Wire Cell Toolkit
Simulation and Integration

Brett Viren
(for WCT team)

Physics Department

BROOKHAVEN
NATIONAL LABORATORY

μB Ana Tools
2017 June 01

Outline

Documentation

Documentation at `https://wirecell.github.io/`

manual HTML (also PDF, EPUB, Markdown and Org formats)

Internals Doxygen reference.

- Manual is still a work in progress with some gaps.
- But, a lot of coverage already.

Next slides give just the highest level concepts.

Wire – Cell Toolkit

The major design elements of the toolkit are:

- Functionality follows core pattern of **component classes** implementing **abstract interfaces**.
- Components are **dynamically constructed** and may be later obtained by their associated **type** and **name** strings.
- Components may accept **configuration** objects which may be constructed from WCT configuration files.
- Implements **data flow programming** paradigm (optional) to **dynamically aggregate** components into an **execution graph** as driven by user configuration.
- A single, **reference application** (`wire-cell`) exposing the toolkit to the command line.

The rest of WCT is set of "**batteries included**" implementations.

Using WCT from the Command Line

Wire – Cell Toolkit is first and foremost a toolkit, but it provides a command line application.

Eg, to run the WCT simulation:

```
wire-cell -c uboone/fourdee.jsonnet
```

Configuration file format is either "flat" (and optionally compressed) JSON or the more structured Jsonnet data templating language.

`wire-cell` CLI Application Outline

The `wire-cell` **reference application** gives example of minimum needed to embed WCT into any application. It implements no "real" toolkit code.

`main()`:

1. **Load configuration** from file and into object representation.
2. Interpret the config objects to **dynamically construct** all referenced **components** and perform plugin library management.
3. **Apply configuration** to components.
4. Special `IApplication` component(s) assemble other components in some way.
5. Each **app component** has their `execute()` called until all terminate.

WCT Simulation Features

- Field response functions (Yichen)
  - 2D Garfield, 0±10 wires, 1/10 pitch impacts.
  - Includes long-range induction and intra-wire field variations.
  - Fields for nominal bias and crossing/shorted wires.
- "Truth field responses" for validation metrics (Brooke/Hanyu).
- Two noise models:
  1. Empirical MicroBooNE noise paper model (Jyoti/Xin).
  2. First-principle analytic calculation (M.Diwan).
- Misconfigured/dead channel emulation.

Touching Points when using WCT

Using WCT can be conceptualized as calling a configured object providing three things:

- **data** provide input and accept output data objects in WCT's data model.
- **config** configure WCT using WCT's configuration model.
- **exec** run the WCT functionality.

Will go into some details of these categories

Granularity

WCT is a granular component toolkit (not a black box!).
It may be entered at many levels:

concrete implementation classes may be directly instantiated. This is typical in WCT unit tests but limits flexibility and makes maintenance harder for integration (current μB WCT noise filter follows this route).

interface classes are dynamically constructed. This leaves their aggregation up to the integration code and means each functionality needs its own LArSoft module or *art* tool.

application classes handle the aggregation. Integration via this level means only a single LArSoft module or *art* tool is needed for all WCT functionality current and future. Allows largely identical code to run in LArSoft as in stand-alone `wire-cell` which makes testing and development easier.

I wish to pursue integration at the **application** layer.

WCT Simulation Data I/O

For WCT simulation, these are the data interfaces which matter:

- `IDepo` energy deposition with $\vec{r}, t, q$ (# ionized e^-) and optional σ_T, σ_L. (aka ???)
- `ITrace` waveform segment with channel, starting time bin (tick) and array of sample values (either in voltage or ADC) (aka `recob::Wire`)
- `IFrame` collection of traces across channels spanning some time (aka "readout", "event", "trigger").

Depositions

WCT simulation starts with "**depositions**".

- Basically, need `G4Hit` or `G4Step` info.
- WCT can apply **ionization physics** or leave that for the calling application to handle.

How do we get this info from LArSoft?

- Currently, we have hacked version of LArG4 that dumps to JSON file.
- Works fine for small number of events but need something more "larsofty".
- Will look into extracting from `sim_ide`.
- Is there a more direct data product?

Data Sources/Sinks

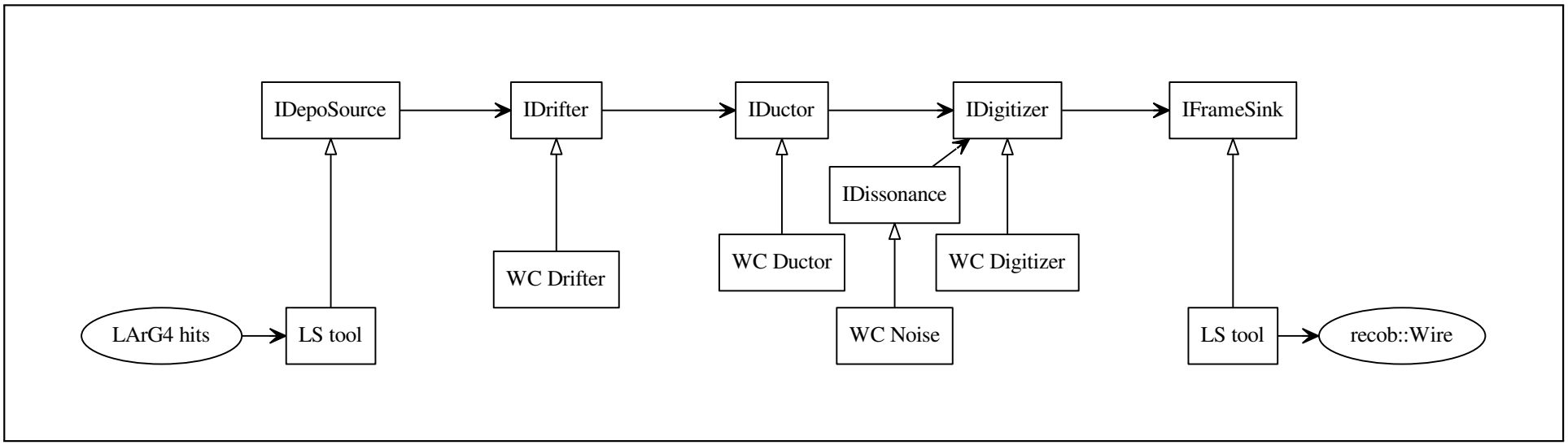
Handling data "touching points" requires implementing:

`IDepoSource` with code that knows how to get depositions out of LArSoft event store and provide them as concrete `IDepo` instances.

`IFrameSink` with code that knows how to accept `IFrame` instances and fill the LArSoft event store with their info converted to `recob::Wire`.

Could explore writing out raw data files too. This would be useful if protoDUNE-like **data reduction** ideas are pursued for μB.

Overall Simulation Integration Concept



WCT `IApplication` implementing graph of simulation components including *art* Tools for source/sink facades to *art* event store.

Note: a more complex graph is ultimately needed to handle

- multiple sets of response functions to simulate touching wires,
- response functions for "truth" metrics or
- potentially producing per-particle frames.

Timeline

1. Complete **simulation** (mostly noise model - going quickly)
2. Port prototype **signal processing** code into WCT (bv/Xin)
3. Integrate **simulation** into LArSoft
4. Integrate **signal processing** into LArSoft

In parallel, working on **MicroBooNE signal processing paper**

- Sim + sigproc needed for this paper

$\rightarrow$ Aiming to have first draft for the next collaboration meeting.