Meld: Exploring the feasibility of a framework-less framework

Kyle J. Knoepfel
11 May 2023
26th International Conference on Computing in High Energy & Nuclear Physics
What type of framework does DUNE need?
What type of framework does DUNE need?

• What DUNE has:
  DUNE’s current framework (art) originates from a collider-physics experiment, steeped in event-based concepts.

• But:
  The “event” is not always a helpful concept for neutrino experiments.

• What DUNE needs…
What type of framework does DUNE need?

• What DUNE has:
  DUNE’s current framework (art) originates from a collider-physics experiment, steeped in event-based concepts.

• But:
  The “event” is not always a helpful concept for neutrino experiments.

• What DUNE needs…

https://doi.org/10.48550/arXiv.2210.15665
Some DUNE framework requirements (paraphrased)
Some DUNE framework requirements (paraphrased)

*Physics algorithms should be framework-agnostic.*
Some DUNE framework requirements (paraphrased)

*Physics algorithms should be framework-agnostic.*

😊 Fine, assuming it’s a requirement for those writing algorithms.
Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

Fine, assuming it’s a requirement for those writing algorithms.

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.
Physics algorithms should be framework-agnostic.

Fine, assuming it’s a requirement for those writing algorithms.

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

Okay, tricky but probably doable.
Some DUNE framework requirements ( paraphrased)

Physics algorithms should be framework-agnostic.

😊 Fine, assuming it’s a requirement for those writing algorithms.

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

😔 Okay, tricky but probably doable.

The framework should support “sliding event windows” to provide “edge effect” coverage for extended time readouts during supernovae events.
Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

Fine, assuming it’s a requirement for those writing algorithms.

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

Okay, tricky but probably doable.

The framework should support “sliding event windows” to provide “edge effect” coverage for extended time readouts during supernovae events.

Cannot take advantage of statistical independence of events, memory issues, etc.
Some DUNE framework requirements (paraphrased)

Physics algorithms should be framework-agnostic.

😊 Fine, assuming it’s a requirement for those writing algorithms.

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

🙁 Okay, tricky but probably doable.

The framework should support “sliding event windows” to provide “edge effect” coverage for extended time readouts during supernovae events.

😢 Cannot take advantage of statistical independence of events, memory issues, etc.

The framework should make minimal assumptions about the data model.
Physics algorithms should be framework-agnostic.

Fine, assuming it’s a requirement for those writing algorithms.

The framework must be able to break apart events into smaller chunks for more granular processing, and then stitch those chunks back together into an event.

Okay, tricky but probably doable.

The framework should support “sliding event windows” to provide “edge effect” coverage for extended time readouts during supernovae events.

Cannot take advantage of statistical independence of events, memory issues, etc.

The framework should make minimal assumptions about the data model.

That sounds like a framework-less framework...
But is it so crazy?

- How many of art’s assumptions can be relaxed/removed to meet DUNE’s needs?
But is it so crazy?

- How many of art’s assumptions can be relaxed/removed to meet DUNE’s needs?
- Asking this question has resulted in a 2-year project called **Meld**, a laboratory-directed R&D project based at Fermilab.
- The goal is to explore options, not necessarily to provide software.
But is it so crazy?

• How many of art’s assumptions can be relaxed/removed to meet DUNE’s needs?
• Asking this question has resulted in a 2-year project called Meld, a laboratory-directed R&D project based at Fermilab.
• The goal is to explore options, not necessarily to provide software.

Meld has been heavily influenced by:

  Regular discussions with DUNE experts
  Existing framework capabilities and limitations
  Functional programming (e.g. Haskell)
  Mathematics (set, graph, and category theory)
But is it so crazy?

- How many of art’s assumptions can be relaxed/removed to meet DUNE’s needs?
- Asking this question has resulted in a 2-year project called Meld, a laboratory-directed R&D project based at Fermilab.
- The goal is to explore options, not necessarily to provide software.

Meld has been heavily influenced by:
- Regular discussions with DUNE experts
- Existing framework capabilities and limitations
- Functional programming (e.g. Haskell)
- Mathematics (set, graph, and category theory)

Prerequisites

- Support user-provided algorithms written in C++20 or newer
- Design for concurrency
- Favor community-provided software
Looking at the data

The following discussion describes a logical organization of data. It does not imply a specific in-memory representation of data.
Looking at the data (set)
Looking at the data (products)
Looking at the data (products)
Looking at the data (products)
Looking at the data (products)
Looking at the data (products)
Looking at the data (products)
Looking at the data (products)
Looking at the data (product mappings)
Looking at the data (product sequences)
Looking at the data (product sequences)
Looking at the data (product sequences)
We can make the following replacement (e.g.):

\[ C = (C)_8 \]

depicting the data products labeled \( c \) from 8 events as a sequence.
Looking at the data (product sequences)
What type of things are we dealing with?

- An operation that converts a sequence of elements \((a)_8\) to a sequence of elements \((b)_8\) of the same length using a function \(f\):
What type of things are we dealing with?

- An operation that converts a sequence of elements \((a)_8\) to a sequence of elements \((b)_8\) of the same length using a function \(f\):

  This is a map or transform.
What type of things are we dealing with?

- An operation that converts a sequence of elements \((a)_8\) to a sequence of elements \((b)_8\) of the same length using a function \(f\):

  This is a map or transform.

- An operation that converts a sequence of elements \((c)_8\) to a shorter sequence of elements \((K)_4\) at a higher level of nesting, using a function \(g_0\):
What type of things are we dealing with?

- An operation that converts a sequence of elements \((a)_8\) to a sequence of elements \((b)_8\) of the same length using a function \(f\):
  
  This is a map or transform.

- An operation that converts a sequence of elements \((c)_8\) to a shorter sequence of elements \((K)_4\) at a higher level of nesting, using a function \(g_0\):
  
  This is a fold or reduction.
What type of things are we dealing with?

- An operation that converts a sequence of elements \((a)_8\) to a sequence of elements \((b)_8\) of the same length using a function \(f\):
  
  This is a map or transform.

- An operation that converts a sequence of elements \((c)_8\) to a shorter sequence of elements \((K)_4\) at a higher level of nesting, using a function \(g_0\):
  
  This is a fold or reduction.

- An operation that pairs element of two sequences \((J)_4\) and \((K)_4\) into one sequence \((J,K)_4\):
What type of things are we dealing with?

- An operation that converts a sequence of elements \((a)_8\) to a sequence of elements \((b)_8\) of the same length using a function \(f\):

  This is a map or transform.

- An operation that converts a sequence of elements \((c)_8\) to a shorter sequence of elements \((K)_4\) at a higher level of nesting, using a function \(g_0\):

  This is a fold or reduction.

- An operation that pairs element of two sequences \((J)_4\) and \((K)_4\) into one sequence \((J,K)_4\):

  This is a zip.
What type of things are we dealing with?

• An operation that converts a sequence of elements $a_8$ to a sequence of elements $b_8$ of the same length using a function $f$:
  
  This is a map or transform.

• An operation that converts a sequence of elements $c_8$ to a shorter sequence of elements $K_0$ at a higher level of nesting, using a function $g_0$:
  
  This is a fold or reduction.

• An operation that pairs elements of two sequences $J_4$ and $K_4$ into one sequence $J_4, K_4$:
  
  This is a zip.

These have to do with higher-order functions.
Graph of data-product sequences

$$\mathcal{R}$$

<table>
<thead>
<tr>
<th>View</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-centric</td>
<td>Data products</td>
<td>Mappings</td>
</tr>
</tbody>
</table>

This work
Graph of data-product sequences

<table>
<thead>
<tr>
<th>View</th>
<th>Nodes</th>
<th>Edges</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-centric</td>
<td>Data products</td>
<td>Mappings</td>
<td>This work</td>
</tr>
<tr>
<td>Map-centric</td>
<td>Mappings</td>
<td>Data products</td>
<td>More common</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
(a)_8 \quad (c)_8 \\
(b)_8 \quad g_0 \\
(J)_4 \quad (K)_4 \quad (J, K)_4 \\
(W)_1 \quad h_0
\end{align*}
\]
The user specifications are the same with either view:

- Which data products to process
- The data set(s) that contain those products (event, etc.)
- Which higher-order function to use (transform, etc.)
- Which user-defined function to serve as the operator to the higher-order function.
- Allowed concurrency of each function.

The focus is just different.
How are data products and their mappings supported now?

With art, users do not transparently interact with data products. They instead:

- Implement functions based on datasets (e.g. event)
- “Open” the dataset to retrieve and insert products
How are data products and their mappings supported now?

With art, users do not transparently interact with data products. They instead:

- Implement functions based on datasets (e.g. event)
- “Open” the dataset to retrieve and insert products

Some of this is historical and due to:

- The object-oriented nature of the framework.
- Technical limitations of C++ whenever the framework was designed.
How are data products and their mappings supported now?

With art, users do not transparently interact with data products. They instead:

- Implement functions based on datasets (e.g. event)
- “Open” the dataset to retrieve and insert products

Some of this is historical and due to:

- The object-oriented nature of the framework.
- Technical limitations of C++ whenever the framework was designed.

*Results in a lot of software mechanics...*
Example

- Create tracks from hits for each event.
Example

- Create tracks from hits for each event.

\[
\begin{align*}
E_n & \quad f \\
(a)_8 & \quad (b)_8
\end{align*}
\]

\[
\begin{align*}
E_n & \quad \text{make\_tracks} \\
(\text{GoodHits})_8 & \quad (\text{GoodTracks})_8
\end{align*}
\]

Tracks make_tracks(Hits const& hits) { ... }
• Create tracks from hits for each event.

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

namespace {
  Tracks make_tracks(Hits const& hits) { ... }
}

namespace expt {
  class TrackMaker : public art::SharedProducer {
  public:
    TrackMaker(fhicl::ParameterSet const&) :
      { consumes<Hits, art::InEvent>("GoodHits");
        produces<Tracks, art::InEvent>("GoodTracks");
        async<art::InEvent>();
      }
    void produce(art::Event& e,
                 art::ProcessingFrame const&) override
      {
        auto const& hits = e.getProduct<Hits>("GoodHits");
        auto tracks = make_tracks(hits);
        e.put(std::make_unique<Tracks>(std::move(tracks)),
              "GoodTracks");
      }
  }
}

DEFINE_ART_MODULE(expt::TrackMaker)
```

```
Trac
k
s
make_tracks(Hits const& hits) { ... }
```
Create tracks from hits for each event.

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

namespace {
  Tracks make_tracks(Hits const& hits) { ... }
}

namespace expt {
  class TrackMaker : public art::SharedProducer {
    public:
    TrackMaker(fhicl::ParameterSet const&) : 
      consumes<Hits, art::InEvent>("GoodHits");
    produces<Tracks, art::InEvent>("GoodTracks");

    async<art::InEvent>();

    void produce(art::Event& e, art::ProcessingFrame const&) override 
    {
      auto const& hits = e.getProduct<Hits>("GoodHits");
      auto tracks = make_tracks(hits);
      e.put(std::move(tracks), "GoodTracks");
    }
  };

  DEFINE_ART_MODULE(expt::TrackMaker)
```

This is just a transform? 😳
Create tracks from hits for each event.

\[ \mathcal{E}_n \xrightarrow{f} \mathcal{E}_n \xrightarrow{\text{make_tracks}} (\text{GoodHits})_8 \xrightarrow{\text{make_tracks}} (\text{GoodTracks})_8 \]

This is just a transform? 😐

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

namespace {
  Tracks make_tracks(Hits const& hits) { ... }
}

namespace expt {
  class TrackMaker : public art::SharedProducer {
    public:
      TrackMaker(fhicl::ParameterSet const&) :
        { consumes<Hits, art::InEvent>("GoodHits");
          produces<Tracks, art::InEvent>("GoodTracks");
        }
      async<art::InEvent>();

      void produce(art::Event& e, art::ProcessingFrame const&) override {
        auto const& hits = e.getProduct<Hits>("GoodHits");
        auto tracks = make_tracks(hits);
        e.put(std::make_unique<Tracks>(std::move(tracks))
          "GoodTracks");
      }
    }
  }
}

DEFINE_ART_MODULE(expt::TrackMaker)
```
Example

- Create tracks from hits for each event.

This is just a transform? 😬

Nobody wants this.

```
#include "art/Framework/Core/SharedProducer.h"
#include "art/Framework/Principal/Event.h"

namespace {  
  Tracks make_tracks(Hits const& hits) { ... }  
}

namespace expt {  
  class TrackMaker : public art::SharedProducer {  
    public:  
      TrackMaker(fhicl::ParameterSet const&):  
        {  
          consumes<Hits, art::InEvent>("GoodHits");  
          produces<Tracks, art::InEvent>("GoodTracks");  
        }  
    async<art::InEvent>();  
  }  
  void produce<art::Event& e, art::ProcessingFrame const&> override  
  {  
    auto const& hits = e.getProduct<Hits>("GoodHits");  
    auto tracks = make_tracks(hits);  
    e.put(std::make_unique<Tracks>(std::move(tracks)), "GoodTracks");  
  }  
};  

DEFINE_ART_MODULE(expt::TrackMaker)
```
Example

- Create tracks from hits for each event.

```cpp
#include "meld/module.hpp"

namespace {
    Hits make_tracks(Tracks const& tracks) { ... }
}

DEFINE_MODULE(m, config) {
    m.with(make_tracks)
        .transform("GoodHits").in_each("Event")
        .to("GoodTracks")
        .using_concurrency(unlimited);
}

Tracks make_tracks(Hits const& hits) { ... }
```

A better way...
Example

- Create tracks from hits for each event.

\[
\begin{align*}
\mathcal{E}_n & \xrightarrow{f} (b) \_8 \\
\mathcal{E}_n & \xrightarrow{\text{make\_tracks}} (\text{GoodTracks}) \_8
\end{align*}
\]

#include "meld/module.hpp"

namespace {
    Hits make_tracks(Tracks const& tracks) { ... }
}

DEFINE_MODULE(m, config) {
    m.with(make_tracks)
    .transform("GoodHits").in_each("Event")
    .to("GoodTracks")
    .using_concurrency(unlimited);
}

Tracks make_tracks(Hits const& hits) { ... }

A better way…

- Minimal boilerplate.
Example

- Create tracks from hits for each event.

\[ (a)_n \rightarrow f \rightarrow (b)_n \]
\[ (\mathcal{E}_n) \rightarrow \text{make\_tracks} \rightarrow (\text{GoodTracks})_n \]

```c++
#include "meld/module.hpp"

namespace {
    Hits make_tracks(Tracks const& tracks) {
        // ... 
    }
}

DEFINE_MODULE(m, config) {
    m.with(make_tracks)
        .transform("GoodHits").in_each("Event")
        .to("GoodTracks")
        .using_concurrency(unlimited);
}
```

- Minimal boilerplate.
- Event is now a label.

A better way...
Example

- Create tracks from hits for each event.

\[
\begin{align*}
\mathcal{E}_n \quad & \xrightarrow{f} \quad \mathcal{E}_n \\
(a)_8 \quad & \quad \text{(GoodHits)}_8 \\
(b)_8 \quad & \quad \text{(GoodTracks)}_8 \\
\end{align*}
\]

A better way…

```
#include "meld/module.hpp"

namespace {
    Hits make_tracks(Tracks const& tracks) { ... }
}

DEFINE_MODULE(m, config) {
    m.with(make_tracks)
        .transform("GoodHits").in_each("Event")
        .to("GoodTracks")
        .using_concurrency(unlimited);
}
```

- Minimal boilerplate.
- Event is now a label.
- Higher-order function is now explicit.
Meld implementation

- [https://github.com/knoepfel/meld](https://github.com/knoepfel/meld) (not even alpha release)
- Implemented using oneTBB’s flow graph

<table>
<thead>
<tr>
<th>Supported construct</th>
<th>User function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform (Map)</td>
<td>$f(a) \to b$</td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>$f(a) \to \text{Boolean}$</td>
<td></td>
</tr>
<tr>
<td>Monitor</td>
<td>$f(a) \to \text{Void}$</td>
<td></td>
</tr>
<tr>
<td>Reduction (Fold)</td>
<td>$f_c(a) \to c$</td>
<td>For splitting and then combining events</td>
</tr>
<tr>
<td>Splitter (Unfold)</td>
<td>$f_n(a) \to (d)_n$</td>
<td></td>
</tr>
<tr>
<td>Zip</td>
<td>—</td>
<td>For combining arguments to user functions</td>
</tr>
<tr>
<td>Sliding window</td>
<td>—</td>
<td>To do: For sliding over adjacent events</td>
</tr>
</tbody>
</table>

Standard data-processing idioms
Sample hierarchies tested by Meld

```
[info] Number of worker threads: 12
[info] Processed levels:
  job
    run: 1
      subrun: 2
        event: 10
[info] CPU efficiency: 259.55%
[info] Max. RSS: 6.205 MB
```

Performance numbers are preliminary

*art-based hierarchy*
Sample hierarchies tested by Meld

Performance numbers are preliminary

Non-trivial hierarchy
Sample hierarchies tested by Meld

Performance numbers are preliminary
Summary

“Ways change, Stil.” —Paul from Dune by Frank Herbert

• Supporting DUNE’s framework needs suggests rethinking framework concepts.
Summary

“Ways change, Stil.” —Paul from *Dune* by Frank Herbert

- Supporting DUNE’s framework needs suggests rethinking framework concepts.
- Meld seeks to address these needs by considering a framework job as a

  (1) **graph of data products** connected by
  (2) **user-provided operators** of
  (3) **higher-order functions**.

- It is not a framework-less framework, but it *is* less framework coupling.
- Preliminary work indicates this is a productive avenue to pursue.
Summary

“Ways change, Stil.” —Paul from Dune by Frank Herbert

• Supporting DUNE’s framework needs suggests rethinking framework concepts.
• Meld seeks to address these needs by considering a framework job as a

(1) graph of data products connected by
(2) user-provided operators of
(3) higher-order functions.

• It is not a framework-less framework, but it is less framework coupling.
• Preliminary work indicates this is a productive avenue to pursue.

Thank you for your time and attention.
Backup slides
Reduction example

class MyAccumulator : public art::EDProducer {
public:
    MyAccumulator(ParameterSet const&)
    {
        produces<int, art::InSubRun>("sum");
    }

    void produce(art::Event&) override
    {
        ++counter_;
    }

    void endSubRun(art::SubRun& sr) override
    {
        sr.put(std::make_unique<int>(counter_), "sum");
        counter_ = 0;
    }

private:
    int counter_ = 0;
};

DEFINE_ART_MODULE(MyAccumulator)

void accumulate(int& counter,
    meld::level_id const&)
{
    ++counter;
}

DEFINE_MODULE(m) {
    m.with(accumulate, 0).for_each("SubRun")
       .reduce("id").in_each("Event")
       .to("sum");
}
Looking at the data (products)

Each element of the set is a *data product*, which is:

- Opaque to the framework
  \[\implies\text{Separation of user space from framework}\]
- Immutable (definition of set element)
- A member of *at least* one set
- Identifiable
Higher-order functions

• We are interested in the mappings of the form:

\[
\{(a)_n \rightarrow (b)_m \} \in \mathcal{D}
\]

• Each object \(a\) corresponds to a tuple of arguments passed to \(f\).
• The signature of \(f\) and the value \(f(a)\), depends on the higher-order function.
• The above mapping happens within a domain \(\mathcal{D}\) (e.g. job, run, event).
• Each object \(a\) is an element of a subset of the domain \(\mathcal{D}\).
## Supported higher-order functions

<table>
<thead>
<tr>
<th>Meld term</th>
<th>CS term</th>
<th>Mathematical description</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform</td>
<td>Map</td>
<td>$(a)_n \overset{f}{\to} (b)_n$</td>
<td>where $f(a) \to b$</td>
</tr>
<tr>
<td>Filter</td>
<td>Filter</td>
<td>$(a)_n \overset{f}{\to} (a)_m$ where $m \leq n$</td>
<td>where $f(a) \to$ Boolean</td>
</tr>
<tr>
<td>Monitor</td>
<td>—</td>
<td>$(a)_n \overset{f}{\to} ()_0$</td>
<td>where $f(a) \to$ Void</td>
</tr>
<tr>
<td>Reduction</td>
<td>Fold</td>
<td>$(a)_n \overset{f_c}{\to} (c)_1$</td>
<td>where $f_c(a) \to c$</td>
</tr>
<tr>
<td>Splitter</td>
<td>Unfold</td>
<td>$(a)_1 \overset{f_n}{\to} (d)_m$</td>
<td>where $f_n(a) \to (d)_n$</td>
</tr>
<tr>
<td>Zip</td>
<td>Zip</td>
<td>$((a)_n , (b)_n) \to (a, b)_n$</td>
<td>More nested domain</td>
</tr>
</tbody>
</table>

---

Transform: $\map{a}{}{b}$

Filter: $\filter{a}{m \leq n}$

Monitor: $\monitor{a}{0}$

Reduction: $\fold{a}{c}$

Splitter: $\unfold{a}{d}$

Zip: $\zip{(a)_n, (b)_n}{(a, b)_n}$