Beware the Default Constructor

Robert Leahy
Lead Software Engineer
rleahy@rleahy.ca

LSEG

Message Callback

Each message has its own type and each type has its own virtual function

LSEG

```
struct INormalizedMessageFeedCallback {
  virtual void OnAggregatedPriceUpdate(const AggregatedPriceUpdate&, void*) = 0;
  virtual void OnPacketBegin(const PacketBegin&, void*) = 0;
  virtual void OnPacketEnd(const PacketEnd&, void*) = 0;
  virtual void OnMissingPackets(const MissingPackets&, void*) = 0;
  virtual void OnError(const Error&, void*) = 0;
  virtual void OnRecoveryBegin(const RecoveryBegin&, void*) = 0;
  virtual void OnRecoveryEnd(const RecoveryEnd&, void*) = 0;
  virtual void OnMissingProductMessages(const MissingProductMessages&, void*) = 0;
  virtual void OnUnsubscribedProduct(const UnsubscribedProduct&, void*) = 0;
  virtual void OnAddOrder(const AddOrder&, void*) = 0;
  virtual void OnModifyOrder(const ModifyOrder&, void*) = 0;
  virtual void OnCancelOrder(const CancelOrder&, void*) = 0;
  virtual void OnPriceLevelUpdate(const PriceLevelUpdate&, void*) = 0;
  virtual void OnAddPriceLevel(const AddPriceLevel&, void*) = 0;
  virtual void OnModifyPriceLevel(const ModifyPriceLevel&, void*) = 0;
  virtual void OnDeletePriceLevel(const DeletePriceLevel&, void*) = 0;
  virtual void OnClearOrders(const ClearOrders&, void*) = 0;
  virtual void OnClearPriceLevels(const ClearPriceLevels&, void*) = 0;
  virtual void OnOrderImbalance(const OrderImbalance&, void*) = 0;
  virtual void OnAuctionSummary(const AuctionSummary&, void*) = 0;
  virtual void OnProductStatus(const ProductStatus&, void*) = 0;
  virtual void OnFeedStatus(const FeedStatus&, void*) = 0;
  virtual void OnMarketStatus(const MarketStatus&, void*) = 0;
  virtual void OnRetailPriceImprovement(const RetailPriceImprovement&, void*) = 0;
  virtual void OnBBOQuote(const BBOQuote&, void*) = 0;
  virtual void OnNBBOQuote(const NBBOQuote&, void*) = 0;
  virtual void OnProductAnnouncement(const ProductAnnouncement&, void*) = 0;
  virtual void OnProductStatistics(const ProductStatistics&, void*) = 0;
  virtual void OnTrade(const Trade&, void*) = 0;
  virtual void OnTradeBreak(const TradeBreak&, void*) = 0;
  // ...
};
```

Singly-Typed Channel

`Mailbox` allows communication between threads but only accepts a single type of data

Adaptation will be required to collapse multiple types into the single type accepted by this channel

```
template<typename T>
struct Mailbox {
  void push(Member m);
  template<typename... Args>
  void emplace(Args&&... args);
  void finish();
};
```

This is a job for `std::variant`

LSEG

Variant

Each message has a corresponding alternative

```
using PipelineVariant = std::variant<
  FlattenedAPU,
  PacketBeginWithStorage,
  PacketEndWithStorage,
  MissingPackets,
  ErrorWithStorage,
  RecoveryBegin,
  RecoveryEnd,
  MissingProductMessages,
  UnsubscribedProduct,
  AddOrder,
  ModifyOrder,
  CancelOrder,
  PriceLevelUpdate,
  AddPriceLevel,
  ModifyPriceLevel,
  DeletePriceLevel,
  ClearOrders,
  ClearPriceLevels,
  OrderImbalance,
  AuctionSummary,
  ProductStatus,
  FeedStatus,
  MarketStatus,
  RetailPriceImprovement,
  BBOQuote,
  NBBOQuote,
  ProductAnnouncement,
  ProductStatistics,
  Trade,
  //  ...
>;
```

LSEG

Adaptation

Due to requirements to persist transient storage some types must be different

```cpp
using PipelineVariant = std::variant<
  FlattenedAPU,
  PacketBeginWithStorage,
  PacketEndWithStorage,
  MissingPackets,
  ErrorWithStorage,
  RecoveryBegin,
  RecoveryEnd,
  MissingProductMessages,
  UnsubscribedProduct,
  AddOrder,
  ModifyOrder,
  CancelOrder,
  PriceLevelUpdate,
  AddPriceLevel,
  ModifyPriceLevel,
  DeletePriceLevel,
  ClearOrders,
  ClearPriceLevels,
  OrderImbalance,
  AuctionSummary,
  ProductStatus,
  FeedStatus,
  MarketStatus,
  RetailPriceImprovement,
  BBOQuote,
  NBBOQuote,
  ProductAnnouncement,
  ProductStatistics,
  Trade,
  //  ...
>;
```

LSEG

Two Variants

Left variant is types that will actually be sent down the pipeline, right variant is types actually emitted by system via virtual functions seen earlier

LSEG

```cpp
using PipelineVariant = std::variant<
  FlattenedAPU,
  PacketBeginWithStorage,
  PacketEndWithStorage,
  MissingPackets,
  ErrorWithStorage,
  RecoveryBegin,
  RecoveryEnd,
  MissingProductMessages,
  UnsubscribedProduct,
  AddOrder,
  ModifyOrder,
  CancelOrder,
  PriceLevelUpdate,
  AddPriceLevel,
  ModifyPriceLevel,
  DeletePriceLevel,
  ClearOrders,
  ClearPriceLevels,
  OrderImbalance,
  AuctionSummary,
  ProductStatus,
  FeedStatus,
  MarketStatus,
  RetailPriceImprovement,
  BBOQuote,
  NBBOQuote,
  ProductAnnouncement,
  ProductStatistics,
  Trade,
  // ...
>;
```

```cpp
using Variant = std::variant<
  AggregatedPriceUpdate,
  PacketBegin,
  PacketEnd,
  MissingPackets,
  Error,
  RecoveryBegin,
  RecoveryEnd,
  MissingProductMessages,
  UnsubscribedProduct,
  AddOrder,
  ModifyOrder,
  CancelOrder,
  PriceLevelUpdate,
  AddPriceLevel,
  ModifyPriceLevel,
  DeletePriceLevel,
  ClearOrders,
  ClearPriceLevels,
  OrderImbalance,
  AuctionSummary,
  ProductStatus,
  FeedStatus,
  MarketStatus,
  RetailPriceImprovement,
  BBOQuote,
  NBBOQuote,
  ProductAnnouncement,
  ProductStatistics,
  Trade,
  // ...
>;
```

Uniform Metadata

Our pipeline variant doesn’t go directly down the pipeline since we want a layer to introduce uniform metadata (a pointer to the session which emitted the message)

```
struct Update {
  PipelineVariant message;
  const ISession *session;
};
```

LSEG

Implementation

Handler for each message type simply defers to `buildMessage`

`buildMessage` grabs the session pointer (unfortunately a global) and defers to the derived class via the template method pattern

Derived implementation (bottom of slide) default constructs an `Update`, populates it, and then pushes it into the output `Mailbox`

LSEG

```
struct Handler : IAggregatedPriceFeedCallback {
  virtual void message(Variant, const ISession*) = 0;
  void buildMessage(Variant variant) {
    prepareMessage(std::move(variant), CurrentExchangeSession::Get());
  }
  void OnAggregatedPriceUpdate(const AggregatedPriceUpdate& arg, void*) final {
    buildMessage(arg);
  }
  void OnPacketBegin(const PacketBegin& arg, void*) final {
    buildMessage(arg);
  }
  void OnPacketEnd(const PacketEnd& arg, void*) final {
    buildMessage(arg);
  }
  void OnMissingPackets(const MissingPackets& arg, void*) final {
    buildMessage(arg);
  }
  void OnError(const Error& arg, void*) final {
    buildMessage(arg);
  }
  void OnRecoveryBegin(const RecoveryBegin& arg, void*) final {
    buildMessage(arg);
  }
  //  ...
};
//  ...
void message(Variant variant, const ISession* const session) {
  Update update;
  update.session = session;
  std::visit(OverloadSet{/* ... */}, std::move(variant));
  output.push(std::move(update));
}
```

Let’s Look Closer

What happens for each update?

- Message is copied into `std::variant` to call `buildMessage`
- `std::variant` is moved to call `message`
- `Update` is default constructed (which default constructs contained `std::variant`, which in turn default constructs `FlattenedAPU`)
- Input `std::variant` is visited
- Alternative of `std::variant` contained by `Update` is changed to reflect actual message type being processed
- `Update` is moved into call to `Mailbox::push`

Everything is moved so it’s fast, right?

Variants are 2 184 bytes

LSEG

Test run takes 8 hours

LSEG

Old application took 4.5

LSEG

Let’s Look Closer

What happens for each update?

- Message is copied into `std::variant` to call `buildMessage`
- `std::variant` is moved to call `message`
- `Update` is default constructed (which default constructs contained `std::variant`, which in turn default constructs `FlattenedAPU`)
- Input `std::variant` is visited
- Alternative of `std::variant` contained by `Update` is changed to reflect actual message type being processed
- `Update` is moved into call to `Mailbox::push`

Revisiting Update

Before...

```
struct Update {
  PipelineVariant message;
  const ISession* session;








};
```

LSEG

Revisiting Update

We could simply use aggregate initialization but adding the constructor enforces that this is the only way to construct this type ensuring we don't wind up with this performance degradation again

```
struct Update {
  PipelineVariant message;
  const ISession* session;
  template<typename T>
    requires std::constructible_from<PipelineVariant, T>
  constexpr explicit Update(
    T&& t,
    const bp::feed::ISession* const session)
      : message(std::forward<T>(t)),
        session(session)
  {}
};
```

Revisiting message

Before...

```
void message(Variant variant, const ISession* const session) {
  Update update;
  update.session = session;
  std::visit(
    OverloadSet{/* ... */},
    std::move(variant));
  output.push(std::move(update));
}
```

LSEG

Revisiting `message`

Rather than creating intermediate objects and moving them around this implementation creates the output object directly in place within the `Mailbox`

```
void message(Variant variant, const ISession* const session) {
  output.emplace(
    Elide([&]() {
      return std::visit(
        OverloadSet{/* ... */},
        std::move(variant));
    }),
    session);
}
```

Elide?

Utility to access C++17 guaranteed RVO within the context of emplacement/forwarding construction

```
void message(Variant variant, const ISession* const session) {
  output.emplace(
    Elide([&]() {
      return std::visit(
        OverloadSet{/* ... */},
        std::move(variant));
    }),
    session);
}
```

LSEG

C++ now

An Adventure in Modern Library Design

Robert Leahy

2024

Time down to 6 hours

LSEG

Let’s Look Closer

What happens for each update?

- Message is copied into `std::variant` to call `buildMessage`
- `std::variant` is moved to call `message`
- `Update` is default constructed (which default constructs contained `std::variant`, which in turn default constructs `FlattenedAPU`)
- Input `std::variant` is visited
- Alternative of `std::variant` contained by `Update` is changed to reflect actual message type being processed
- `Update` is moved into call to `Mailbox::push`

Revisiting Handler

Before...

```
struct Handler : IAggregatedPriceFeedCallback {
  virtual void message(Variant, const ISession*) = 0;
  void buildMessage(Variant variant) {
    prepareMessage(
      std::move(variant),
      CurrentExchangeSession::Get());
  }
  //  ...
};
```

LSEG

Revisiting Handler

Using the CRTP (curiously recurring template pattern) allows us to both eliminate the virtual call and eliminate the intermediate `std::variant`

```
template<typename Derived>
struct Handler : IAggregatedPriceFeedCallback {
  template<typename Message>
  void buildMessage(const Message& message) {
    static_cast<Derived&>(*this).message(
      message,
      CurrentExchangeSession::Get());
  }
  // ...
};
```

Re-revisiting `message`

Before…

```
void message(Variant variant, const ISession* const session) {
  output.emplace(
    Elide([&]() {
      return std::visit(
        OverloadSet{/* ... */},
        std::move(variant));
    }),
    session);
}
```

LSEG

Re-revisiting `message`

Refactoring into a template eliminates the need to call `std::visit` since we're directly instantiated with the input type

```
template<typename Message>
void message(const Message& message, const ISession* const session) {
  output.emplace(
    Elide([&]() {
      if constexpr (/* ... */) {
        return PipelineVariant(
          std::in_place_type</* ... */>,
          /* ... */);
      } else if constexpr (/* ... */) {
        //  ...
      } else {
        return PipelineVariant(
          std::in_place_type</* ... */>,
          /* ... */);
      }
    }),
    session);
}
```

LSEG

Time comparable to original

LSEG

Everything is moved so it’s fast, right?

LSEG

TANSTAAFL

There Ain’t No Such Thing As A Free Lunch

LSEG

Thank you

Robert Leahy
Lead Software Engineer
rleahy@rleahy.ca

LSEG