fully functioning validation.



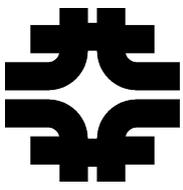
Validation Applications

- Active work at Fermilab:
 - J. Yarba on the hadronization package.
- There are many existing apps. We need to be able to re-use as many of these as possible, but we will probably need to do some refactoring / rewriting in some cases.
- Philosophy: think carefully about input and output requirements and make sure we can develop a uniform way of specifying required input, required auxiliary files, how to handle output, etc.



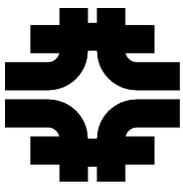
Validation Applications

- The current set is reasonably comprehensive but some pieces are missing:
 - Re-weighting (some pieces are available, but nothing coherent)
 - Flux driver
 - Geometry
 - Many recent "flagship" results (e.g., MiniBooNE double-differential QE results, MINERvA results, etc.) that are important to the community lack comparisons.
- Good interface design means we can and should support work on applications concurrent with framework development.
- We should especially aim to provide clear guidance to developers working on new features now – how do they build validation into the model they are working on now?



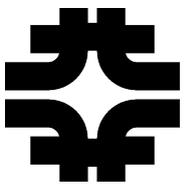
Validation Framework

- We will build a framework that lives at Fermilab and attempts to **reuse as many existing software products** as possible.
- New **components should be as general as possible and well-focused** ("Unix philosophy"). The idea is to create products usable across the lab that can fit into different use cases.
- Philosophy: get the minimum viable framework running quickly, and add features incrementally.



Institutional Concerns

- Currently, most the validation runs on the batch system at Rutherford–Appleton Laboratory.
- Different parts run at the University of Pittsburgh.
- GENIE is a universal generator and is used by non–FNAL experiments. It is a requirement that these collaborations be able to access validation results.
- We additionally require a highly factorable system so it is portable with the minimum reasonable amount of effort. Some pieces will invariably be institution specific (e.g., dCache/Enstore at Fermilab), but the framework must be designed such that with minimum reasonable effort, different components of the system may be swapped in and out as needed at different institutions.
- Individual validation applications should always be executable "by hand" in an interactive environment for supported operating systems.



Basic Plan

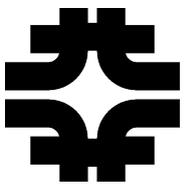
Specify Requirements

Design Minimum Viable Product

Implement MVP

Use lessons from the MVP to design the Full Product.

Implement the Full Product.



Step

MVP

Full Product



Build

Check out one branch of the code.

Build the code on SLF.

Report errors to a webpage.

Build multiple branches.

Email summaries to watchers.

Build the code on Ubuntu, OSX, etc.

Generate

Generate Cross Section Splines

Generate App Specific Events for One App

Generate App Specific Events for Many Apps

Monitor resource usage: memory, CPU, etc.

Check 1

Search logs for errors.

Comprehensive tests for conservation laws.

Check 2

Run one validation app.

Run all validation apps.

Compute uncertainties on the predictions.

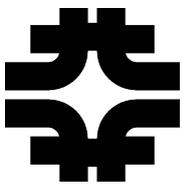
Summary

Aggregate validation results for one app.

Aggregate results for all apps.

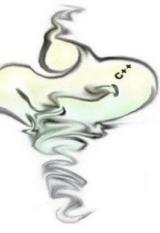
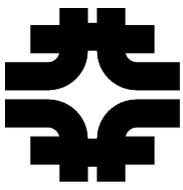
Produce high level summaries.

Compare to reference releases.



Requirements: Hardware

- Computing resources to build the code and store ~ 10 releases (\sim several reference releases and one–two months of weekly builds).
- Re–use third–party software when possible.
- Storage for ~ 10 validation cycles. Assume ~ 1 TB per cycle.
- Web server for error reporting. Ideally serve a history of validation reports.
- Grid allocation to run $O(1000)$ jobs per week.



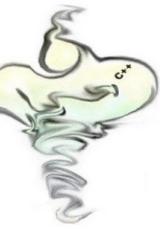
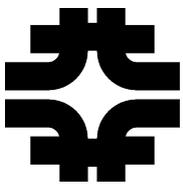
Requirements: Hardware

- We would also like to secure interactive nodes for development purposes.
- The needs here are modest (perhaps two interactive nodes), but we would want sufficient computing power to serve the core group and GENIE developers in the FNAL Users community.
- For example, some users would want to work with release candidate snapshots built through the validation process and it would help with reporting and tracking if we had a common environment for development at the lab.



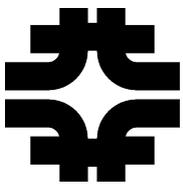
Requirements: Hardware

- We would also like to secure one special dedicated node for performance profiling.
- Memory usage can be usefully monitored on a generic Grid node, but careful performance optimization requires a "quiet", stable machine.
- Because this node would not be in constant use, it would make sense to share it with a small number of other groups with similar needs at the lab – there is a lot of synergy and personnel overlap with Geant, for example.



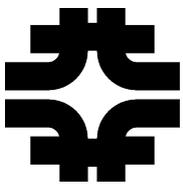
Requirements: Software

- Scheduler: run each piece of the framework, monitor results at the proceed/stop level, notify principals if there are problems / success.
- Build system: check out specified branches (e.g., "integration") of the code, run a build script ("lamp" project, or other).
- Job submission to use FermiGrid CPU.
- Data management: store / retrieve cross section splines (0(10 - 100 GB)), retrieve experimental data sets (< 1 GB, but many), store / retrieve validation app output (< 1 GB, but many). Archival storage can bundle the entire output into one tar-ball (hundreds of GB), with high-level summary plots (< 10 MB?) and detailed plot book (hundreds of MB) ideally simply accessible.
- Copying H. Wentzel's Geant4 validation repository web app to display current and historical plots would be a good idea.



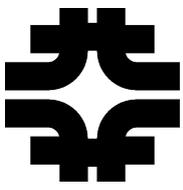
Schedule

- Aim for the minimum viable product (MVP) first:
 - Secure appropriate hardware resources. (The overall effort will scale with this factor – pieces may be delayed if we need to first secure funding, etc.)
 - Checkout and build only one branch (trunk), and report errors to a webpage. Okay to overwrite the existing installation (but "rolling" directories would be better).
 - Generate splines and samples appropriate for one application.
 - Run the application and analyze the output produce comparison plots.
- Add other features (more comprehensive sanity checks, resource monitoring, high-level summaries, use reweighting to compute uncertainty bands, creation of flexible tunes, etc.) once we have something running weekly. (But we should not design so these features are hard / impossible.)
- Iterate to add additional apps interspersed with other features.



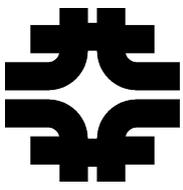
Schedule

- Times are *wall-clock* times that reflect goals.
- Aim for the minimum viable product first (6-week project):
 - Secure appropriate hardware resources. One-week project (assuming resources are available).
 - Checkout and build only one branch (trunk), and report errors. One-week project.
 - Generate splines and samples appropriate for one application. Two-week project.
 - Run the application and analyze the output produce comparison plots. Two-week project.
- Add other features. 6-month project. We should begin to prioritize the task list once the MVP is finished.
- Iterate to add additional apps. 6-months to 1-year to integrate all existing applications (including current core apps plus Pittsburgh apps). May proceed concurrently with other feature expansion.



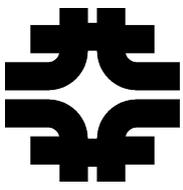
Key Drivers

- Hardware resources.
 - What is available now? How long will it take to secure new resources if required? The expectation is that it may take some time (months) to secure resources – computers and disk don't appear through magic. We need to scale this effort to available resources as they come online.
- Personnel.
 - Validation apps: J. Yarba, R. Hatcher (?), M. Zielinski (?), other developers at Fermilab.
 - Framework: G. Perdue, other developers at Fermilab.
 - Note: I write "good code for a physicist." The framework could use architecture and implementation advice and assistance from computing professionals at the lab. **At the minimum I will need consulting services from lab specialists on existing products.** The validation framework should utilize as many existing lab products as possible and new products should be application-neutral to encourage re-usability.
 - We could probably use ~ 0.25 FTE for three months (maybe a lower FTE number, but the needs would be "bursty").



How will this be tracked?

- We plan on using Redmine.
- The initial plan will be to plan and divide responsibilities, connecting milestones with individuals using a Gantt chart.
- We will also use issue tracking on Redmine to organize new ideas as they occur and to report and handle bugs.



Recap

- Our goal is to reduce the GENIE validation cycle time to one week.
- We need a grid allocation sufficient to run $O(1000)$ jobs per week and 10 TB of storage (rolling usage).
- We need a way to serve the results of the validation suite and to display plots (with official releases publicly archived).
- We need some software consulting expertise on using lab software products (new build service, job_sub, etc.) and some help producing new, general use products at the level of 0.25 FTE for three months.