Accessing service providers

Gianluca Petrillo

University of Rochester/Fermilab

LArSoft architecture review meeting, August 12th , 2015

LArSoft service model

We pursue a two-layer model:

service provider is independent of the framework and provides the service

art service interface coordinates the provider with the framework and delivers it to the users

- users (and especially, algorithms) communicate only through the service provider interface
- framework modules are in charge of informing the algorithm about the provider

This allows testing and execution of the algorithms (and services) independently from the sorrounding framework.

How does it look like from the user

In a module, get *art* service and ask it for service provider:

```
geo::GeometryCore const& geom = *(art::ServiceHandle<geo::Geometry>());
```

Listing 1: `Geometry` *service provider*

```
util::SimpleTimeService const& timeSrv
  = &*art::ServiceHandle<util::TimeService>();
```

Listing 2: `TimeService` *service provider*

```
filter::ChannelFilterProvider const& chanFilt
  = art::ServiceHandle<filter::ChannelFilterService>()->GetProvider();
```

Listing 3: `ChannelFilterService` *service provider*

```
lariov::IDetPedestalProvider const& pedestalRetrieval
  = art::ServiceHandle<lariov::IDetPedestalService>()
    ->GetPedestalProvider();
```

Listing 4: `IDetPedestalService` *service provider*

```
util::DetectorProperties const& detProp
  = &*art::ServiceHandle<util::DetectorProperties>();
```

Listing 5: `DetectorProperties` *service (no splitting yet)*

Discussion topics

- the model itself (Jim K. asked about it on the last meeting)
- the way to access the provider
  - should be easy to convince you that a uniform approach is preferable...
  - `Geometry` service is so widely used that it seemed unwise to force a interface change
  - also from the last meeting: Brian R. was qualifying the syntax as “confusing”
  - possibly allowing also for alternative specific ones?
  - either way: which way to go?

Additional material

What I like

- implementing two options:

```
1 lariov::IDetPedestalProvider const& pedestalRetrieval
2   = art::ServiceHandle<lariov::IDetPedestalService>()
3     ->GetProvider();
4 lariov::IDetPedestalProvider const* pPedestalRetrieval
5   = art::ServiceHandle<lariov::IDetPedestalService>()
6     ->GetProviderPtr();
```

Listing 6: `IDetPedestalService` *service provider*

(should the second return a shared pointer? I'd think not)

What I like *(cont'd)*

- I like implicit conversion so and so:
  - – compilers tend to take the initiative and convert where users don't expect
  - \+ but, *art* services have a very limited use, it's hard to think how this could go wrong
  - – implicit conversion would allow algorithms to use *art* service instead of the provider:

```
1  lariov::IDetPedestalProvider const* pPedRetr
2    = art::ServiceHandle<lariov::IDetPedestalService>()
3      ->GetProviderPtr();
4  float mean1 = pPedRetr->PedMean(channel);  // fine
5
6  lariov::IDetPedestalService const* pPedRetrService
7    = &*art::ServiceHandle<lariov::IDetPedestalService>();
8  float mean2 = pPedRetrService->PedMean(channel);  // not fine
```

Listing 7: The issue with implicit conversion