Overview of the CMS Framework Review

Christopher Jones

CMS Software Scale

People

3000 collaborators
128 contributors to production code since June last year

Code

2.5M LOC C++
1.3M LOC python
670 shared libraries and 1300 shared plugins

Usage

100,000 concurrently running jobs
200M jobs per year
60B events processed (simulation and data)
40PB of MC and Data

Usages

High Level Trigger

Prompt Reconstruction

Prompt Calibration

Full Calibration and Alignment

Supports iterative refinement

Simulated Event Generation

Detector Simulation

Supports MC and data event mixing

Reconstruction

Analysis

Event Display

Random event access

Modify settings of algorithms (modules) while processing

Data File Merging

Used by CMS grid tools

Features and Subsystems

Python based configuration

Event processing engine

State machine driven

may be changed for threading extensions

Runs, LuminosityBlocks, Events, open/close files

Conditions

Validity interval based

Manages dependencies between conditions

Plugin management

Publish/subscribe data flow

Provenance tracking

Job configuration

Event selection decisions

Plugin Based Framework

Processing Plugins

Algorithms encapsulated in modules

Types: Producers, Filters and Analyzers (3424)

Processing Storage Plugins

Encapsulation of I/O

Types: Sources (20) and OutputModules (7)

Conditions (EventSetup) Plugins

Conditions/Alignment/Geometry encapsulated in modules

Types: Sources (114) and Producers (357)

Miscellaneous Plugins (Services)

Non-physics changing extension (80)

Can monitor the state of the framework

e.g. what modules are being run at the moment

Event Looping Plugins

Controls repeated looping over events in job (20)

Generalized Plugins System

80 plugin types used by developers

Job Configuration

Jobs are configured using python

python objects are used to create a C++ tree structure
- edm::ParameterSet class

Module validation code can modify its part of the tree structure

Module constructors are passed their part of the tree structure
- Immutable in constructor

Final configuration tree stored in output files
- Provenance tracking
- Serialized as a list of (hash, string) pairs

Support tools
- inspect provenance in files
- query modules as to allowed parameters

Multi-threaded Processing

Present

Supports processing multiple events concurrently
Supports use of Intel's Thread Building Blocks from within a module
Provides mechanism to serialize thread unsafe algorithms
All modules must declare what data products they will consume
Provides a thread-safe message logging system

Spring

Support concurrently running modules processing same event
A prototype of the code already exists

Future

Support concurrent processing of LuminosityBlocks and Runs

Data Model Support

Modules are passed Run/Lumi/Event/EventSetup to process

Producers publish products to Run/Lumi/Event/EventSetup

Data and conditions products are immutable once published

const member functions must return same value given the same inputs

Data Relationships

edm::Ref<> persistent index into any container, fast

edm::Ptr<> persistent index into most containers, supports polymorphism

edm::RefToBase<> persistent index into any container, supports polymorphism

- extremely difficult to specify all classes it needs for storage
- deprecated

edm::AssociationVector<> associates data to each item in another collection

edm::AssociationMap<> associate one or more data to items in a collection

- deprecated

edm::ValueMap<> associates data to items in multiple collections

- more efficient memory and I/O than edm::AssociationMap

Polymorphic and container type agnostic lookup

edm::View<>

Data Passing Interface

Data is requested from a generic container passed to modules

edm::Event, edm::Run, edm::LuminosityBlock and edm::EventSetup

requests are done in a type safe manner

Reduced set of requests

::getByLabel<T>

pass set of strings which uniquely identify a product

pass edm::InputTag which encodes the same strings as above

::getByToken<T>

pass an edm::EDGetToken which uniquely identify a product

edm::EDGetToken obtained by calling consumes on module's constructor

::getManyByType<T>

gets all products of that type

Data publishing

::put<T>

passed an std::auto_ptr<T>

Data 'Mixing'

Event pile-up support

Ability to take MC data products from N secondary events and accumulate them into the primary event

Used by all simulation jobs to approximate multiple beam interactions per event

Knows how to change timing of hits to correspond to 'bunch crossings'

Re-engineered to use products from only one secondary event at a time

'Digi' mixing support

Ability to take one secondary event and mix digis with primary event

Digi: data for one detector Plugin after calibration has been applied

Used for embedding studies

Inject a known simulation event into a real data event to study tracking efficiency

Re-engineered to allow standard 'raw to digi' modules to be used internally

I.e. uses a reduced version of the framework internally to run standard modules