Modularization of DetSim and DataPrep in DUNE and beyond

LArSoft coordination

David Adams
BNL
September 27, 2016

Introduction

Last year I looked at early stages of DUNE data processing

- Module that does detector simulation
  - Input is sim::SimChannel (derived from GEANT hits)
  - Output is raw::RawDigit (larsoft formatting of raw TPC data)
- Module that does "data preparation"
  - Input is raw::RawDigit
  - Output is recob::Wire

Some observations

- There was significant code duplication
  - Separate modules for 35t and FD
    - (And for other DUNE and non-DUNE detectors)
- Each module did many things
  - Many lines (> 1000) of code
  - Difficult to understand what was done and where
- Both make it difficult to understand and improve code

Introduction (2)

Decided to rewrite the code

- New DetSim module delivered last winter
  - Worked for both 35t and FD
  - Service to extend to DP (dual phase) provided soon thereafter
    - By others with no help or guidance from me
- New DataPrep module delivered over the summer
  - Validated for 35 ton and FD
  - Still need fcl, validation (and C++ mods?) for protoDUNE and singe-phase

Design goals

- Consolidate so same module is used for different detectors
  - Remove code duplication
- Add modularity at finer granularity
  - Runtime user can easily swap out different actions (e.g. ZS or noise supp.)
  - Developers can easily contribute ideas for these actions
- Like to access much of the functionality outside of art
  - E.g. standalone program, Root script, next generation framework

Design

Development plan

- Identify the "actions" carried out by each module
  - Here action is a processing step with well-defined input and output that can be carried out independent of the other actions
  - Do not require that this transient output be distinct from input
    - E.g. action might update an existing structure rather than creating new
    - Thus avoid data duplication or cumbersome art associations (for transient data)
- Specify the interface for each action
  - I.e. the input and output data
- Provide a default implementation for each action
  - Code that reproduces what was done in the old modules
    - Provide multiple implementations if the old code had distinct options
  - Later developers may add other implementations
- Provide a new module that
  - Reads input data from event store
  - Carries out the actions via their interfaces
    - This sequence of actions may itself be an action
      - » Called Higher-level action (HLA)
  - Writes the output data to the event store

Design (2)

Action hierarchy

- Useful to have a hierarchy of actions
- So one action can invoke another
  - HLA (higher-level action) is an action that invokes another action
- Levels:
  - High level derives one data product from another
    - Either is a (producer) module or has a module wrapper to allow use in art
  - Next level down are the steps in this derivation
  - Next are the utilities or tools used in these steps
    - E.g. a track fitter
- Note that runtime configuration (e.g. fcl) is desired at all levels

Actions for data preparation

There are many steps to prepare data

- Extract raw data from larsoft RawDigit container
  - Uncompress, int-to-float, subtract pedestals
  - Flag under/overflows, stuck bits and other detector-specific issues
- Mitigation
  - E.g. interpolation for stuck bits in DUNE 35t
- Early signal finding (to aid in noise removal)
- Noise removal
  - Single-channel, coherent or both
- Pedestal adjustment
- Deconvolution
- ROI building (more signal finding)
- Construct output data product (recob::Wire)
  - With associations to digits

These naturally map onto step actions (previous pages)

Packaging actions

How do we package the SW for each action?

- Natural to have a C++ class for each action implementation
- Use C++ base class for the action interface
  - Client C++ code only uses the interface
- Typical action will have some configuration data
  - Configuration includes implementation class name and object data
  - Should be able to specify/modify configuration at run time
  - Natural to use FCL for DUNE/larsoft
  - Useful to be able to name standard configurations
    - So provenance record can refer to a name rather than full parameter map
- Require means to discover actions at run time
  - HLA (e.g. a module) does not include action (implementation) headers
    - Does include interface headers
    - Configuration specifies the action class and its parameters
  - Module library need not be linked against action library
    - Framework (or other magic) find and loads the required libraries

Options for packaging actions

Utility class aka provider (no inherent art/larsoft dependence)

- Can get configuration with convention to construct from FCL
- No means for discovery

Art module

- Provides configuration and discovery
- Can only be called from art framework—cannot be used in hierarchy

Art service

- Provides configuration and discovery
- Can be used outside art framework
  - Requires art libraries and lightweight fcl registration wrapper
    - See dunetpc/dune/ArtSupport/ArtServiceHelper
    - Need to add discovery
- But service is singleton
  - Job can only access one configuration for each action interface
  - Can work around this by introducing additional interfaces and complex mapping between interfaces and implementations. Not desirable.

Options for packaging actions (2)

Art tool

- Proposed extension to art
- Just like a service except multiple instances would be accessible
- Access by interface name and instance name
  - Instead of interface name alone
  - ServiceHandle<MyInterface>() → ToolHandle<MyInterface>("myinstance")
- Prefer this implemented so that existing services become tools
  - Or some trivial change e.g. "Service" → "Tool"
  - No need to modify existing services
  - Developer need not decide in advance if an action is a service or a tool
  - Job might event use an interface as both service and tool
    - E.g. default service provides pedestal to multiple clients while a calibration module applies refined (e.g. dynamic) pedestals

Options for packaging actions (3)

Current strategy: TSI (tool-service-interface) model

- Package actions as services
- With a distinct interface class for each type of action
  - Clients only access actions via these interfaces
- Duplicate code or build complex structures where distinct actions could benefit from the same code
  - E.g. zero suppression and ROI building
- Hope that we one day get art tools
  - Then remove preceding duplication and revisit the choice of interfaces
  - Getting tools sooner ➔ cleaner code and less overall effort

Data prep: Internal data representation

Struct AdcChannelData carries both raw and prepared data

- Between DataPrep tools
- Also provides convenient interface for use analysis
- See dunetpc/dune/DuneInterface/AdcChannelData.h
- One channel per object
- Struct member are listed in table on following page
- Not persistent but could provide a tool to build these from recob::Wire and raw::RawDigit containers

Container AdcChannelDataMap is used for multiple channels

- Type is map<short, AdcChannelData> (index is channel number)
- So tools can act on multiple channels
  - E.g. full detector or one TPC, APA plane, ROP, regulator, ASIC, …

Typedefs are in AdcTypes.h, e.g.

- AdcIndex = unsigned int
- AdcROI = std::pair<AdcIndex, AdcIndex>
- AdcFlag (see later slide)

Internal data representation (2)

AdcChannelData members:

Type	Name	Meaning
unsigned int	channel	Channel number
float	pedestal	Assumed pedestal
vector<short>	raw	Integral raw count for each tick
vector<float>	samples	Floating corrected count for each tick
vector<AdcFlag>	flags	Status for each tick (next page)
vector<bool>	signal	Indicates if each tick holds “signal”
vector<AdcRoi>	rois	ROI representation of “signal”
const raw::RawDigit*	digit	Raw digit from which this data is derived
const recob::Wire*	wire	Wire corresponding to this data
AdcIndex	digitIndex	Index of digit in digit container
AdcIndex	wireIndex	Index of wire in wire container

Internal data representation (3)

AdcFlag specifies the state for each tick

- Type is short
- Intended for use by noise removal, calibration and monitoring tools
- Recognized values are listed in table

Variable name	Value	Meaning
AdcGood	0	OK
AdcUnderflow	1	Raw count is underflow (0)
AdcOverFlow	2	Raw count is overflow (4095)
AdcStuckOff	3	Raw low six bits are all 0
AdcStuckOn	4	Raw low six bit are all 1
AdcSetFixed	5	Corrected count set to fixed value (e.g. 0)
AdcInterpolated	6	Corrected count interpolated from other ticks
AdcExtrapolated	7	Corrected count extrapolated from other ticks

Memory consumption

Transient data has expanded raw and corrected data

- Potentially consuming a large amount of memory
  - (# channels) × (# ticks) × (int, flag, float)
- Convenient for tools—no need to repeatedly find and unpack this data into local structures
- Partly mitigated by correcting float data in place (rather than copying)
- Primary control on memory is to process a subset of channels at once
  - E.g. one TPC, APA or ROP
  - Results from preceding subsets are stored in compressed form
  - Working on HLA to do this for DUNE
    - Muse use APA rather than TPC dues to shared wires
    - Geometry support is not yet complete (https://cdcvs.fnal.gov/redmine/issues/9264)
    - May switch to dedicated tool rather than continue to wait for geometry

New services

See appendix for descriptions of new DataPrep services

- Provide the DataPrep flow
- Exchange and update AdcChannelData objects
- Service interfaces are in dunetpc/dune/DuneInterface/XXXService.h
  - XXXService is interface class name
- Service implementations in dunetpc/dune/DataPrep/Service
  - Service XXXService has header XXXService.h
    - Not needed because access is always via interface?
  - Source file for service is XXXService_service.cc
    - Naming convention required by use of cetbuildtools
- These services duplicate the code available in the old DUNE modules
- I am happy to discuss and or all of these
  - But we might want another talk or meeting
- Now easy to add new options for any of the action steps

Beyond the new services

New structure provides convenient basis for many extensions

- New ideas for signal processing steps, e.g. noise removal
- Parallelism
  - Provide service with parallelization for one or more of the compute-intensive steps
  - E.g. deconvolution or noise removal
- Wirecell
  - Data preparation software developed at BNL outside larsoft
  - Old approach has been to cut and paste the code into multiple modules
    - For different detectors (35-ton, DUNE FD)
    - and experiments (DUNE, uBooNE)
  - New approach is
    - Package wirecell in UPS
    - Add wrappers so wirecell actions appear as services with current interfaces
    - Then possible to use wirecell from the DataPrep module
    - Make that module and many supporting services available to uBooNE

Sharing DataPrep

Like to share the new DataPrep with uBooNE and others

- Move relevant code to larsoft
  - Service interfaces
  - Transient data classes
  - Many of the service implementations

Major motivation is to provide access to wirecell

- This requires the following
  - Above code sharing
  - Package wirecell in UPS
  - Add DataPrep service wrappers for wirecell code
  - Validation and possibly some evolution in the DataPrep design

Summary/conclusions

DetSim and DataPrep modules have been rewritten for DUNE

- Most code moved to services
- Easier to understand
- Easier to replace actions with new implementations (new code)
- New detsim was already part of standard DUNE production

New services can be used outside of art framework

- E.g. ArtServiceHelper to access services in Root or standalone
- Art canvas should enable reading of event data

Make new DetSim and DataPrep available to other experiments

- Move service interfaces, transient data classes and relevant service implementations to larsoft

Will use DataPrep for wirecell integration

- Package wirecell in UPS
- Add DataPrep services that call the wirecell code

Extras

Data prep action implementation services

- StandardRawDigitPrepService (high-level service)
- StandardRawDigitExtractService
- InterpolatingAdcMitigationService
- AdcSuppressSignalFindingService
- Dune35tNoiseRemovalService
- DuneDeconvolutionService
- MedianPedestalEvaluationService
- DuneRoiBuildingService
- StandardAdcWireBuildingService

StandardRawDigitPrepService

Interface: RawDigitPrepService

- Input: Raw digit vector
- Output: AdcChannelDataMap

This high-level service provides the data prep flow

- Calls the other data prep services (via interfaces)
- Configurable via FCL
  - Steps can be skipped
  - Choice of type and configuration for each low-level service

Data prep module is a thin wrapper around service

- Extract raw digits
- Call this service
- Build wire-digit associations
- Record wires and associations in event

Service distinct from module can be used outside art FW

- E.g. a Root event display

StandardRawDigitPrepService (2)

Flow:

- Loop over Raw digits
  - Add AdcChannelData object to output map
  - Fill with RawDigitExtractService
  - Patch bad ticks with AdcMitigationService
  - Find signals with AdcSignalFindingService
    - This is “early signal finding,” , i.e. before noise removal and dconvolution
    - This information may be used during noise removal
- Do noise removal with AdcNoiseRemovalService
- Deconvolute signal
- Adjust pedestals with value obtained from PedestalEvaluationService
- Build ROIs (final signal finding)
  - Issue: ROI building requires signal finding on deconvoluted data. It would be nice to have tools (rather than just services) so the same interface but different implementations can be used here and for early signal finding
- Build wires

StandardRawDigitPrepService (3)

Configuration:

- Note: Other services follow the same convention for LogLevel.
- In a world with tools, replace the DoXXX flags with tool names.

Type	Name	Meaning
int	LogLevel	0: No log messages 1: Log messages only during initialization 2+: Log messages for every event
bool	DoMitigation	Patch bad ticks
bool	DoEarlySignalFinding	Do early signal finding
bool	DoNoiseRemoval	Do noise removal
bool	DoDeconvolution	Deconvolute signal
bool	DoPedestalAdjustment	Do pedestal adjustment
bool	DoROI	Build ROIs
bool	DoWires	Build wires

StandardRawDigitExtractService

Interface: RawDigitExtractService

- Input: const RawDigit&
- Output: Data for AdcChannelData
  - channel, pedestal, raw, samples, flags

Table gives the configuration parameters

Type	Name	Meaning
int	LogLevel	Per convention
int	PedestalOption	1: Take pedestal from digit 2: Take pedestal from pedestal provider
bool	FlagStuckOff	Set flag if bits are stuck low
bool	FlagStuckOn	Set flag if bits are stuck high

InterpolatingAdcMitigationService

Interface: AdcMitigationService

- Input/output: AdcChannelData

Algorithm

- Samples with stuck bits (and optionally under/overflows) are updated with values obtained by linear interpolation between nearest good neighbors.

Configuration:

Type	Name	Meaning
int	LogLevel	Per convention
bool	SkipUnderflows	If true, underflows are not updated
bool	SkipOverflows	if true, overflows are not updated
int	MaxConsecutiveSamples	Maximum # consecutive samples to update
int	MaxConsecutiveFlag	Action for too many consecutive samples: 1: Leave sample unchanged. 2: Set sample to zero.

AdcSuppressSignalFindingService

Interface: AdcSignalFindingService

- Input/output: AdcChannelData&

Algorithm:

- Uses the AdcSuppressService interface to re-use the signal-finding suppression services developed for DetSim
  - Legacy35tZeroSuppressService – The code from the old DetSim
  - Dune35tZeroSuppressService – Simulation of the DUNE 35t algorithm

No configuration parameters

- Service is discovered by it type name
- Add name when service becomes a tool

Dune35tNoiseRemovalService

Interface: AdcNoiseRemovalService

- Input/output: AdcChannelDataMap&

Algorithm:

- Channels are organized into groups
  - Options to do this by regulator or ASIC
  - Also separate groups for the three wire orientations
- The BG is estimated as median of all signals for each tick and channel
- This value is subtracted from all corresponding signals
- Same algorithm and groups as the old FilterWF service
  - Used in 35t data production
- Service is specific to 35t because it uses 35-ton channel map interface
  - That provides regulator and ASIC grouping
  - Could be generalized to other detectors

Dune35tNoiseRemovalService (2)

Configuration:

Type	Name	Meaning
int	LogLevel	Per convention
int	GroupingFlag	1: By regulator (128 channels) 2: By asic (32 channels)
bool	SkipStuckCodes	Ignore channels with stuck bits in BG estimate
bool	SkipSignals	Channels identified as “signal” are not used in BG est.
bool	CorrectStuckCodes	If true, samples with stuck codes are corrected
int	ShowGroups	If nonzero, channel grouping is displayed in log 1: Organize by orientation and then group 2: Organize by group and then orientation
int	ShowGroupsChannel	Channel type used when groups are show: 0: none 1: online 2: offline, 3,4: offline by name (z1, z2 or z, Z)

DuneDeconvolutionService

Interface: AdcDeconvolutionService

- Input/output: AdcChannelData&

Algorithm:

- Uses SignalShapingServiceDUNE which folds and unfolds the signal with one provided in a histogram
  - Service is found with the usual service handle
  - Different signals for each view (z, u, v)

Configuration:

Type	Name	Meaning
int	LogLevel	Per convention

MedianPedestalEvaluationService

Interface: PedestalEvaluationService

- Input: const AdcChannelData&
- Output: 4 float*: pedestal, rms and error for each

Algorithm:

- Evaluates pedestal as median value in input samples
- Options to omit samples flagged (under/overflow, stuck bits, …) or identified as signal

Configuration:

Type	Name	Meaning
int	LogLevel	Per convention
bool	SkipFlaggedSamples	If true, flagged samples are omitted
bool	SkipSignals	if true, sample identified as signal are not used

DuneRoiBuildingService

Interface: AdcRoiBuildingService

- Input/output: AdcChannelData&

Algorithm

- Copy of the existing 35-ton ROI builder
  - Look for a tick above high threshold
    - Threshold in sigma from signal shaping service
  - Keep it all following ticks until signal falls below some threshold
  - Add padding below and above each selected region

Configuration:

Type	Name	Meaning
int	LogLevel	Per convention
float	NSigmaStart	High threshold
float	NSigmaStop	Low threshold
bool	PadLow	Number of ticks for padding below
bool	PadHigh	Number of ticks for padding above

StandardAdcWireBuildingService

Interface: AdcWireBuildingService

Algorithm:

- Construct wires (recob::Wire) from channel data

Configuration

Type	Name	Meaning
int	LogLevel	Per convention