Tool/service software model

DUNE Software and Computing

David Adams

BNL

December 15, 2015

Introduction

I will discuss tool/service model for software

- Based on experiences in D0, ATLAS and DUNE

Outline

- Goals
- Art framework
- Terminology: utility, tool, service
- Tools and services
- Connection to the framework
- Implementation
- Path forward
- DetSimDUNE35t as concrete example

Goals

Comprehensibility (make SW easy to understand)

- Non-expert user should be able to look at job configuration and understand what is done and what parameters are used
  - And then modify actions or parameters for a subsequent run
- Easy to find the code corresponding to a give action in the configuration
  - And then easy to understand what that code is doing

Modularity

- Make it easy to replace an action with an alternative implementation
  - At natural and fine granularity so user does not have to cut and paste a lot of irrelevant code
  - Ideal development environment would not require user to check out and build code other than his or her own

Portability (use SW in other frameworks)

- Convenient if code and configuration can be used in other frameworks
  - Standalone main or root scripts for analysis
  - Other full frameworks to allow for future migration of the production FW

Art framework

DUNE and many other experiments use the art framework

- Modularity provided by producers and services
  - Both configured at initialization time by art from fcl
- Producer is a class
  - art loops over events and calls each producer once for each event
  - Producer has means to retrieve data from the current event
  - And write data to that event
  - It would not make much sense to use a producer outside the art FW as one would just be duplicating art
  - (art also supports analyzers that are similar but do not write data)
- Service is a class
  - Called by producers or other services
  - Accessed with handle specifying type (either interface of full type)
    - Only instance is available for each type
  - It is possible to use services and (service handles) outside the art FW
    - Still depend on art libraries for configuration and handle
    - See https://github.com/dladams/art_extensions

Terminology

Utility

- Class that carries our some action, i.e. provides one or methods
- Configuration typically specified by ctor arguments
- Often stateless after configuration
  - Action methods are const

Tool is like a utility except

- Configuration taken from global job configuration
  - E.g. a fcl file
- Means provided to access any configured instance by name
  - E.g. a tool handle: ToolHandle<SomeType>("SomeName")
  - Type name can be that of a Tool interface (w/ multiple implementations)

Service is a tool except

- Only one instance is accessible (singleton)
  - So name is not needed in handle
    - ServiceHandle<SomeType> () or ToolHandle<SomeType>("")

Art supports services but not tools

Tools and services

Tools and services help to meet out goals

- Mapping between configuration and action in code
- Nested structure (tools call other tools)
  - Easy to view or modify at different levels of granularity
- Tool interface
  - Makes it easy to replace one implementation with another
    - At run time
    - Can provide new implementation in a user library
- No connection (yet) to FW
  - Means tools can be used in other frameworks
  - For art services, only true if the service does not register for callbacks

Tools vs. services

- In the above, tool can be replaced with “tool and service”
- Services are sufficient if we never want in the same job:
  - Two implementations of the same interface
  - Two configurations of any implementation

Connection to the framework

To run production jobs, we do need to connect to the FW

- For art, this means to define producers
- Like to have access to producer granularity outside of art
- Can accomplish this by defining a tool implementation for each desired producer
    - art producer is a thin wrapper that calls this tool
    - Outside art, user would call tool directly
        - User might pass and receive the relevant event data
        - Probably want an event data service for this

Implementation

Use of tools does not ensure modularity and comprehensibility

- Must also do work to define appropriate SW architecture

For modularity, identify and define appropriate interfaces

- If done well, later implementations of a tool will not change interface
- Will truly be able to plug in a tool from a user repository

Comprehensibility requires attention to configuration and code

- Intuitive naming of tool instances
  - In art, prolog allows multiple named instances of a given service
    - Only one instance can be used in the job
    - And name does not survive when fcl is resolved (bad for comprehensibility)

Nested structure

- I.e. tools use other tools
- So comprehension and extension can be done at different levels
- Important for both modularity and comprehensibility

Path forward

We might move forward as follows

- Note we can stop at any point (including our current state)

1. Move code from producers to services
   - Identify appropriate nested structure
   - Identify and implement service interfaces
     - Consider other producers which may use the same interface
   - Implement services copying code from producer
   - Replace producer code with service calls
   - Ensure fcl for service is comprehensible
2. Create producer service
   - Move remaining producer code to a service
   - Replace producer code with a call to this service
     - E.g. passing and receiving event data
   - After this, non-art user can also create data with producer service
3. Replace services with tools where appropriate (see next)

Tools

Services compared to tools

- Service has one instance (singleton)
- Tool can have multiple named instances
  - Name is a label in configuration used in code to access the tool instance.

Art services can act as tools

- As long as we only use one instance in the code
- Multiple instances "exist" only in fcl prolog
- Must map prolog instance to (interface) type name in fcl

Would be nice to have real tools

- Instance names would appear in resolved fcl
  - Not just names in prolog (invisible after fcl resolution)
  - Get rid of ugly "@local::" in fcl
- Could directly compare action of one tool instance to another
- Could use different tools at different stages of reco
  - E.g. ideal and non-deal geometry or calibration for sim and reco

Example: DetSimDUNE35t module

I have been working on DUNE DetSim

- Step 1 described earlier
- DetSimDUNE35t → DetSimDUNE and many services
  - See https://github.com/dladams/dunezs
- Motivation was the desire to add an alternative zero suppression
  - Also like to add new noise simulation based on 35t data

Following slides

- Summarize the work being done
- Provide an example of step 1 of the proposed path forward
- Presented in 35t sim/reco meeting last week

DetSimDUNE35t module issues

Monolithic code

- Most of the code is inside the module (> 1500 lines)
- Exception is ZS/compression is partly in raw.cxx
  - But nearest neighbor (bulk of code) is in module
- Adding new options (e.g. for noise or ZS) means making this even bigger (and more difficult to understand)

Monolithic fcl

- Fcl is block of many parameters
- Often not obvious which influence which stage of processing
- Many irrelevant because they are for an option not selected

Duplicated code

- Much of code is 35t-specific and so module is duplicated for FD

This is not intended as criticism of development path

- Natural to put code directly in an initially simple module
- And then see this grow as features are added
- But now time to split this up

Old module fcl parameters

daq: {
CollectionCalibPed: 500
CollectionCalibPedRMS: 0.01
CollectionPed: 500
CollectionPedRMS: 0.01
CompressionType: "ZeroSuppression"
DriftEModuleLabel: "largeant"
FractHorizGapUCollect: 0.1
FractHorizGapUMiss: 0.8
FractHorizGapVCollect: 0.1
FractHorizGapVMiss: 0.8
FractHorizGapZMiss: 0.8
FractUUCollect: 0.5
FractUUMiss: 0.2
FractUVCollect: 0.1
FractUVMiss: 0.2
FractVUCollect: 0.5
FractVUMiss: 0.2
FractVVCollect: 0.1
FractVVMiss: 0.2
FractVertGapUCollect: 0.1
FractVertGapUMiss: 0.8
FractVertGapVCollect: 0.1
FractVertGapVMiss: 0.8
FractVertGapZMiss: 0.8
FractZUMiss: 0.2
FractZVMiss: 0.2
InductionCalibPed: 1800
InductionCalibPedRMS: 0.01
InductionPed: 1800
InductionPedRMS: 0.01
LowCutoffU: 7.5
LowCutoffV: 7.5
LowCutoffZ: 7.5
NearestNeighbor: 25
~~NeighboringChannels: 3~~
NoiseArrayPoints: 1000
NoiseFactU: 0.05
NoiseFactV: 0.05
NoiseFactZ: 0.05
NoiseModel: 1
NoiseWidthU: 2000
NoiseWidthV: 2000
NoiseWidthZ: 2000
PedestalOn: "false"
SaveEmptyChannel: "true"
SimCombs: "false"
SimStuckBits: "false"
StuckBitsOverflowProbHistoName: "pCorrFracOverflowVsInputLsbCell"
StuckBitsProbabilitiesFname: "ADCStuckCodeProbabilities35t/output_produceDcScanSummaryPlots_20150827_coldTest_0p1to1p4_step0p0010.root"
StuckBitsUnderflowProbHistoName: "pCorrFracUnderflowVsInputLsbCell"
~~ZeroThreshold: 5~~
module_type: "SimWireDUNE”
}

DetSim module flow

The old module is DetSimDUNE35t

- Input is SimChannel
  - GEANT energy deposit for each channel and particle
  - Includes drift attenuation and diffusion
- Output is raw data in LArSoft format
  - ADC count and assigned threshold for each channel
  - Flag indicating if and how the ADC data is zero-suppressed and/or compressed
- Calculations take place in loop over channels
- Output product is constructed

The new module is DetSimDUNE

- Same except calculations mostly done with services
- And a few minor mods

Old DetSim module channel loop

Calculations in loop over channels

- ADC signal is extracted from SimChannel for the channel
  - There is an option to create extra signal for the "combs"
  - Rather complicated; 35 ton specific
- SignalShapingServiceDUNE35t adds electronics response
  - Separate call uses collection response for extra signal
- Noise is calculated
- Combined and converted from floating signal to 12-bit integer count
- Pedestal and pedestal fluctuations are added
  - Pedestals and RMS (for fluctuations) are fcl parameters
- Conversion from floating signal to 12-bit integer count is repeated
- Zero suppression, compression and stuck bits are applied
  - Some of the code is in raw.cxx
  - Complicated ring buffers to allow ZS to take nearby channels into account

New DetSim module channel loop

Service calls in loop over channels

- SimChannelExtractServiceBase extracts signals from SimChannel
  - SimChannelExtractAllService ignore combs
  - SimChannelExtract35tService produces base and extra signal as before
- SignalShapingServiceDUNE35t adds electronics response
  - Separate call uses collection response for extra signal
- Extra signal is added to the base signal
- ChannelNoiseServiceBase is used to add noise to the signal
  - ExponentialChannelNoiseService reproduces old option 1
  - WhiteChannelNoiseService will reproduce option 2
- ~~Combined and converted from floating signal to 12-bit integer count~~
- Pedestal and pedestal fluctuation is added
  - Pedestal values and RMS from IDetPedestalProvider
- Conversion from floating signal to 12-bit integer count
- Add stuck bits (plan to move this to a service)
- ZeroSuppressBase applies zero suppression
- CompressReplaceService compresses (switch to CompressServiceBase)

Top level fcl

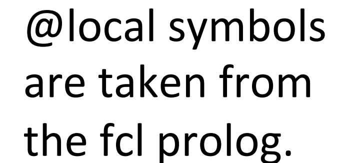
```
# Signal extraction service.
services.user.SimChannelExtractServiceBase: @local::scxall

# Channel noise service.
services.user.ChannelNoiseServiceBase: @local::chnoiseold

# Pedestal service.
services.user.IDetPedestalService: @local::pedfixed

# Zero suppression service.
#services.user.ZeroSuppressBase: @local::zsnone
#services.user.ZeroSuppressBase: @local::zslegacy
services.user.ZeroSuppressBase: @local::zsonline

# Compression service.
services.user.CompressReplaceService: { Zero: 0 }
```



Prolog fcl for SimChannelExtractBase

```
scxall: {
  service_provider: SimChannelExtractAllService
}
```

No combs.

35t combs

```
scx35t: {
  service_provider: SimChannelExtract35tService
  FractHorizGapUCollect: 0.1
  FractHorizGapUMiss: 0.8
  FractHorizGapVCollect: 0.1
  FractHorizGapVMiss: 0.8
  FractHorizGapZMiss: 0.8
  FractUUCollect: 0.5
  FractUUMiss: 0.2
  FractUVCollect: 0.1
  FractUVMiss: 0.2
  FractVUCollect: 0.5
  FractVUMiss: 0.2
  FractVVCollect: 0.1
  FractVVMiss: 0.2
  FractVertGapUCollect: 0.1
  FractVertGapUMiss: 0.8
  FractVertGapVCollect: 0.1
  FractVertGapVMiss: 0.8
  FractVertGapZMiss: 0.8
  FractZUMiss: 0.2
  FractZVMiss: 0.2
}
```

Prolog fcl for ChannelNoiseServiceBase

```
chnoiseold: {
  service_provider: ExponentialChannelNoiseService
  NoiseFactU: 0.05
  NoiseFactV: 0.05
  NoiseFactZ: 0.05
  NoiseWidthU: 2000
  NoiseWidthV: 2000
  NoiseWidthZ: 2000
  LowCutoffU: 7.5
  LowCutoffV: 7.5
  LowCutoffZ: 7.5
  NoiseArrayPoints: 1000
  OldNoiseIndex: true
}
```

Noise model 1

```
chnoisewhite: {
  service_provider: WhiteChannelNoiseService
}
```

Noise model 2

Prolog fcl for pedestal retrieval

```
dbretrieval: {
  AlgName: "DatabaseRetrievalAlg"
  DBFolderName:  ""
  DBUrl: ""
  DBTag: ""
}

pedfixed: {
  service_provider: SIOVDetPedestalService
  DetPedestalRetrievalAlg: {
    AlgName:          "DetPedestalRetrievalAlg"
    DatabaseRetrievalAlg: @local::dbretrieval
    UseDB: false
    UseFile: false
    DefaultCollMean:  500.0
    DefaultCollRms:     0.01
    DefaultIndMean:  1800.0
    DefaultIndRms:      0.01
    DefaultMeanErr:     0.0
    DefaultRmsErr:     0.0
  }
}
```

LarSoft pedestal service configured to used fixed values as in old DetSim

Prolog fcl for ZeroSuppressServiceBase

```
zsnone: {
  service_provider: ZeroSuppressFixedService
}

zslegacy: {
  service_provider:
ZeroSuppress35tLegacyService
  AdcThreshold: 10.0
  TickRange: 10
  MinTickGap: 2
  SuppressStickyBits: true
}

zsonline: {
  service_provider: ZeroSuppress35tService
  NS:  5
  NL: 15
  ND:  5
  NT:  3
  TS:  3
  TL:  7
  TD:  5
}
```

No zero suppression.

Same ZS as in old DetSim

Simulation of online ZS

Summary/conclusions

Like SW to be comprehensible, modular and portable

Code in tools and services can be used to achieve these

Multi-step path to move code from modules to tools/services

- Described here
- Example of first step given for DetSim

Helpful to have support for tools in art