Models for the access to pedestals of wire signal

Gianluca Petrillo, Saba Sehrish, Erica Snider

University of Rochester/Fermilab

LArSoft Architecture Review Meeting, July 15th , 2015





I have heard of two categories of pedestal computation:

online relative to LArSoft, i.e. pedestals are already known and final when creating raw::RawDigit's the first time

- read from the configuration
- extracted from the raw digit itself

offline meaning that it's not already in the first art event

- obtained from dedicated runs
- extracted after the fact

We need:

a common, transparent interface:
 "here are event and channel IDs, give me the pedestal"

implementations able to support both models

From LArSoft point of view, pedestals can come from:

- - another data product amending that information

algorithm computing them on the spot

- service accessing a data base
 - reading a simple configuration (e.g. FHiCL)

A proposal

My proposal consists of:

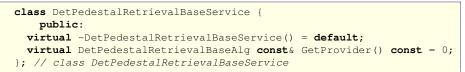
- \Rightarrow an abstract service interface
- a provided implementation reading pedestal from
 raw::RawDigit
- ⇒ a provided implementation using PedestalRetrievalAlg (database access)
 - if needed, another implementation can be written to read the pedestals from a new, specific data product
 - \rightarrow saves from reading the raw digits, and allows for recomputing
 - if an algorithmic approach is required, that should be turned into a module producing a data product, and use the previous approach

Comments?

- does this satisfy all current needs?
- is this flexible enough to accommodate any foreseeable need?

A proposal: interface

Each framework will implement concrete services. In art, we will have an abstract service interface:



Listing 1: Pedestal retrieval service interface

The service provider interface might reflect:

```
class DetPedestalRetrievalBaseAlg {
    public:
    virtual ~DetPedestalRetrievalBaseAlg() = default;
    virtual float PedMean(raw::ChannelID_t ch) const = 0;
    virtual float PedRms(raw::ChannelID_t ch) const = 0;
    virtual float PedMeanErr(raw::ChannelID_t ch) const = 0;
    virtual float PedRmsErr(raw::ChannelID_t ch) const = 0;
    virtual float PedRmsErr(raw::ChannelID_t ch) const = 0;
    virtual DetPedestal const& Pedestal(raw::ChannelID_t ch) const;
}; // class DetPedestalRetrievalBaseAlg
```

Listing 2: Service provider interface

Example implementation of art service

Implementations must take care of updating the status of the service provider. For example, a database-based implementation might show:

```
class DetPedestalRetrievalDBService: public DetPedestalRetrievalBaseService
  std::unique ptr<DetPedestalRetrievalDBAlg> algo;
 void Update(art::Event const& evt)
    { algo->Update(lariov::ExtractIOVfromEvent(evt)); }
   public:
 Det Pedestal Retrieval DBService
    (fhicl::ParameterSet const pset, art::ActivityRegistry& reg):
    algo (new DetPedestalRetrievalBaseService (pset))
      reg.sPreProcessEvent.watch
        (this, &DetPedestalRetrievalDBService::Update);
 virtual DetPedestalRetrievalBaseAlg const (GetProvider() const overric
    { return *(algo.get()); }
}; // class DetPedestalRetrievalDBService
         Listing 3: Example of pedestal retrieval art service implementation
```

Example implementation of service provider

The service provider interface might reflect:

```
class DetPedestalRetrievalDBAlg: public DetPedestalRetrievalBaseAlg {
   public:
 DetPedestalRetrievalDBAlg(fhicl::ParameterSet const& pset);
 virtual float PedMean(raw::ChannelID t ch) const override
    { return Pedestal(ch).PedMean(); }
 virtual float PedRms(raw::ChannelID t ch) const override
    { return Pedestal(ch).PedRms(): }
 virtual float PedMeanErr(raw::ChannelID t ch) const override
    { return Pedestal(ch).PedMeanErr(); }
 virtual float PedRmsErr(raw::ChannelID t ch) const override
    { return Pedestal(ch).PedRmsErr(); }
  /// Fetch all the channel data at once
 virtual DetPedestal const @ Pedestal (raw::ChannelID_t ch) const overric
  /// Update according to the current interval of validity
 void Update(lariov::IOVTimeStamp const& iov);
}; // class DetPedestalRetrievalBaseAlg
```

Listing 4: Example of service provider implementation

These examples are heavily inspired by the current implementation by Brandon Eberly. But details differ:

- using raw::ChannelID_t instead of unsigned int
- accessors are constant (might be less than trivial due to caching)
- although each service provider will know how to react to an update request, that request is not part of the abstract interface
 ⇒ framework modules can't control update
- provider's Update() does not accept art::Event

I need also to talk to him before I attempt any change.