Making `geo::Geometry` (and other services) thread-safe

Kyle J. Knoepfel

25 February 2020

LArSoft coordination meeting

Making services thread-safe

- Service *scope* definitions
  - `LEGACY`: service that can be used with only one schedule and only one thread configured
  - `SHARED`: service that can be used with *n* schedules and *m* threads

Making services thread-safe

- Service *scope* definitions
  - `LEGACY`: service that can be used with only one schedule and only one thread configured
  - `SHARED`: service that can be used with *n* schedules and *m* threads

- Several impediments to making services thread-safe:
  - Many services have state that is updated throughout processing
    - (e.g.) “current event” data, has no meaning when multiple events are being processed at the same time
  - LArSoft uses many services that are polymorphic
    - Until now, *art* has required any `SHARED` service interface to have a matching `SHARED` service implementation.
    - This means that all implementations of a service interface must be thread-safe.
  - Limited effort

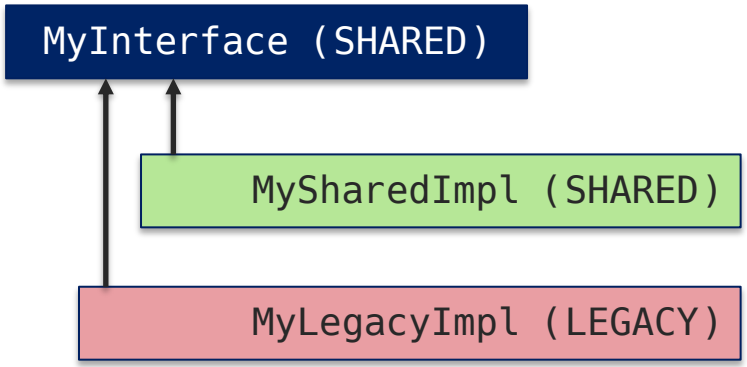
Making services thread-safe

- Service *scope* definitions
  - LEGACY: service that can be used with only one schedule and only one thread configured
  - SHARED: service that can be used with *n* schedules and *m* threads
- Several impediments to making services thread-safe:
  - Many services have state that is updated throughout processing
    - (e.g.) “current event” data, has no meaning when multiple events are being processed at the same time
  - LArSoft uses many services that are polymorphic
    - Until now, *art* has required any SHARED service interface to have a matching SHARED service implementation.
    - This means that all implementations of a service interface must be thread-safe.
  - Limited effort

Today, I will discuss ways to move forward on thread-safety.

art 3.05 to be released this week

- Same product stack as before
- Still supports macOS Mojave (SIP disabled)
- (Nearly) decouples service interface scope from implementation scope
  - LEGACY service interfaces must have LEGACY implementations
  - SHARED service interfaces may have either SHARED or LEGACY implementations
- One implementation's thread-safety has no bearing on another's



`lar::EnsureOnlyOneSchedule`

- A PR for this week's release includes a new class that ensures only one art schedule has been configured for the job

lar::EnsureOnlyOneSchedule

- A PR for this week's release includes a new class that ensures only one art schedule has been configured for the job

```
class MyLegacyImpl : public MyInterface {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, LEGACY)
```

1 schedule and
1 thread

```
class MySharedImpl : public MyInterface {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, SHARED)
```

n schedules and
m threads

lar::EnsureOnlyOneSchedule

- A PR for this week's release includes a new class that ensures only one art schedule has been configured for the job

```
class MyLegacyImpl : public MyInterface {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, LEGACY)
```

1 schedule and
1 thread

Suppose a service cannot support concurrent events, but it can support multiple threads

```
class MySharedImpl : public MyInterface {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, SHARED)
```

n schedules and
m threads

lar::EnsureOnlyOneSchedule

- A PR for this week's release includes a new class that ensures only one art schedule has been configured for the job

```
class MyLegacyImpl : public MyInterface {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, LEGACY)
```

1 schedule and
1 thread

```
class MySharedImpl : public MyInterface,
                     private lar::EnsureOnlyOneSchedule {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, SHARED)
```

1 schedule and
m threads

```
class MySharedImpl : public MyInterface {
};

DECLARE_ART_SERVICE_INTERFACE_IMPL(MyLegacyImpl, MyInterface, SHARED)
```

n schedules and
m threads

Geometry service (and friends)

- Broadly speaking, the geometry system is thread-safe within a run
  - Will not discuss the issue of run-dependent geometries here

Geometry service (and friends)

- Broadly speaking, the geometry system is thread-safe within a run
  - Will not discuss the issue of run-dependent geometries here

- There are some entanglements with the `ExptGeoHelperInterface` interface
  - Implementations of `ExptGeoHelperInterface` have been allowed to adjust the `geo::GeometryCore` object.
  - The interface exposed by `ExptGeoHelperInterface` can, in principle, be called anywhere, which is the primary thread-safety issue.

- The changes implemented in my PRs remove the entanglement of `geo::GeometryCore` with `ExptGeoHelperInterface`.

ExptGeoHelperInterface changes

```
class ExptGeoHelperInterface {
public:
  using ChannelMapAlgPtr_t = std::shared_ptr<const ChannelMapAlg>;

  virtual ~ExptGeoHelperInterface() = default;

  void
  ConfigureChannelMapAlg(fhicl::ParameterSet const& sortingParameters,
                         geo::GeometryCore* geom);

  ChannelMapAlgPtr_t GetChannelMapAlg() const;

private:

  virtual
  void
  doConfigureChannelMapAlg(fhicl::ParameterSet const& sortingParameters,
                           geo::GeometryCore* geom) = 0;

  virtual
  ChannelMapAlgPtr_t
  doGetChannelMapAlg() const = 0;

};
```

Before PR

GeoHelper directly adjusts the GeometryCode while still co-owning the channel map algorithm.

Configure function is not ‘const’, and it is accessible from anywhere via a handle.

`ExptGeoHelperInterface` changes

```
class ExptGeoHelperInterface {
public:
  using ChannelMapAlgPtr_t = std::unique_ptr<ChannelMapAlg>;

  virtual ~ExptGeoHelperInterface() = default;

  ChannelMapAlgPtr_t
  ConfigureChannelMapAlg(fhicl::ParameterSet const& sortingParameters,
                         std::string const& detectorName) const;

private:

  virtual
  ChannelMapAlgPtr_t
  doConfigureChannelMapAlg(fhicl::ParameterSet const& sortingParameters,
                           std::string const& detectorName) const = 0;
  };
```

After PR

GeoHelper calculates and returns the map algorithm.

Interface is 'const'.

Only the detector name is passed to the function.

Ownership of the channel map algorithm is no longer shared but belongs to the caller.

`ExptGeoHelperInterface` usage changes in Geometry service

```
103       -     art::ServiceHandle<geo::ExptGeoHelperInterface>()
104       -     ->ConfigureChannelMapAlg(fSortingParameters, this);
105       -
106       -     if ( ! ChannelMap() ) {
      104 +     art::ServiceHandle<geo::ExptGeoHelperInterface const> helper{};
      105 +     auto channelMapAlg = helper->ConfigureChannelMapAlg(fSortingParameters,
      106 +                                                         DetectorName());
      107 +     if (!channelMapAlg) {
107   108           throw cet::exception("ChannelMapLoadFail")
108   109             << " failed to load new channel map";
109   110       }
110       -
      111 +     ApplyChannelMap(move(channelMapAlg));
```

Similar changes made for `AuxDetExptGeoHelperInterface` and its use.

Other simplifications

- `GeoObjectSorter` (and related code) no longer relies on geometry collections to own by pointer instead of by value:

```
31  -     virtual void SortAuxDets        (std::vector<geo::AuxDetGeo*>          & adgeo)   const = 0;
32  -     virtual void SortAuxDetSensitive(std::vector<geo::AuxDetSensitiveGeo*> & adsgeo)  const = 0;
33  -     virtual void SortCryostats      (std::vector<geo::CryostatGeo*>        & cgeo)    const = 0;
34  -     virtual void SortTPCs                (std::vector<geo::TPCGeo*>            & tgeo)     const = 0;
35  -     virtual void SortPlanes              (std::vector<geo::PlaneGeo*>          & pgeo,
36  -                                       geo::DriftDirection_t          const & driftDir) const = 0;
37  -     virtual void SortWires               (std::vector<geo::WireGeo*>           & wgeo)     const = 0;
38  -     virtual void SortOpDets         (std::vector<geo::OpDetGeo*>          & opdet) const;
39  -   private:
40  -
29  +     virtual void SortAuxDets(std::vector<geo::AuxDetGeo>& adgeo) const = 0;
30  +     virtual void SortAuxDetSensitive(std::vector<geo::AuxDetSensitiveGeo>& adsgeo) const = 0;
31  +     virtual void SortCryostats(std::vector<geo::CryostatGeo>& cgeo) const = 0;
32  +     virtual void SortTPCs(std::vector<geo::TPCGeo>& tgeo) const = 0;
33  +     virtual void SortPlanes(std::vector<geo::PlaneGeo>& pgeo,
34  +                             geo::DriftDirection_t driftDir) const = 0;
35  +     virtual void SortWires(std::vector<geo::WireGeo>& wgeo) const = 0;
36  +     virtual void SortOpDets(std::vector<geo::OpDetGeo>& opdet) const;
```

Other changes

- It is possible that the changes on the previous page will conflict with some pixel-geometry efforts Gianluca is working on.
  - I propose adopting the changes on the previous page, and then I will work with Gianluca to adjust the interface once we determine a change is necessary.

- `Geometry` and `AuxDetGeometry` service callbacks now private.
- All `(AuxDet)Geometry` and `(AuxDet)ExptGeoHelperInterface` services have been marked `SHARED`.
- Any code that creates service handles to these classes must link against `${ART_UTILITIES}`

- Pull requests here:
  - https://github.com/LArSoft/larcorealg/pull/3
  - https://github.com/LArSoft/larcore/pull/3

Feature branches for the experiments

- dunetpc:feature/knoepfel_threadsafe_geometry
- icaruscode:feature/knoepfel_threadsafe_geometry
- lariatsoft:feature/knoepfel_threadsafe_geometry
- sbndcode:feature/knoepfel_threadsafe_geometry
- ubcore:feature/knoepfel_threadsafe_geometry
- ubcrt:/feature/knoepfel_threadsafe_geometry

Timescale for other services

- Once *art* 3.05 is adopted, changing service interface scopes to SHARED will be quick (for places where it makes sense).
- For services that depend on 'current' events, changing the scope to SHARED and inheriting from `lar::EnsureOnlyOneSchedule` can also happen relatively quickly (e.g. `DetectorClocks`, `DetectorProperties`, etc.)
- We are actively working on how to make the following services thread-safe:
  - DetectorClocks (https://indico.fnal.gov/event/22735/contribution/1/material/slides/0.pdf)
  - ChannelStatus (will likely be a "producing service" as supported by *art*)

- General guidance:
  - Service implementations generally should not have header files
  - Reduce mutable state and the number of side effects
    - Avoid functions that do not return anything