AF Forwarding Utility

Painless Data Distribution in Actor Framework

Casey May

Zyah Solutions Co-Founder

October 20, 2021

Introduction

- Zyah Solutions
  - Founded by a group of CLAs with particular experience and interest in developing large-scale applications.
  - We pride ourselves on clean, modular and high-quality solutions adhering to SOLID design principles.
  - Together we have over 50 years of LabVIEW experience.
- Personally been using LabVIEW since 2006 and been a CLA since 2007.

Agenda

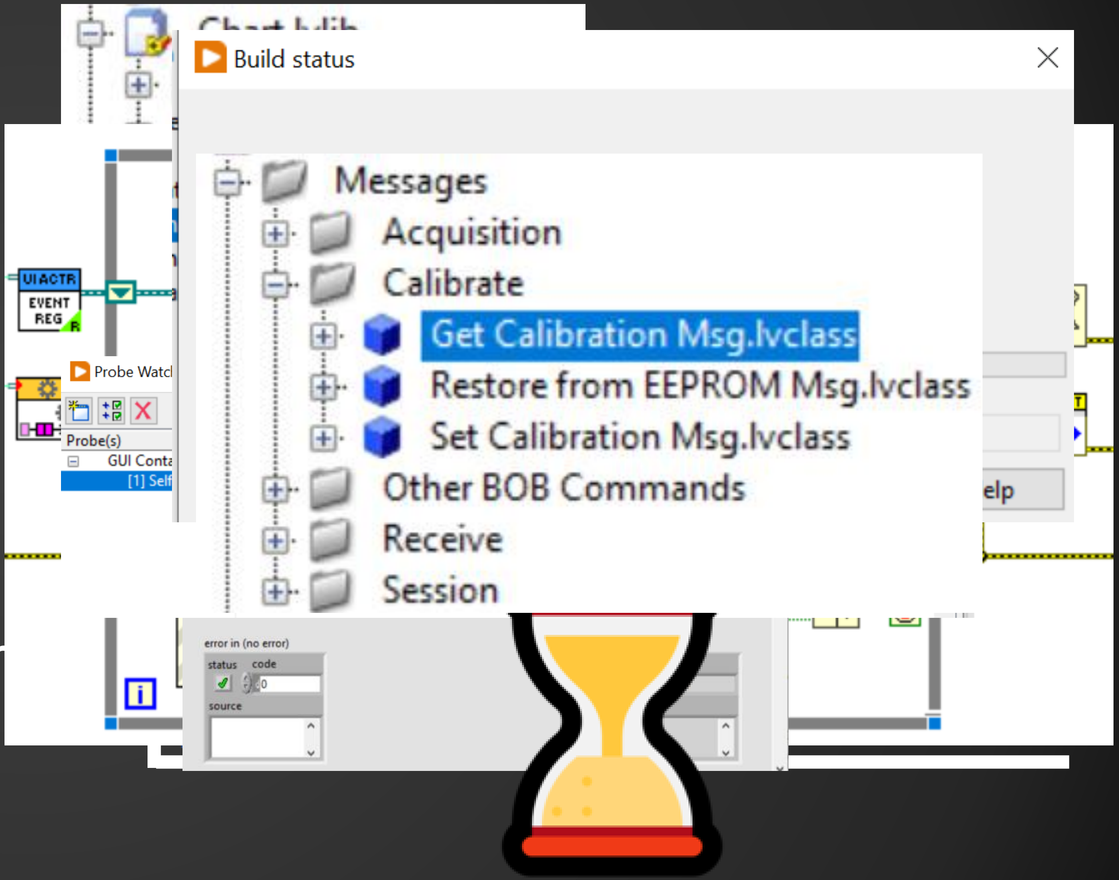
- Benefits and Challenges of Actor Framework
- Today’s Challenge: Distributing Data
- Standard Solutions
- A Better Way?

Actor Framework:
The Broccoli of LabVIEW

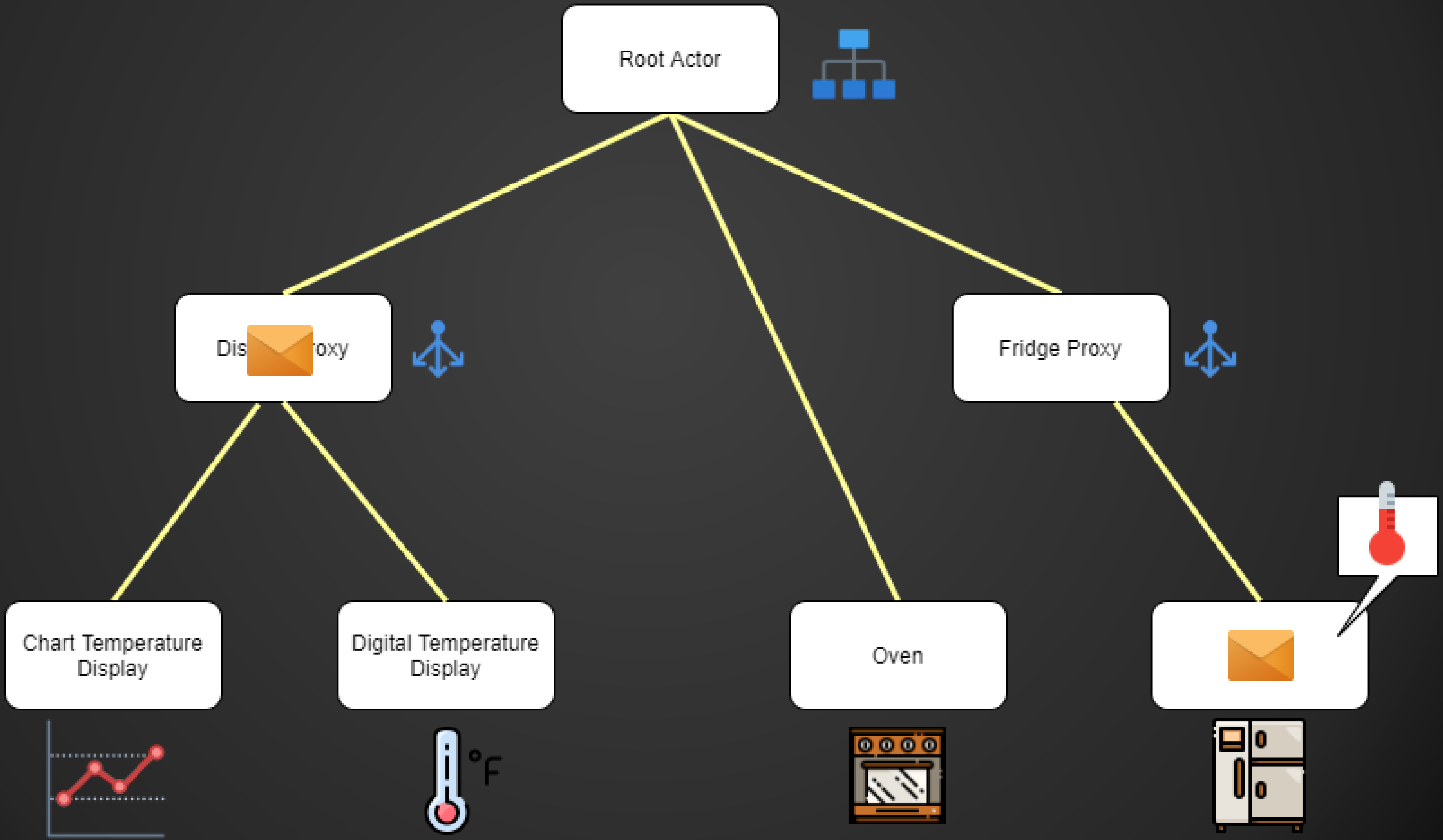
- It’s a clean, healthy solution for large and complex applications.
- Scales well and provides great flexibility with out-of-the-box clone handling, inheritance, and dynamic dispatch.
- Lends itself to SOLID design.
- Unpalatable to some LabVIEW users.

Actor Framework:
Common Objections

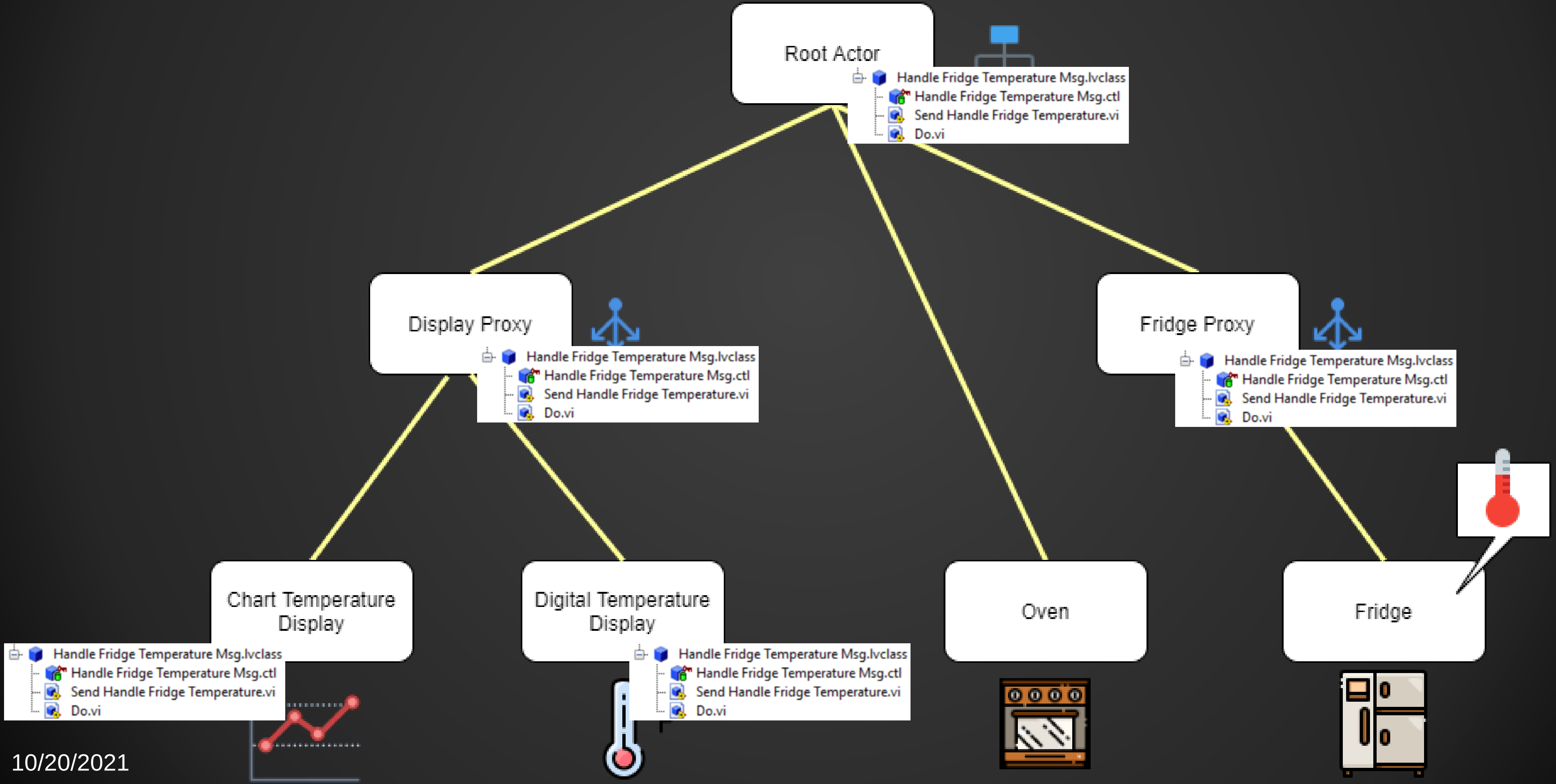
- Steep learning curve
- Challenging to debug
- IDE slowdown for large Actor projects
- Standard AF Msgs force tight coupling*
- Tooling is lacking for common operations
- Moving data through wide and/or deep trees is challenging



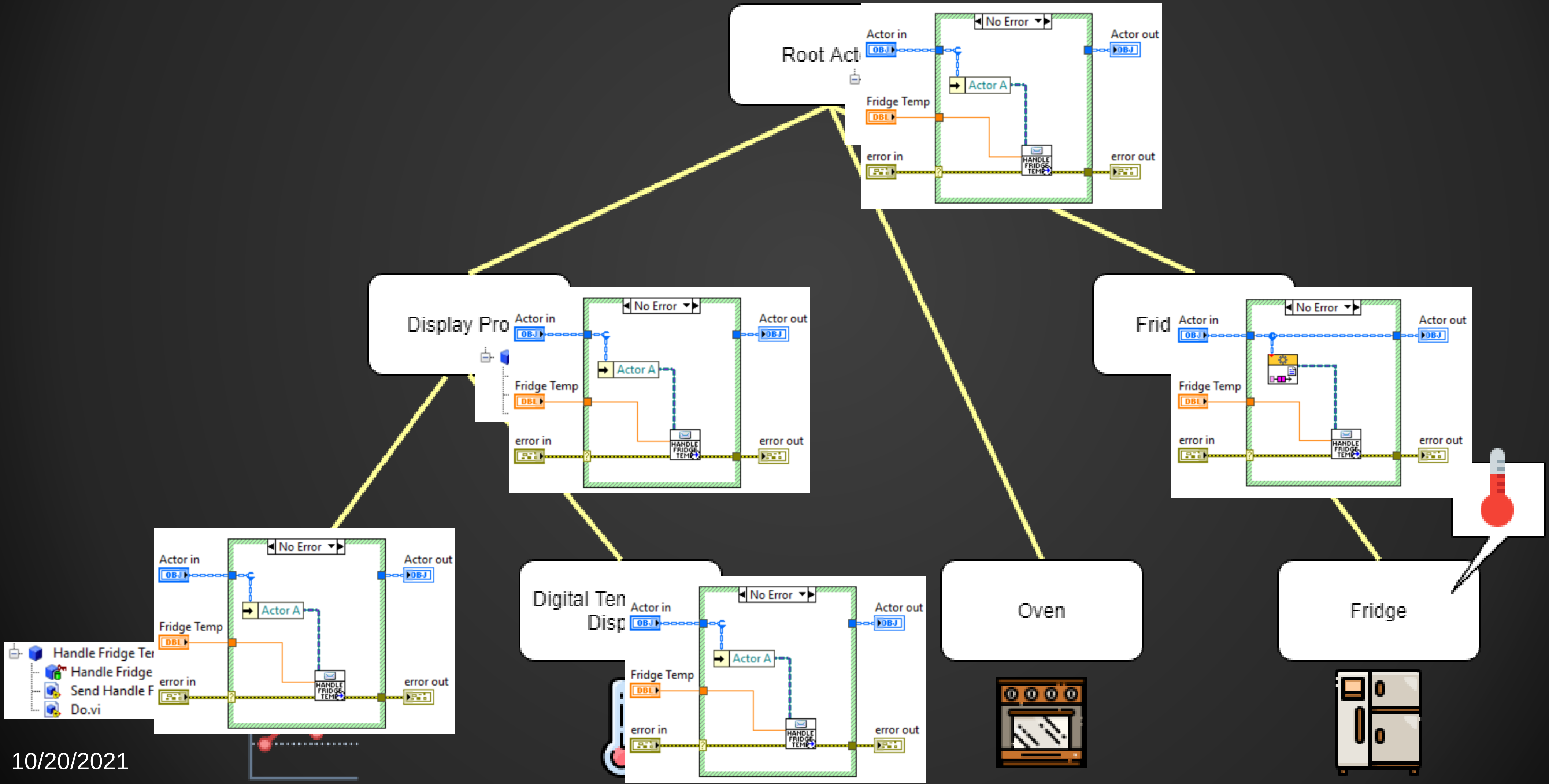
Traversing the Tree



Traversing the Tree

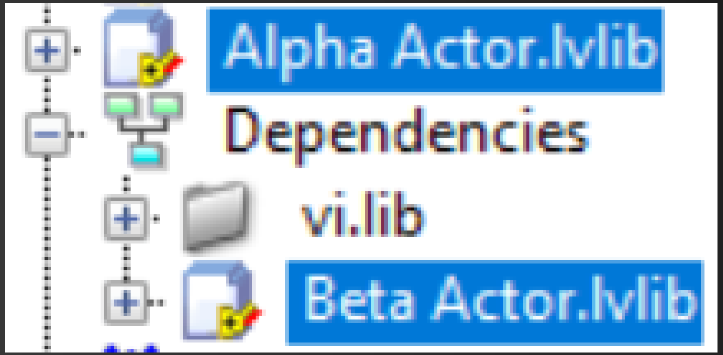

Root Actor
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi
Display Proxy
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi
Fridge Proxy
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi
Chart Temperature Display
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi
Digital Temperature Display
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi
Oven
Fridge

Traversing the Tree



Additional Challenges

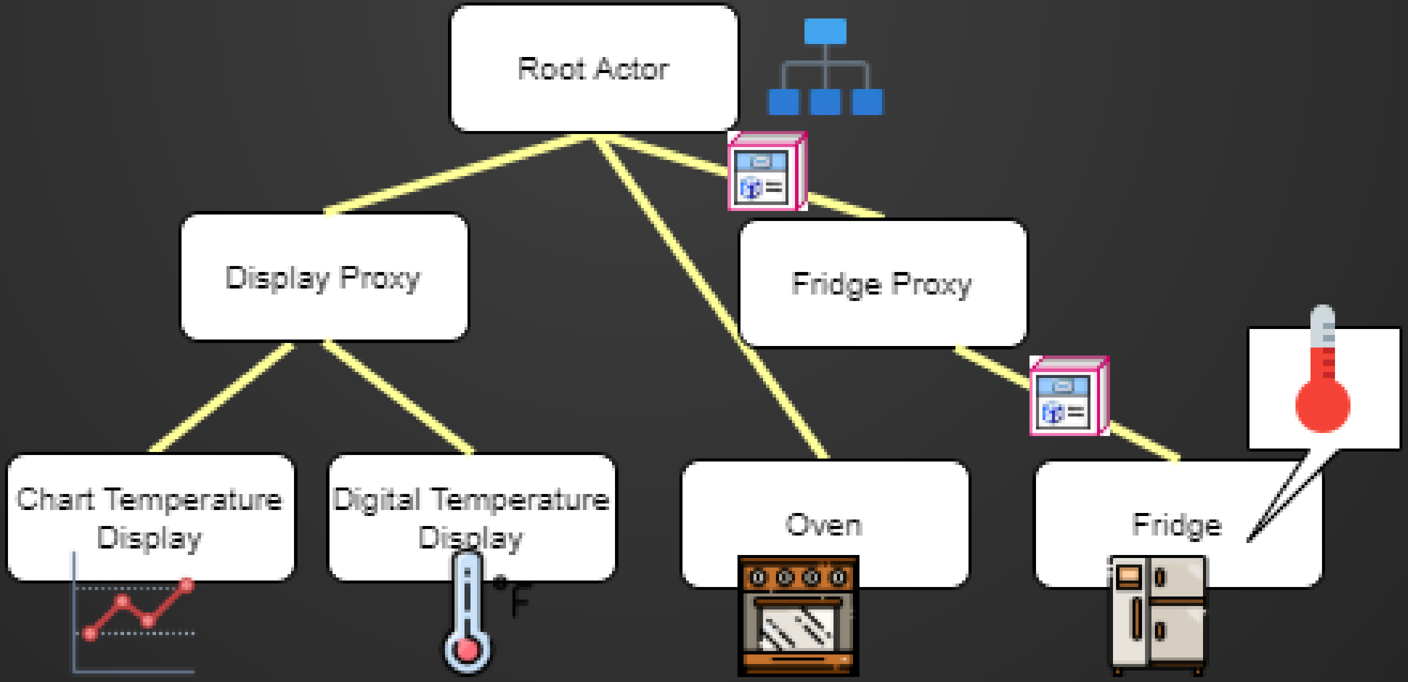
- Using **default** Actor messaging forces tight coupling between actor layers.*
- In plugin architectures, data recipients could be dynamic, which requires additional overhead to handle.



Alpha Actor.lvlib
Dependencies
vi.lib
Beta Actor.lvlib

Important Note #1

- We're only talking about **announcements** and not **requests**.
- Simple distribution of data updates is relatively safe and often proceeds unchanged through the tree.



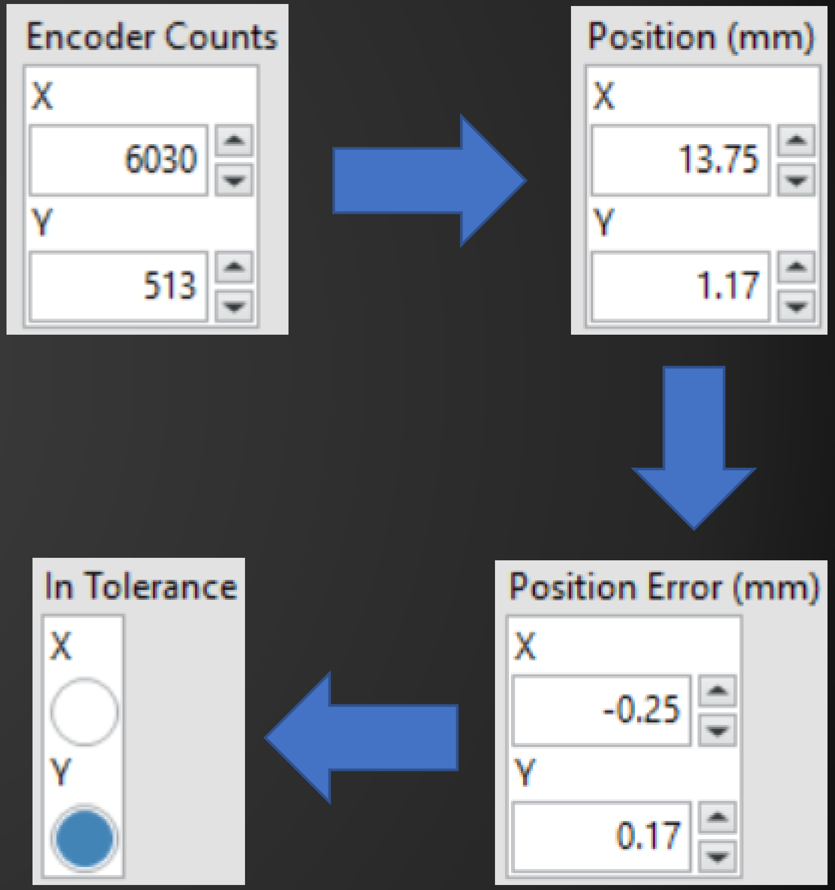
Important Note #1

- We’re only talking about **announcements** and not **requests**.
- Simple distribution of data updates is relatively safe and often proceeds unchanged through the tree.
- Requests involve “control” of an Actor and are less safe for automatic broadcast.

Traditional Solutions

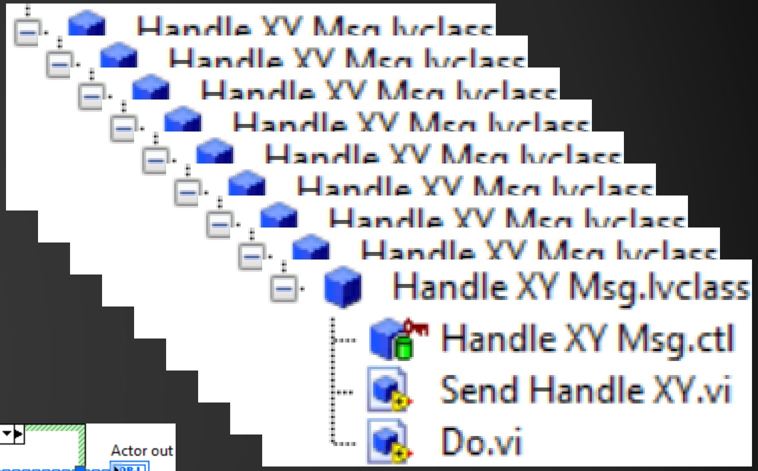
- Many individual messages that do exactly (or nearly exactly) the same thing.
- **Pro:**

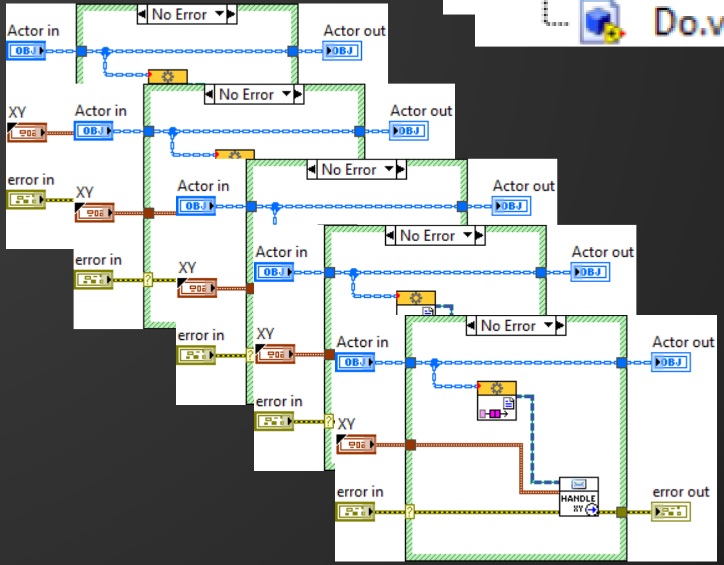
  Explicit handling at each actor, so data can be processed, modified, or abstracted as-needed.



Traditional Solutions

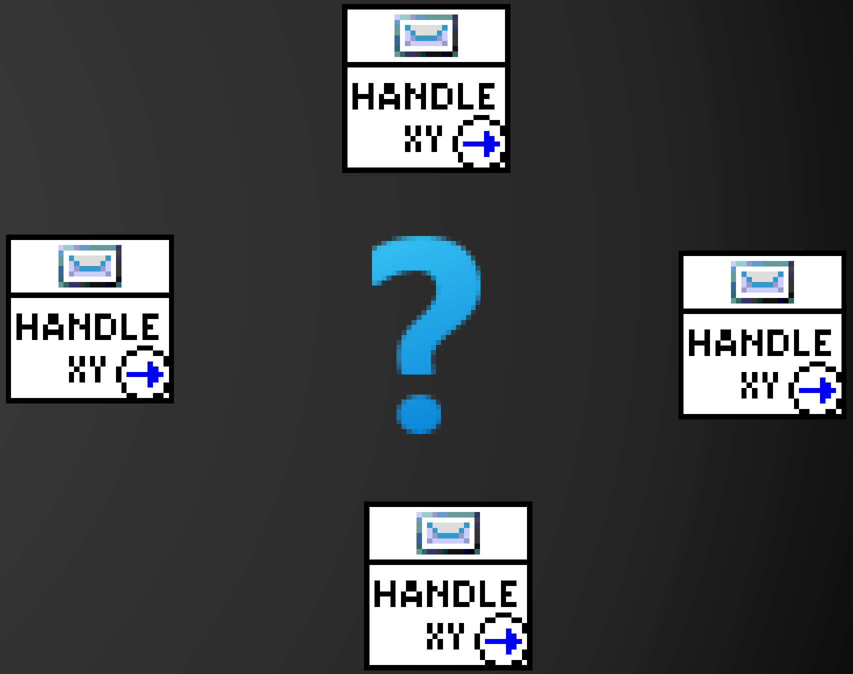
- Many individual messages that do exactly (or nearly exactly) the same thing.
- **Cons:**
  - Lots of additional classes and methods, which can bloat the IDE.
  - Undesirable duplication of identical or nearly-identical code.





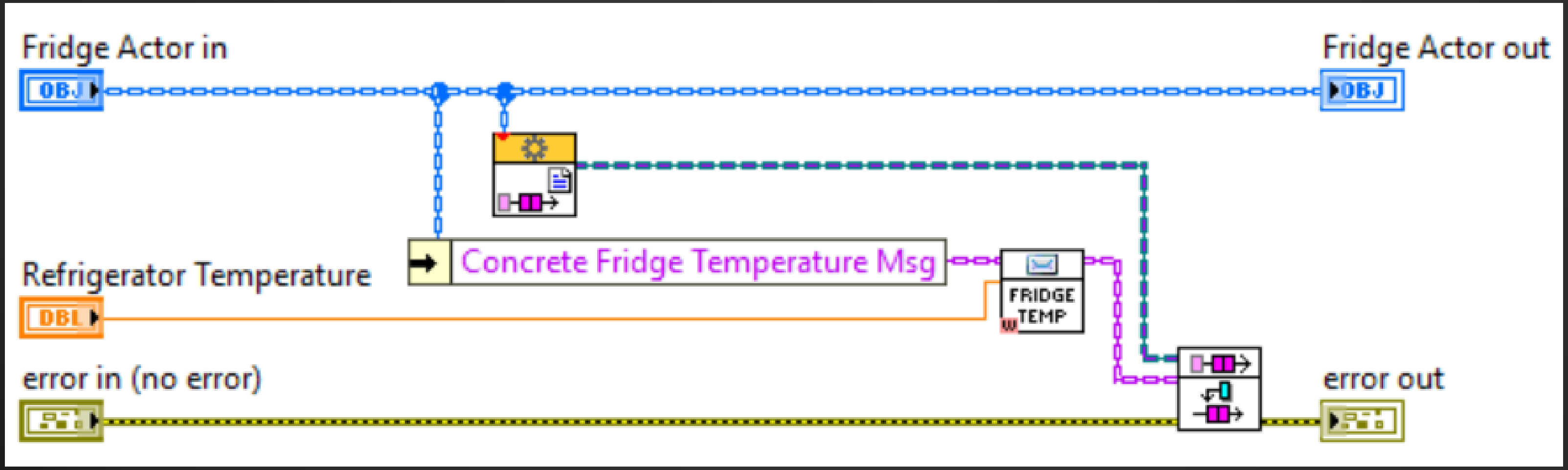
Traditional Solutions

- Many individual messages that do exactly (or nearly exactly) the same thing.
- **Cons:**
  - Lots of additional classes and methods, which can bloat the IDE.
  - Undesirable duplication of identical or nearly-identical code.
  - Easy to mix up Send methods with similar names and identical connector panes.

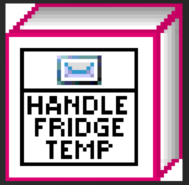


Traditional Solutions

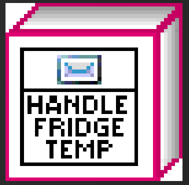
- Creating "0-coupled" messages for each announcement type
  - Allows strict-typing
  - Problem of many overrides remains
  - Setup/debugging is challenging because messaging origin / routing is less clear



Abstract

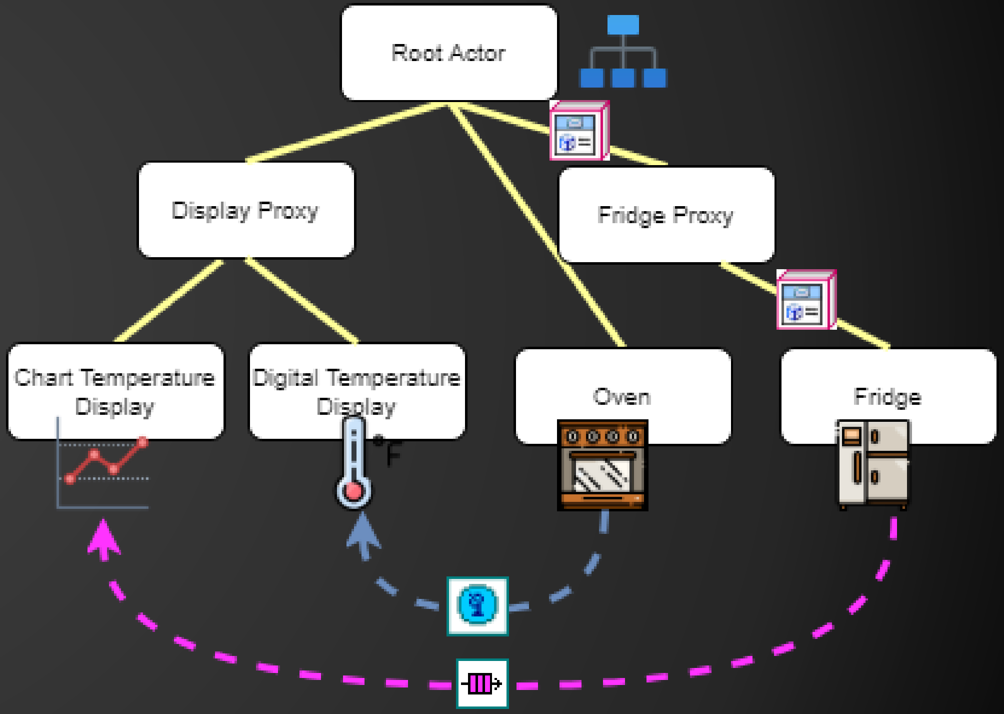


Concrete



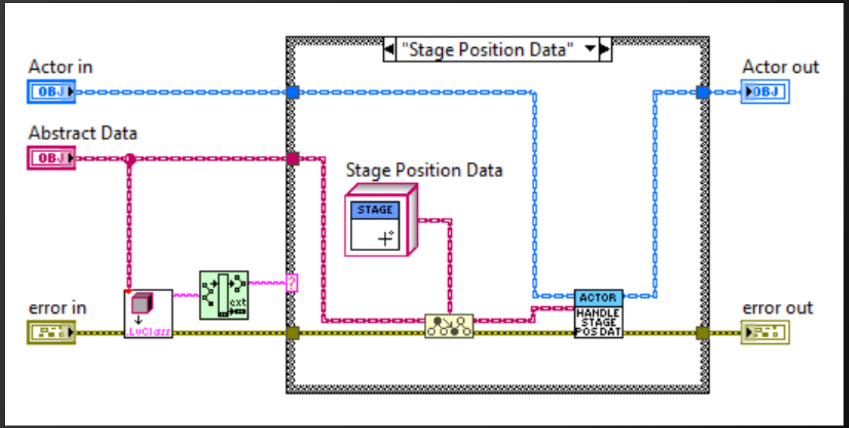
Traditional Solutions

- Circumventing the actor tree using alternate messaging such as user events, queues, etc.
  - Messaging are routed through multiple transport mechanisms, which can complicate debugging.
  - Code clarity and readability often suffers with multiple transport mechanisms.
  - Competing transport mechanisms increases potential for timing/locking issues.



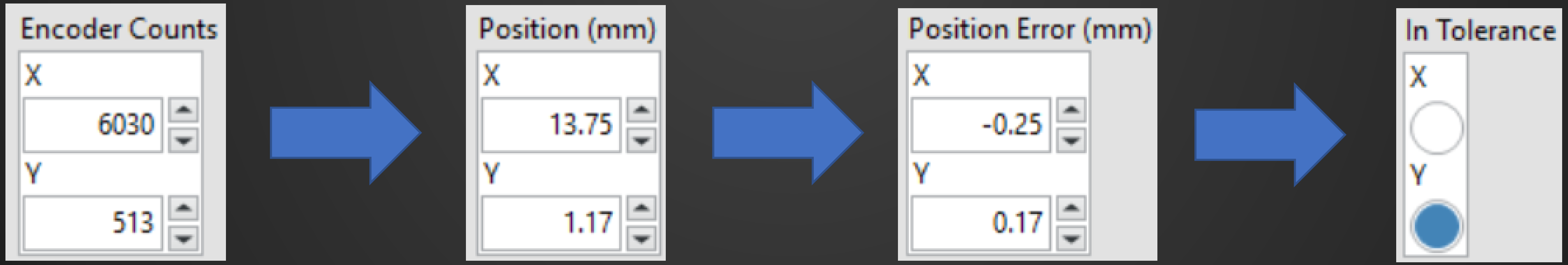
Traditional Solutions

- Sharing abstract data through a common method in your base actor
  - Results in overloaded override methods
  - Potential for run-time parsing errors
  - Brittle to changes in message names, etc



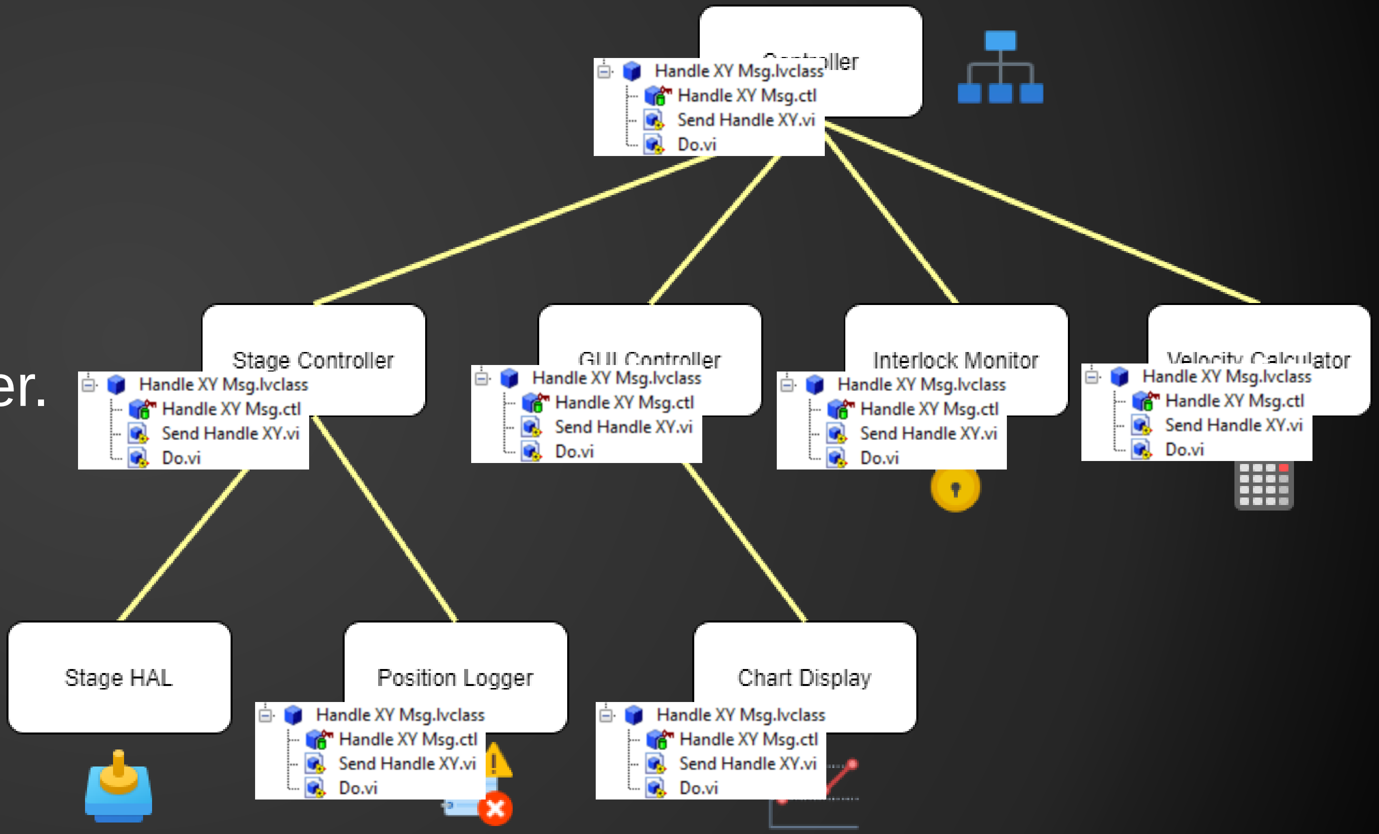
Important Note #2

- We're going to address announcement Msgs that do not need to be modified before reaching their ultimate destination(s).
  - If the contents of the messages need to be modified in transit, we suggest sticking to the individual/unique Msg approach.



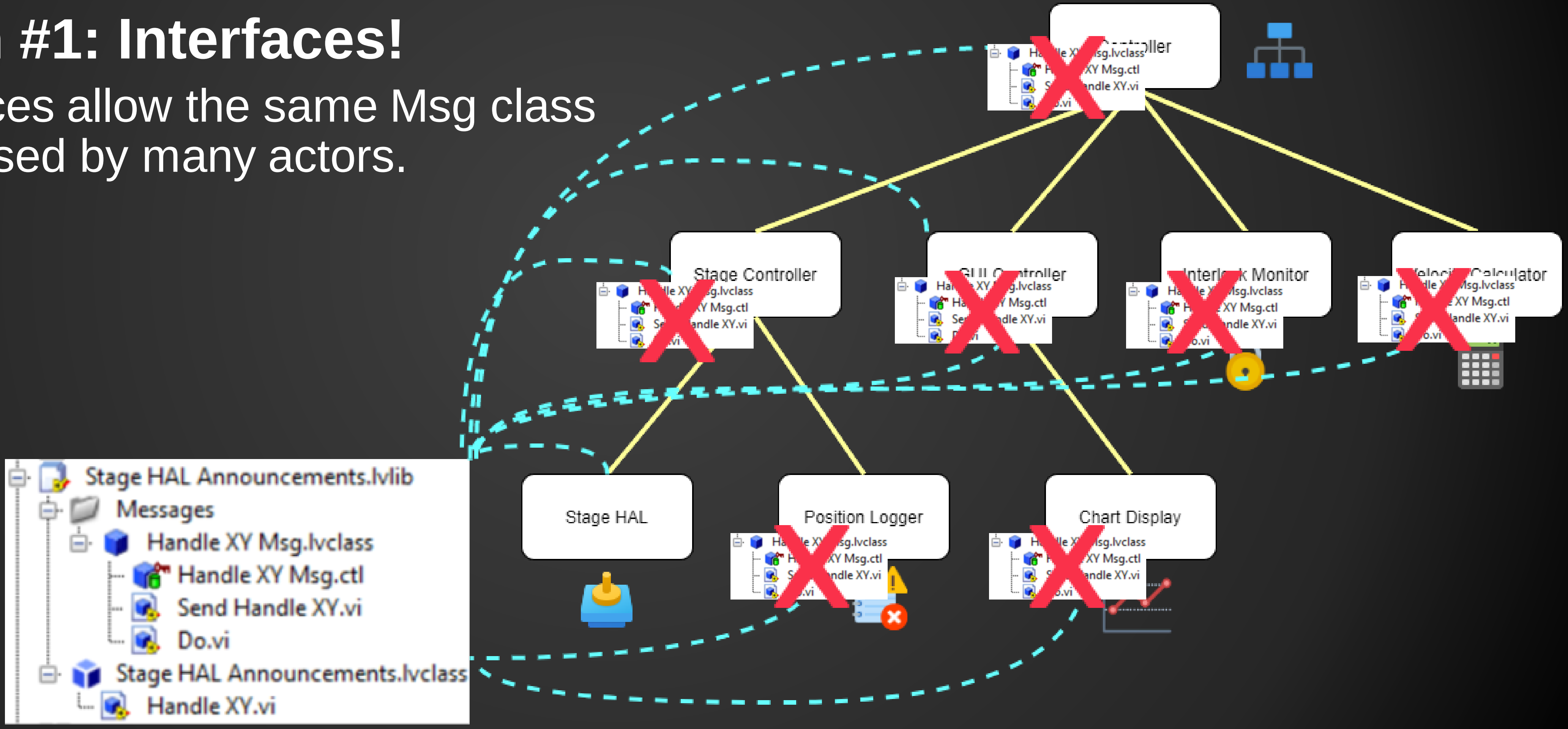
How Can We Make This Better?

- **Challenge #1: A better solution would need to…**
  - Limit duplication of identical messages when appropriate.
  - Not couple actors to one another.
  - Use strictly-typed messages.



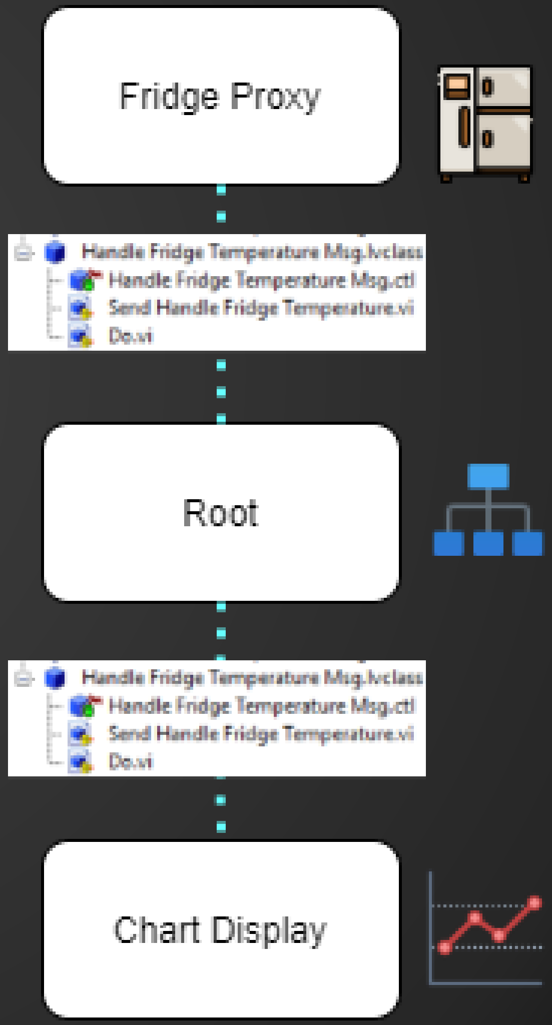
How Can We Make This Better?

- **Solution #1: Interfaces!**
  - Interfaces allow the same Msg class to be used by many actors.



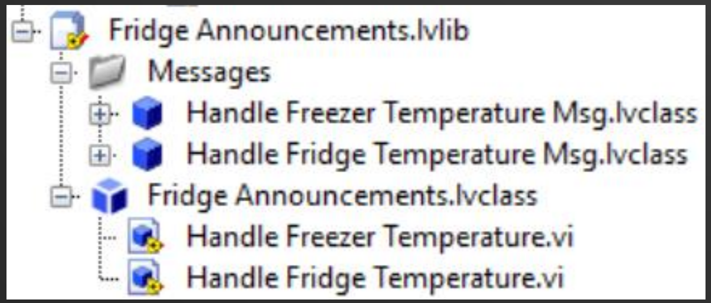
How Can We Make This Better?

- **Solution #1: Interfaces!**
  - Interfaces allow the same Msg class to be used by many actors.



Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi

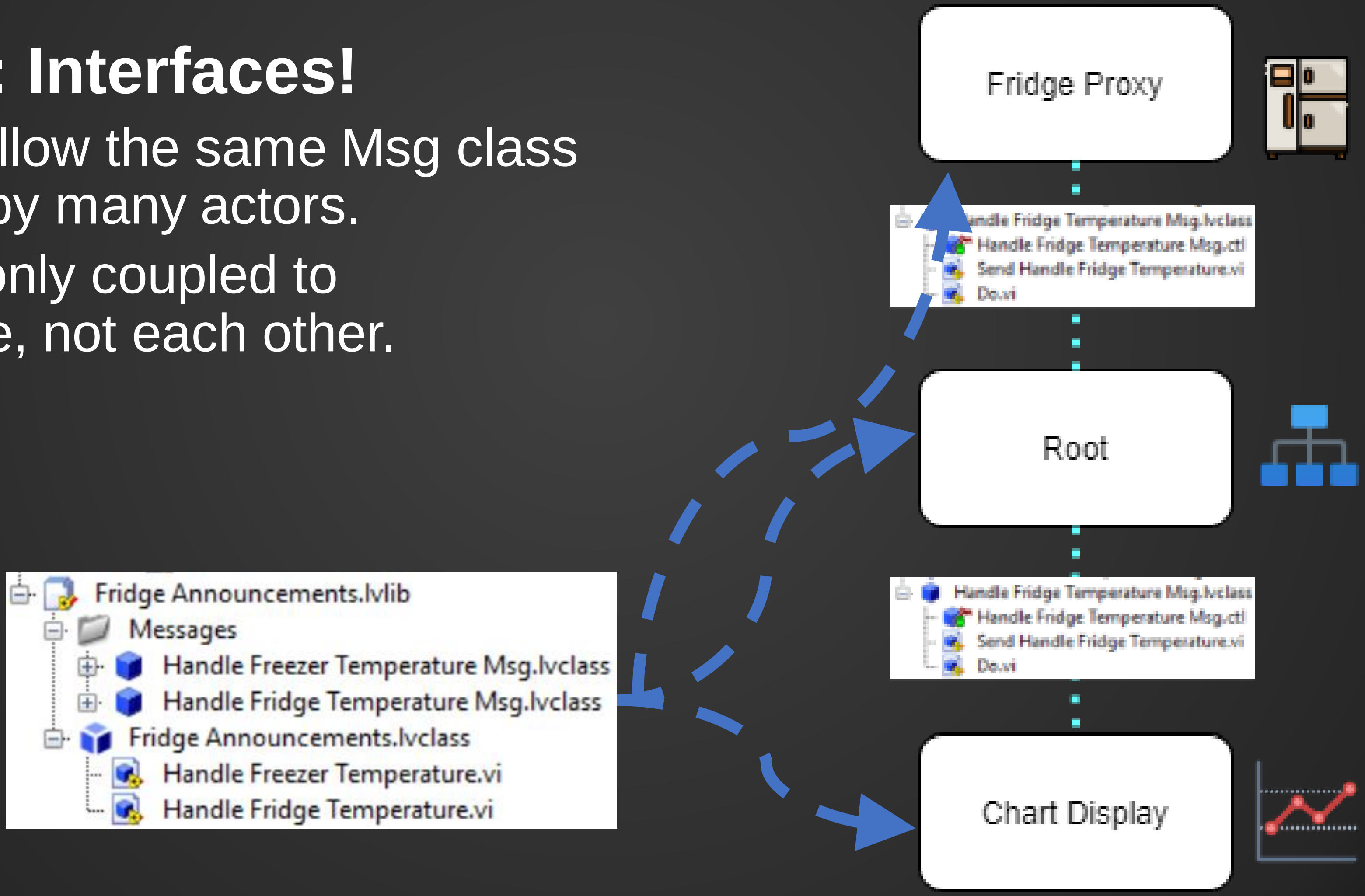
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi



Fridge Announcements.lvlib
Messages
Handle Freezer Temperature Msg.lvclass
Handle Fridge Temperature Msg.lvclass
Fridge Announcements.lvclass
Handle Freezer Temperature.vi
Handle Fridge Temperature.vi

How Can We Make This Better?

- **Solution #1: Interfaces!**
  - Interfaces allow the same Msg class to be used by many actors.
  - Actors are only coupled to the interface, not each other.



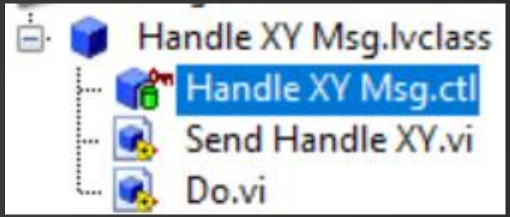
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi

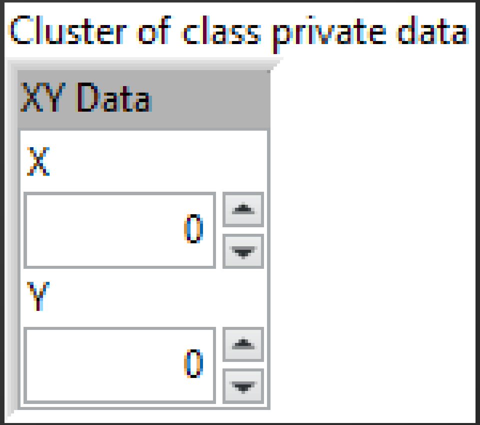
Handle Fridge Temperature Msg.lvclass
Handle Fridge Temperature Msg.ctl
Send Handle Fridge Temperature.vi
Do.vi

Fridge Announcements.lvlib
Messages
Handle Freezer Temperature Msg.lvclass
Handle Fridge Temperature Msg.lvclass
Fridge Announcements.lvclass
Handle Freezer Temperature.vi
Handle Fridge Temperature.vi

How Can We Make This Better?

- **Solution #1: Interfaces!**
  - Interfaces allow the same Msg class to be used by many actors.
  - Actors are only coupled to the interface, not each other.
  - Interface Msgs remain strictly typed.

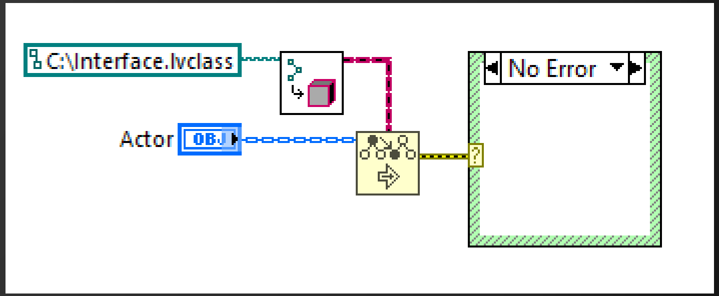

Handle XY Msg.lvclass
Handle XY Msg.ctl
Send Handle XY.vi
Do.vi


Cluster of class private data
XY Data
X
0
Y
0

How Can We Make This Better?

- **Challenge #2: How do we know which Actors support the different Msg classes?**
  - We need a generic way to test whether ANY Actor inherits from ANY other Msg Interface.
  - We need this to know which Actors to forward Msgs to.

- **Solution #2**
  - Using the <Preserve Run-Time Class> function, we can determine if an Actor supports the Msg being forwarded.
  - Note that <To More Specific Class> will NOT work here.



How Can We Make This Better?

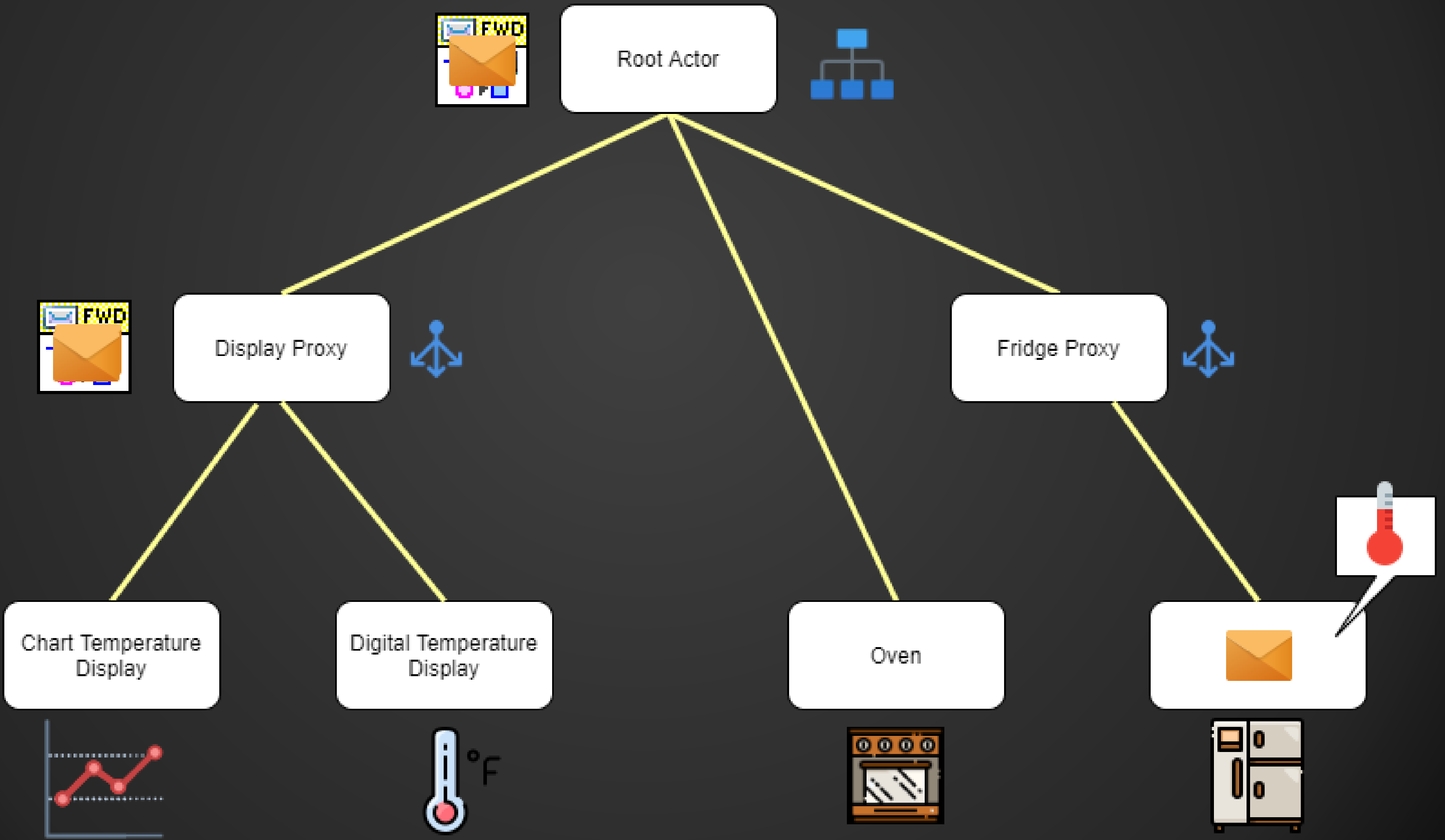
- **Challenge #3: How can we route the Msgs automatically while still adhering to the AF tree?**
  - We have re-usable Msgs that are a part of an interface.
  - We can determine if an Actor inherits from a Msg’s interface.
  - How can we use those two pieces of information to automatically route messages?

How Can We Make This Better?

- **Solution #3a: Route Msgs using a map of enqueuers**
  - When an Actor receives a Msg, determine the Msg's interface and look-up (e.g. with a ***map***) the enqueuers of the nested Actors that support that Msg.
  - But how is that map populated?

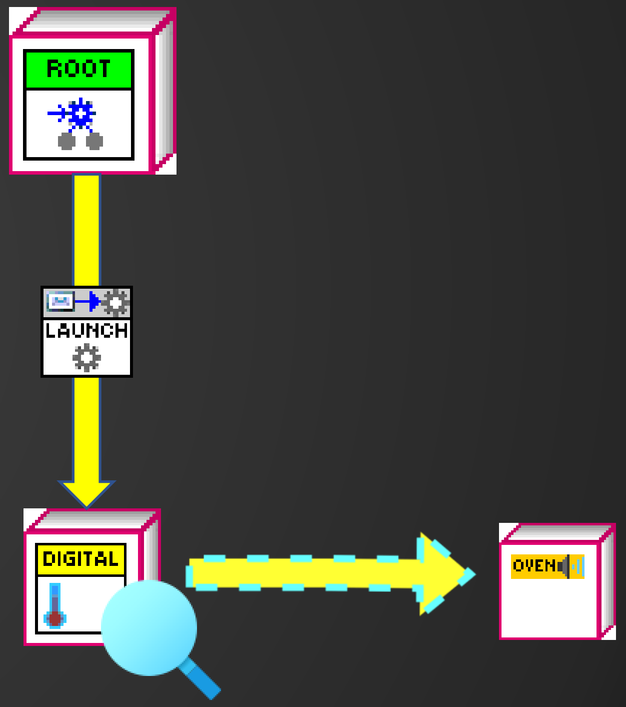
- **Solution #3b: Populate the map as nested Actors are launched**
  - If there were list of interfaces containing AF Msgs that could be forwarded, when an Actor is launched, we could check that to see whether that Actor supports that interface and, if so, store its enqueuer for look-up later.

Message Routing



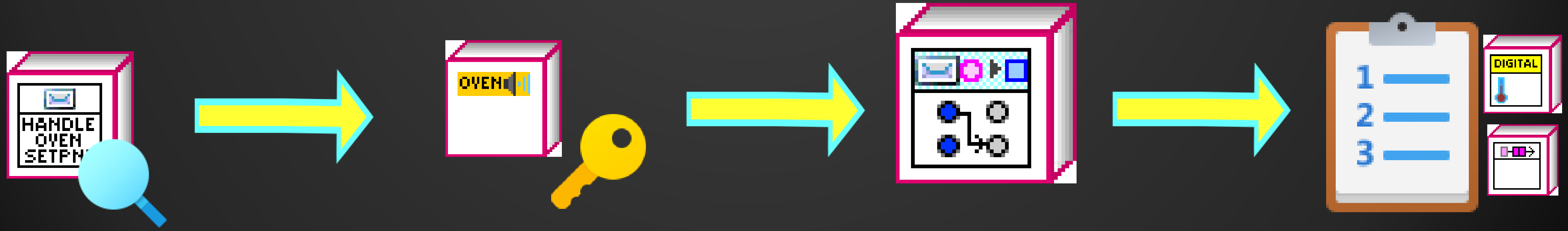
High Level

- Every time you launch a Nested Actor, figure out what interfaces it inherits from and add to map.



High Level

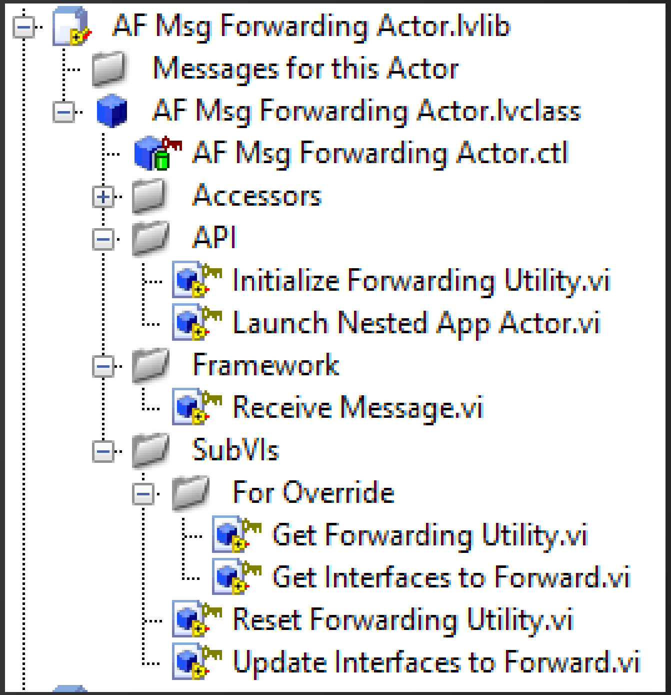
- Every time you launch a Nested Actor, figure out what interfaces it inherits from and add to map.
- Every time an Actor receives a Msg, check if it's from an interface of interest, if so, forward to any interested nested actors.



Demo

How It Integrates

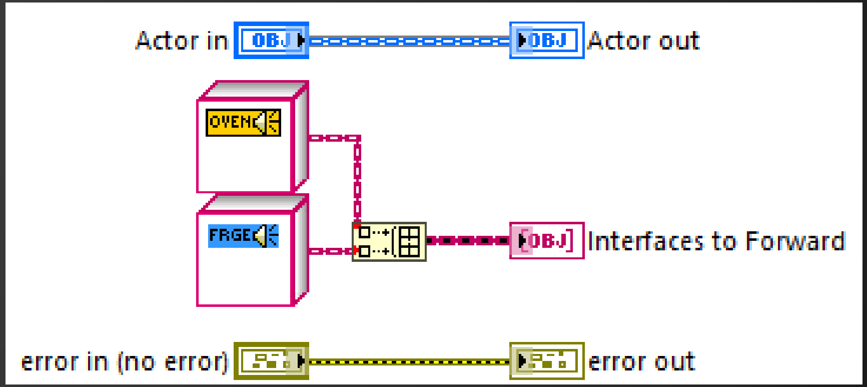
- All the pieces you need to start integrating this with your own AF application comes in a VIP*.
- Once installed, the easiest way to get started using this utility is to insert the “AF Msg Forwarding Actor” in your inheritance hierarchy.
  - A merge VI with this actor can be found on the Zyah palette.



- AF Msg Forwarding Actor.lvlib
  - Messages for this Actor
  - AF Msg Forwarding Actor.lvclass
    - AF Msg Forwarding Actor.ctl
    - Accessors
    - API
      - Initialize Forwarding Utility.vi
      - Launch Nested App Actor.vi
    - Framework
      - Receive Message.vi
    - SubVIs
      - For Override
        - Get Forwarding Utility.vi
        - Get Interfaces to Forward.vi
      - Reset Forwarding Utility.vi
      - Update Interfaces to Forward.vi

How It Integrates
Identify Interfaces to Forward

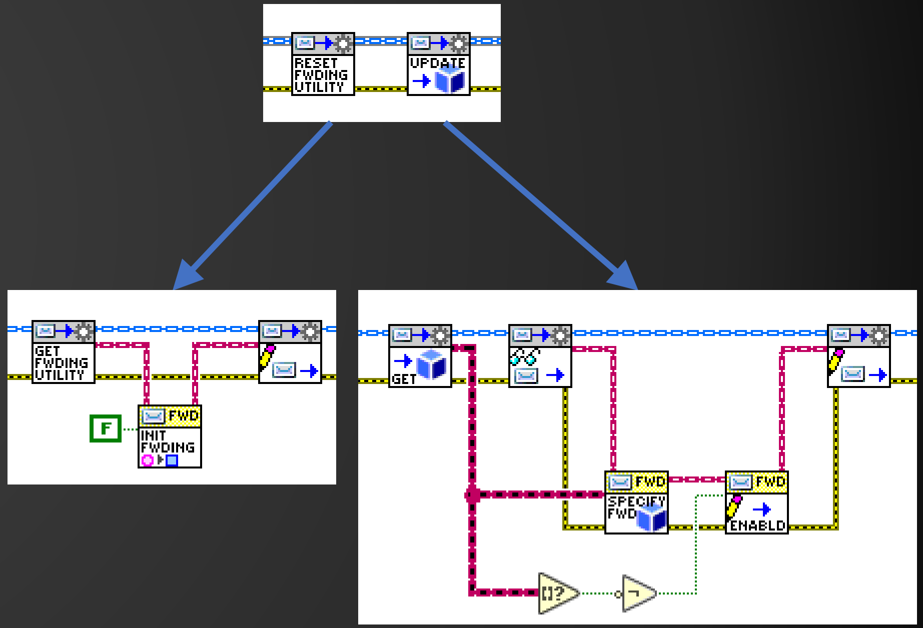
- Override <Get Interfaces to Forward> in each actor that directly inherits from “AF Msg Forwarding Actor” to enable forwarding.
  - Necessary to specify the which Msgs should be forwarded in the application (identified by their owning interfaces).
  - Identification can be hard-coded or done programmatically (e.g. for plug-ins).
  - We recommend using a base Actor (specific to your application) so you only have to override the VI once.



How It Integrates

AF Msg Forwarding Actor

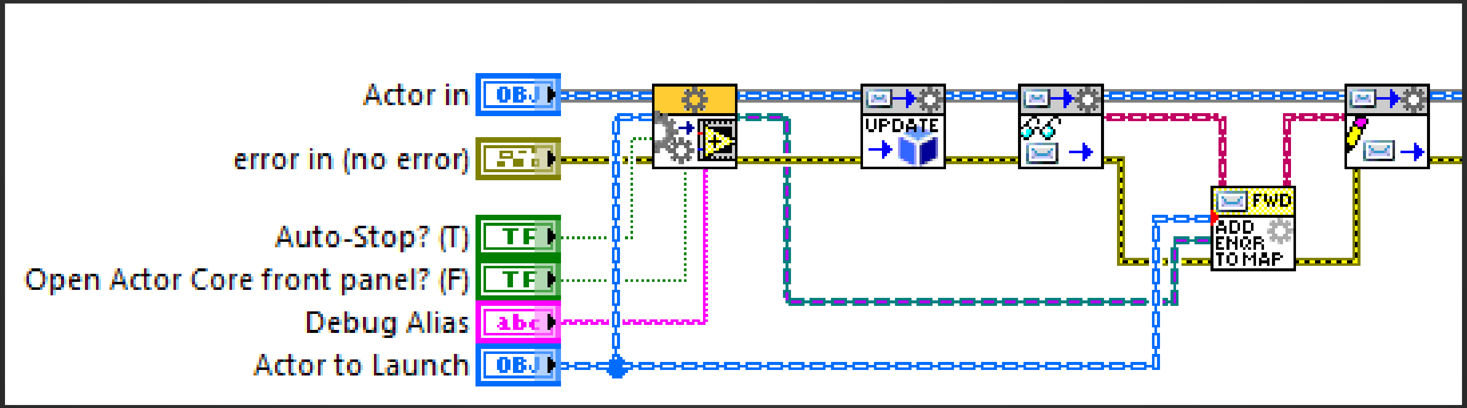
- API
  - <Initialize Forwarding Utility.vi>
    - Used to identify the Forwarding Utility and forwarding interfaces for the given application (using <Get Forwarding Utility> and <Get Interfaces to Forward>).
    - This VI should be called before any nested actors are launched to establish and clear the ***map***.
      - In a common initialization method or override of <Pre Launch Init> could reduce code duplication.



How It Integrates

AF Msg Forwarding Actor

- API
  - <Launch Nested App Actor.vi>
    - Calls the normal <Launch Nested Actor.vi> and populates the forwarding ***map*** with the newly launched actor.



How It Integrates

- At this point, all announcement Msg routing is determined by inheritance.
- Payload method overrides must be created separately.
- Actors that are forwarding the message *up* the AF Tree need to have forwarding enabled, but do NOT need to inherit from the Msg owning interfaces.

Recap/Benefits

- One transport method that adheres to the AF message tree guidelines.
- Automatic message routing via inheritance (i.e. less work for developers).
  - Easy to change routing without needing to modify block diagrams.
  - This is ESPECIALLY helpful for plug-ins.
- Small code footprint – fewer Msgs and overrides needed (especially if wrapped into a Base Actor).
- Root Actor never needs to change to accommodate more forwarding Msgs.

Future Features/Enhancements

- Compatibility with AF PPLs.
- This hasn’t been tested on targets other than a PC – might need to modify to work with RT, etc.
- Separate maps for what gets forwarded up the tree vs down rather than relying on (lack) of interface inheritance.*

Where to Get It

- Available as a VI package or as source code
  - VIPM (Easiest)
  - https://gitlab.com/zyah-solutions/af-msg-forwarding-utility
- Blog post coming soon! (http://www.zyahsolutions.com)
- Need help?
  - Email us: info@zyahsolutions.com
  - Discord server link on the website

Questions?

Appendix

Technical Details

Notes on Routing

- By default, if the Msg is not implemented by the receiving Actor, but the Msg is identified as a forwarding message, it will send the Msg the Actor’s *caller* (i.e. up the AF Tree).*
- Msgs only get forwarded to nested actors if the receiving Actor inherits from the Msg owning interface or it is the root actor.
    - This is done to avoid infinite forwarding between an actor that supports the message and its caller.

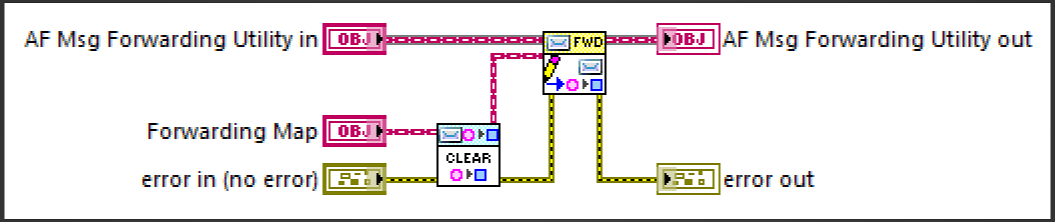
Zyah AF Msg Forwarding Utility

- Overview
  - Part 1 (the Zyah AF Msg Forwarding Utility class)
    - Based off inheritance, we can tell whether a Msg is supported by an Actor.
    - We use the Msg's owning library as a ***key*** to create a ***map*** of enqueuers that belong to nested Actors that support that message.
  - Part 2 (integrating the class with an AF application)
    - Establish which Actor announcements should be forwarded by way of their owning libraries.
    - Route the messages using an override of **Receive Message.vi**

How it Works

Initializing the Forwarding Map

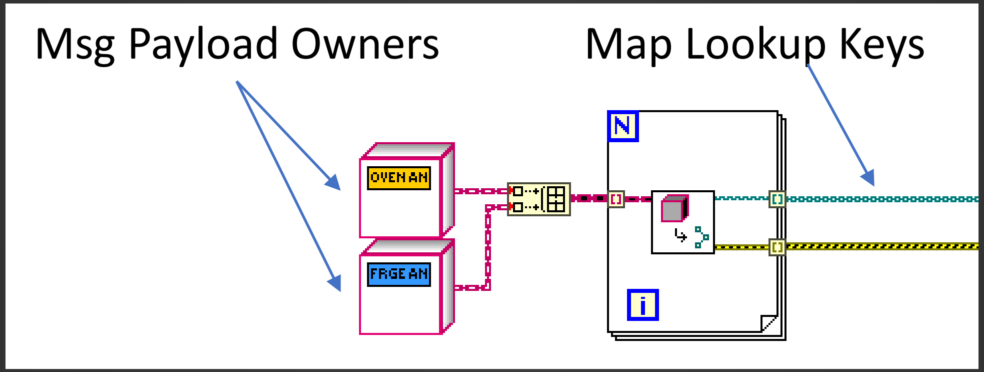
- <Initialize Forwarding Map.vi>



  - Must be called first.
  - Establishes which type of ***map*** to key to use (paths in this example).
    - Paths are the ***keys***.
    - A ***set*** of Actor enqueuers are the map ***values***.
  - Stores the map as a part of the forwarding utility.
  - This is the first of two *must override* Vis.

How it Works
Identifying Msgs to Be Forwarded

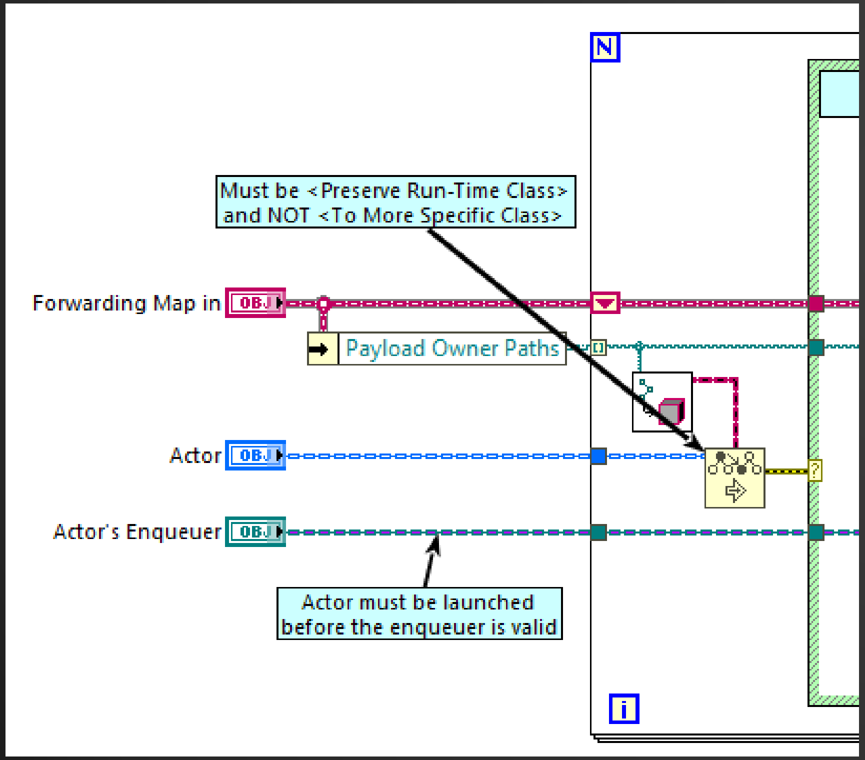
- <Identify Msg Payload Owners.vi>



  - This is the second and final *must override* VI.
  - All Msgs in a given library will be forwarded.
  - You can change the lookup keys if desired.
  - The ***keys*** of the ***map*** are also stored as a ***set***.

How it Works
Adding Actors to the Map

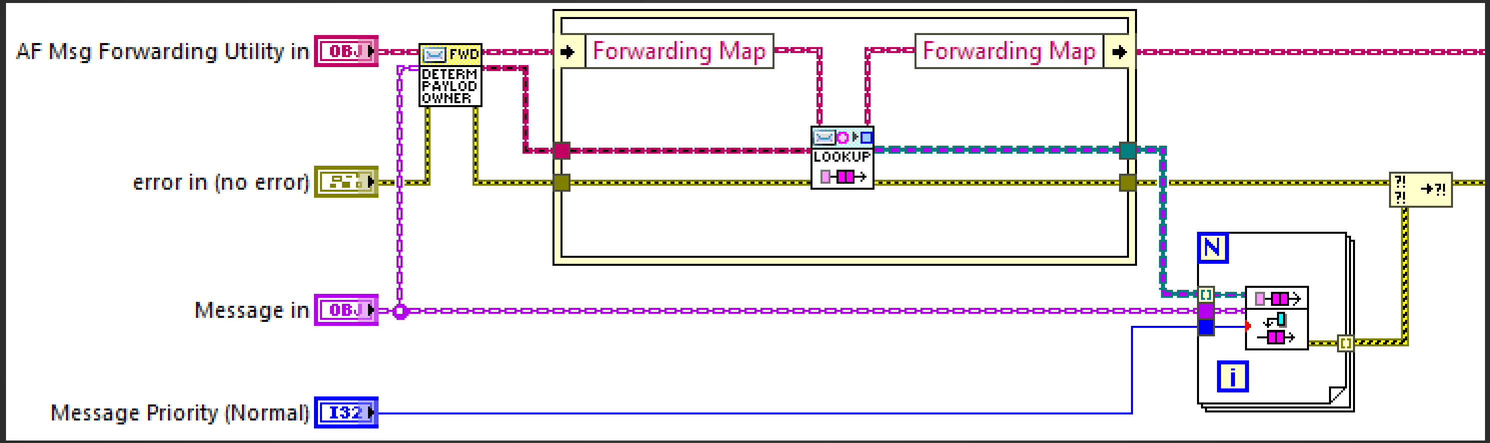
- <Add Actor Enqueuer to Map.vi>


Must be <Preserve Run-Time Class> and NOT <To More Specific Class>
Forwarding Map in
OBJ
Payload Owner Paths
Actor
Actor's Enqueuer
Actor must be launched before the enqueuer is valid
N
i

- When an Actor is launched, the Actor wire and its enqueuer is fed into this VI where it is checked for inheritance against the ***set*** of Msg Payload Owners that were previously shown.
- Actors that inherit from the given Msg Payload Owner are added to the ***map*** appropriately.
- This VI should be invoked every time a nested actor is launched.

How it Works
Forwarding Msgs

- <Forward Msg Using Map.vi>



  - When a Msg is received by an Actor (in <Receive Message.vi>), it uses this API to lookup the enqueuers of the Actors that support the message and forward it along accordingly.

How it Works
Recap

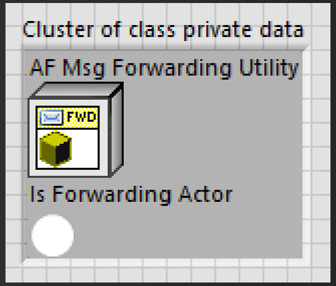
- <Initialize Forwarding Map.vi>
  - Used to set up the type of ***keys*** for the ***map*** and initialize values.
  - Must override.
- <Identify Msg Payload Owners.vi>
  - Used to identify application specific owning libraries containing Msgs to forward to other nested Actors.
  - Must override.

How it Works
Recap (continued)

- <Add Actor Enqueuer to Map.vi>
  - Used when launching a nested Actor to populate the forwarding map.
- <Forward Msg Using Map.vi>
  - Used when receiving a Msg to automatically forward to any nested actors that support it.

How it Integrates
Base Actor Private Data

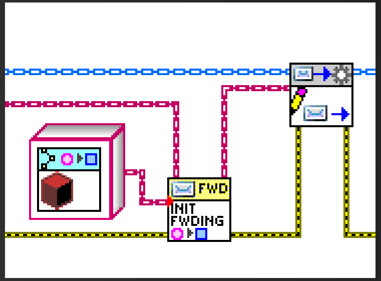
- The Forwarding Utility must be added to every actor that could be used to forward Msgs.
- Though not strictly necessary, to avoid duplicated code, we recommend adding the utility class (as well as an opt-in flag) to a Base Actor.
  - You could put it in your own or inherit it from one Zyah provides.
- Accessor VIs for each item should also be created within the Base Actor.

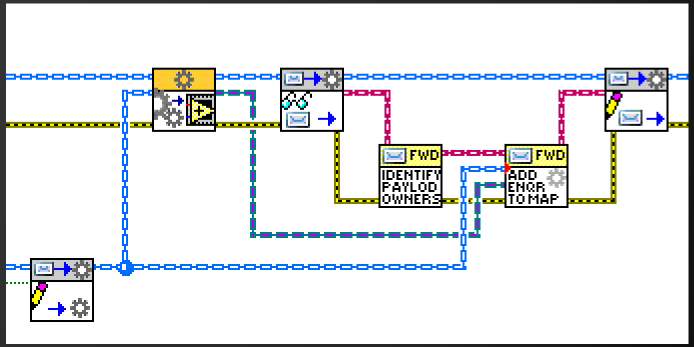


How it Integrates

Base Actor API (Protected)

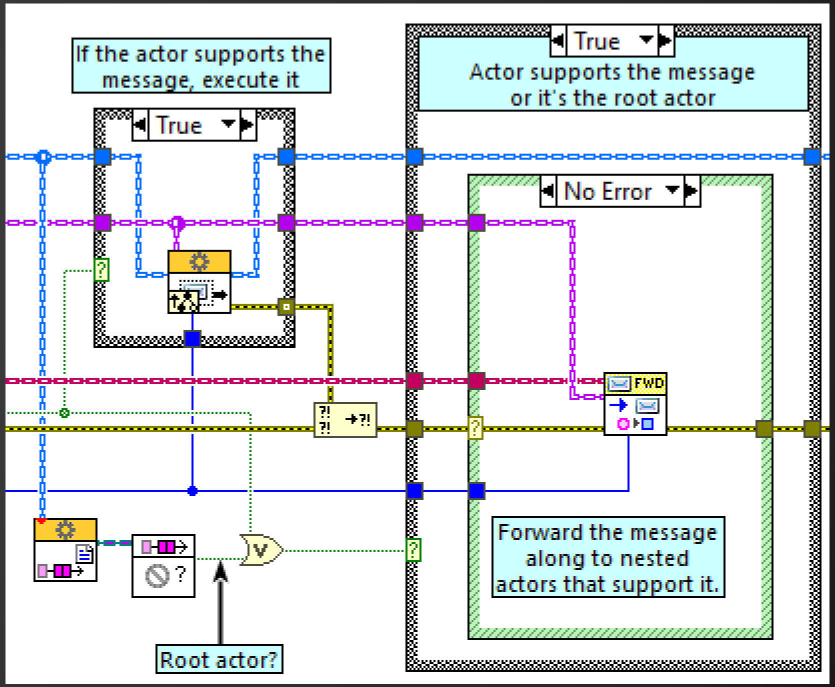
- <Initialize Forwarding Map.vi>
  - Specifies which ***map*** to use along with which child instance of the Forwarding Utility and then calls the VI of the same name from within the Forwarding Utility.
  - Stores the ***map*** in the Base Actor’s private data.
- <Launch Nested App Actor.vi> (recommended)
  - Wraps normal <Launch Nested Actor.vi> functionality with ***map*** population functions from the Forwarding Utility.
  - Populates “Is Forwarding Actor” parameter.





How it Integrates
Base Actor Override

- <Receive Message.vi>
  - Verifies that the Msg and Actor support forwarding (not shown).
  - If the current actor implements the message, executes it.
  - If the actor does *not* implement the message and it's not the root actor, forwards the message up the Msg tree.
  - Otherwise forwards the Msg to all the supported nested Actors.

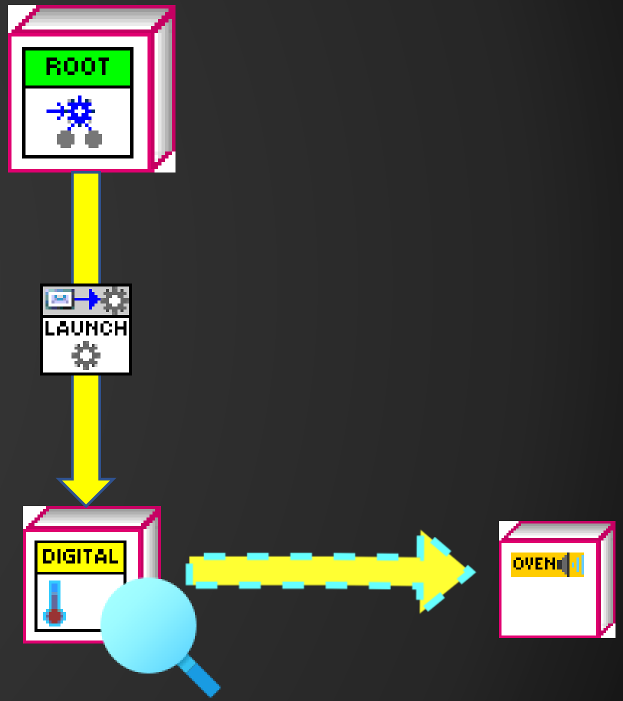

If the actor supports the message, execute it
True
True
Actor supports the message or it's the root actor
No Error
FWD
Forward the message along to nested actors that support it.
Root actor?

Actor Framework:
The Broccoli of LabVIEW

- Steep learning curve
- Challenging to debug
- IDE slowdown
- Refactoring can be painful
- AF messages force tight coupling
- Tooling is lacking for common operations
- Messaging in large hierarchies can be painful

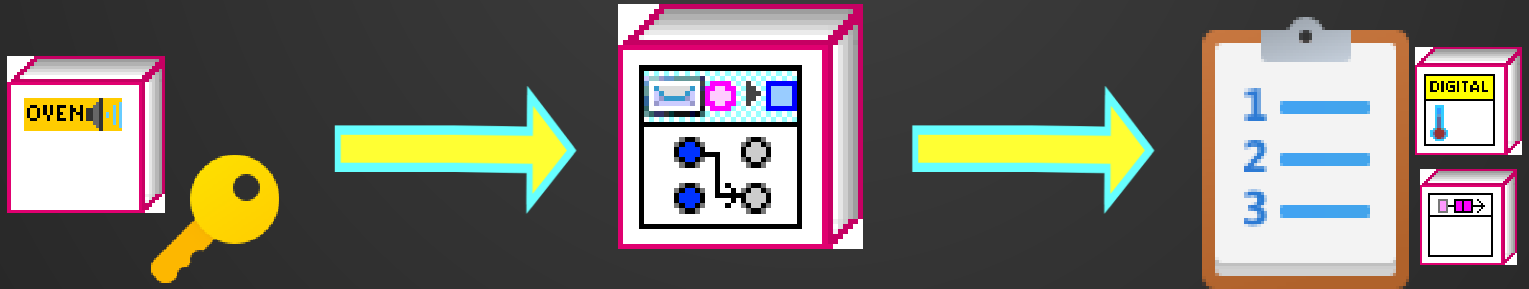
Launching and Routing

- Root launches Digital Temperature Display.
- During launch, Root determines Digital Temperature Display's parent interfaces.
- In this case, Digital Temperature Display inherits from the Oven Announcements interface.



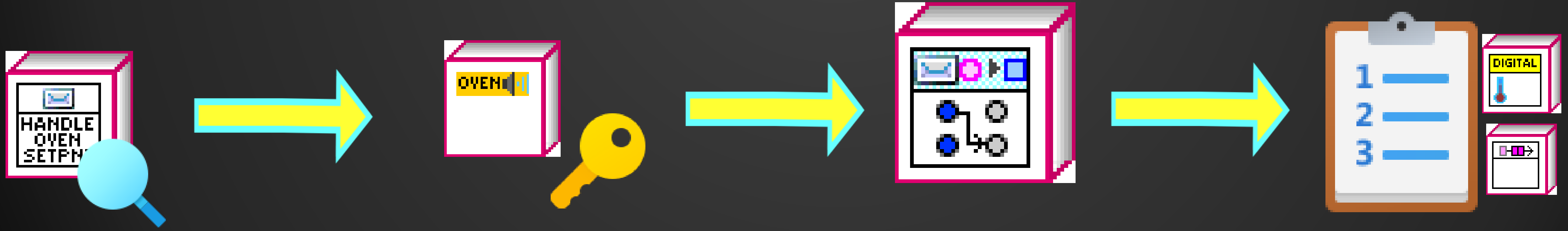
Launching and Routing

- Digital Temperature Display’s enqueuer then gets added to the Forwarding Map.
- When you pass Oven Announcements in as a key, all Nested Actor enqueuers that inherit from that interface are returned.

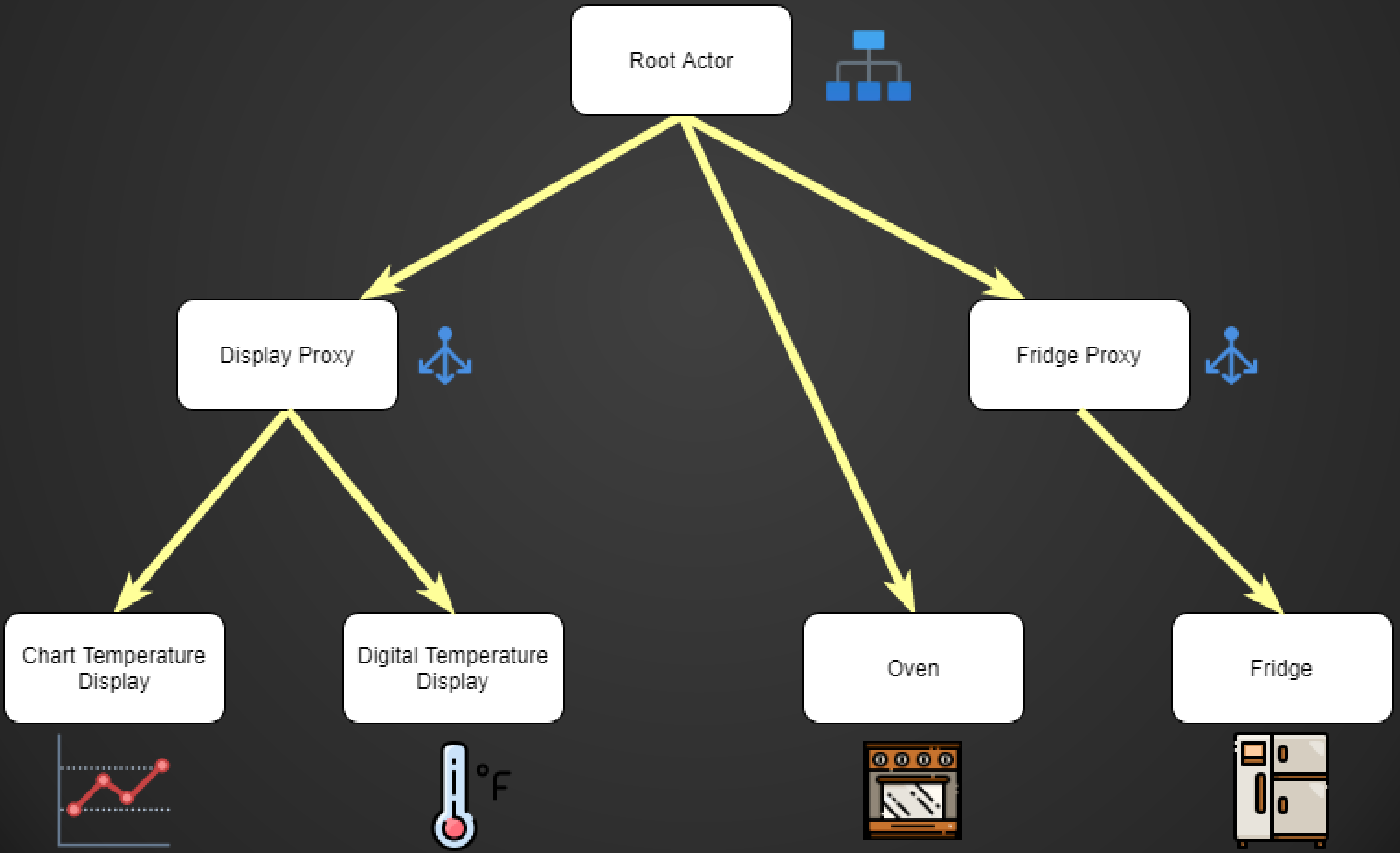


Launching and Routing

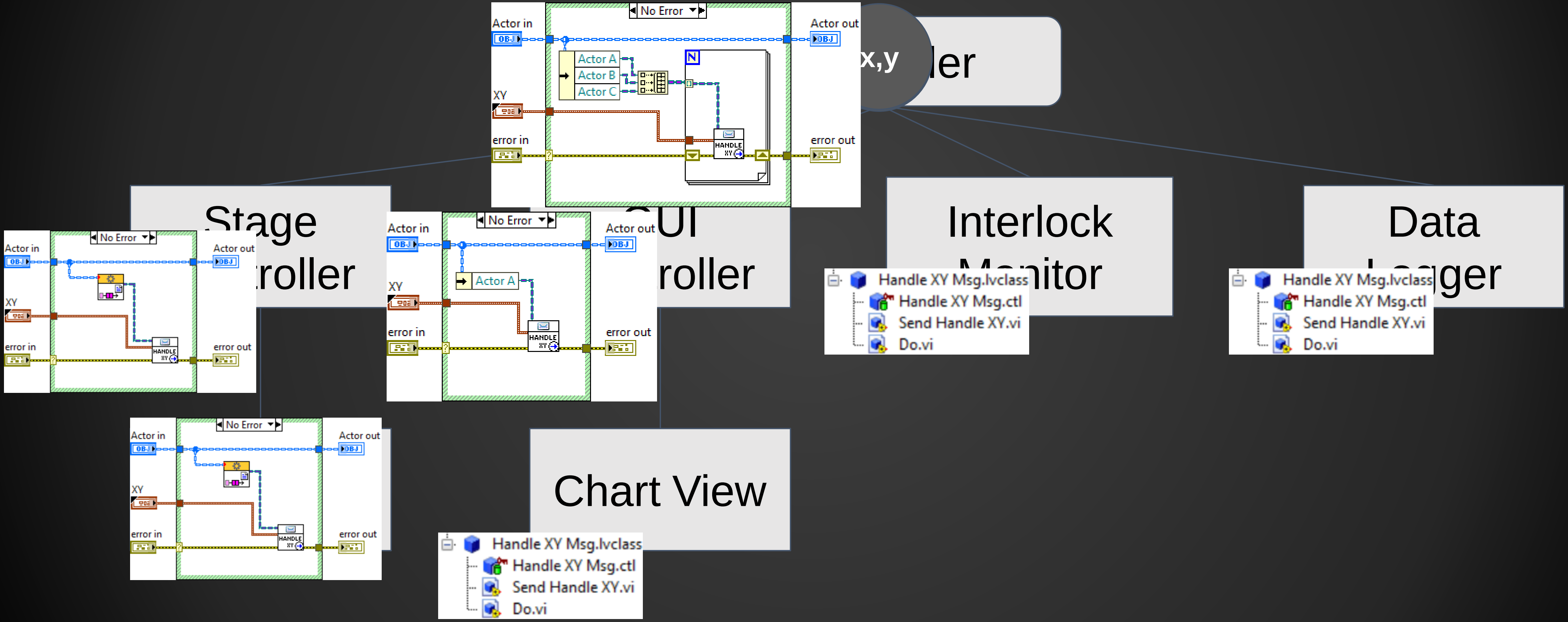
- When Display Proxy receives a Message, it determines the Message’s owning interface.
- The owning interface is used as the map key and the Message is forwarded to any matching enqueuers.



Launch Diagram

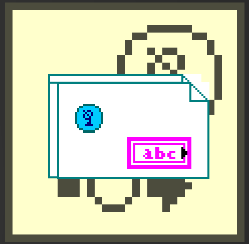


Traversing the Tree

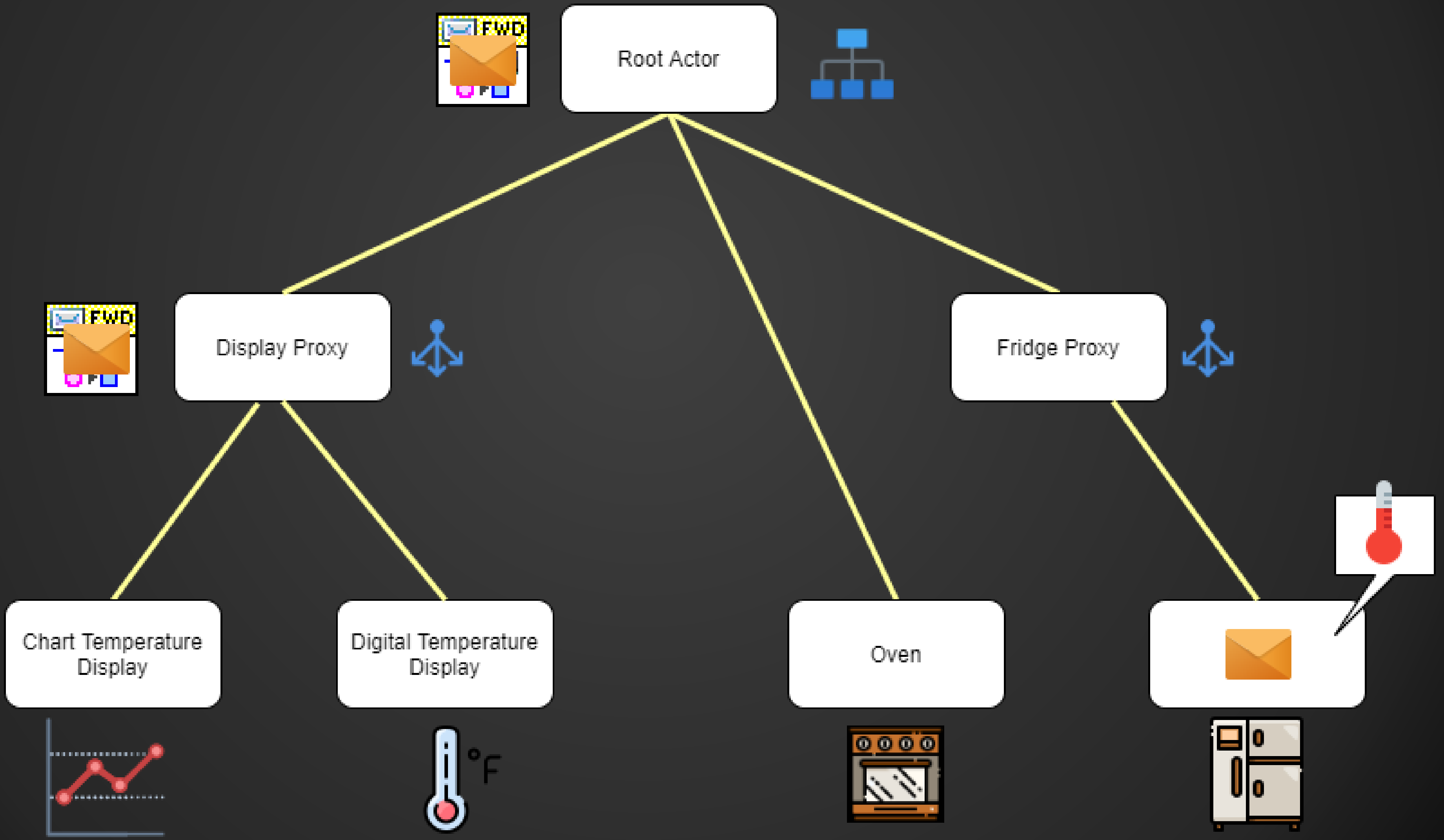


Traditional Solutions

- Circumventing the actor tree using alternate messaging such as user events, queues, etc.
  - –Messaging are routed through multiple transport mechanisms, which can complicate debugging.
  - –Code clarity and readability often suffers with multiple transport mechanisms.
  - –Competing transport mechanisms increases potential for timing/locking issues.



Message Routing



Zyah AF Msg Forwarding Utility

- Interfaces (solution 1) are included out of the box in LabVIEW 2020+.
- Determining interface inheritance and map population (solutions 2 and 3) are wrapped up into the Zyah AF Msg Forwarding Utility.
  - Available today via VIPM.