GLA Summit 2020

Broker Framework

Felipe Pinheiro Silva – CLA

(PTI – Foz do Iguaçu - Brazil)

2020 - Nov 9th and 10th

Agenda

- **What is and what is not this presentation…**
- Broker “Framework” Idea
- Message Patterns and the Broker Concept
- Abstract Message/Interfaces
- Implementation
- Future Work

What is and what isn’t this presentation…

- ✓ Explain the concept of the “broker” idea
- ✗ Teach you to start using it right away
- ✗ Force you to use this pattern
- ? Maybe build your own
- ✓ Share example code

Agenda

- What is and what is not this presentation…
- **Broker "Framework" Idea**
- Message Patterns and the Broker Concept
- Abstract Message/Interfaces
- Implementation
- Future Work

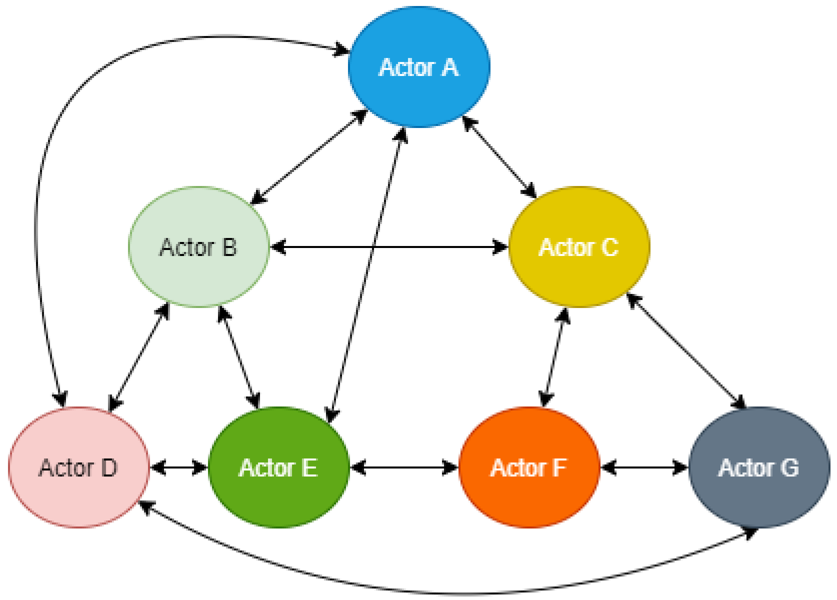
The origin of the “broker idea”

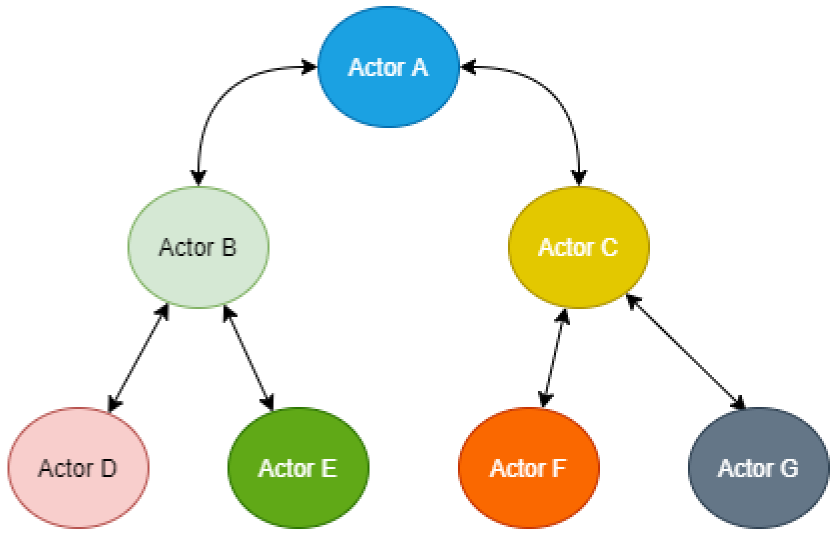
- Extensive use of the Actor Framework
  - Most of the projects using AF

- Search for an easy messaging system in Actor Framework
  - Avoiding the coupling between different modules (actors) and emphasizing code reuse

The origin of the “broker idea”

- Avoid harming the Actor Framework Tree message
  - Keeps the hierarchy of Caller – Nested actors
  - Many discussions on the forums





Broker Requirements

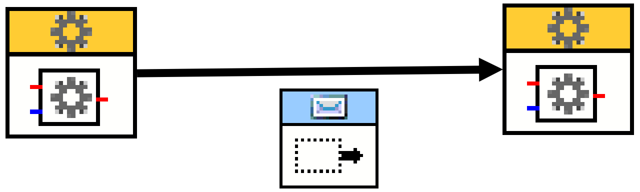
- Simple to use
  - Minimum AF knowledge
  - Interfaces helped to achieve this!!!!!
- Enforce decoupling – mainly between sibling actors
  - Consequently Code Reuse
- Allow Nested “brokers”
  - Allow the pattern to be reproduced down the tree
- Provide minimum debugging tools
  - Screen with message sniffing
- Works on Windows, RT …

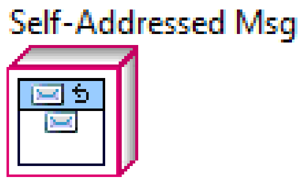
Agenda

- What is and what is not this presentation…
- Broker “Framework” Idea
- **Message Patterns and the Broker Concept**
- Abstract Message/Interfaces
- Implementation
- Future Work

Message Patterns

- Asynchronous
  - Traditional idea of Actor Framework
  - Most of the designs is towards this pattern

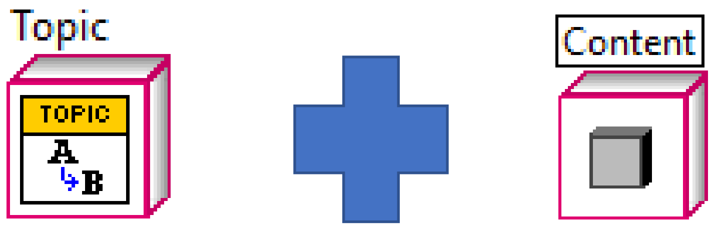




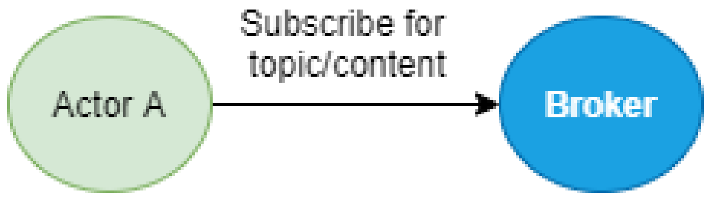
- Synchronous
  - Not recommended in Actor Framework
  - Blocking Communication

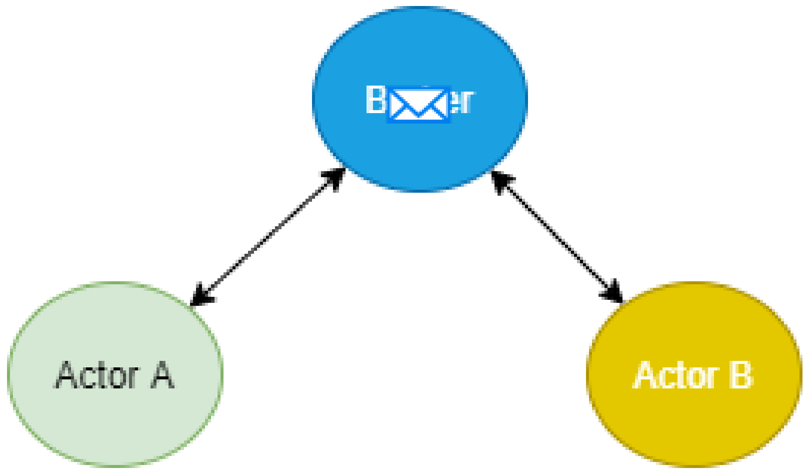
Broker Concept

- Message Distribution
- Asynchronous
- Publish and Subscribe
- Loose Coupling
- Widely used: internet protocols, MQTT, AMQP, Kafka, etc…
- Two types of message filtering:



Broker Concept





Agenda

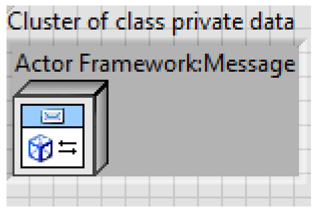
- What is and what is not this presentation…
- Broker "Framework" Idea
- Message Patterns and the Broker Concept
- **Abstract Message/Interfaces**
- Implementation
- Future Work

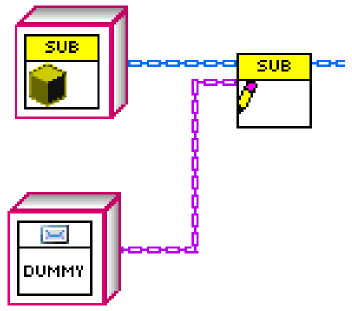
Abstract Messages

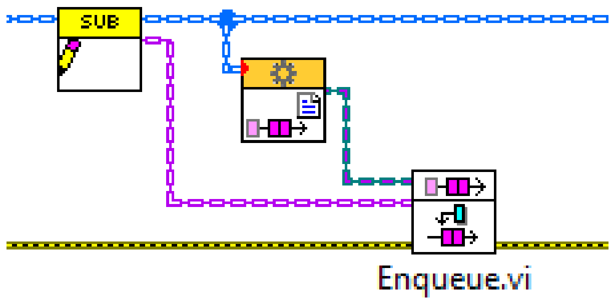
- Standard way of decoupling actors <= LV2019
- First implementation of “my broker”
- Uses a Common Abstract Message Class or the AF Message Class
  - Children of the class used in runtime

Actor Framework:Message

- Needs to inform the Message Class during startup or load dynamically in runtime.





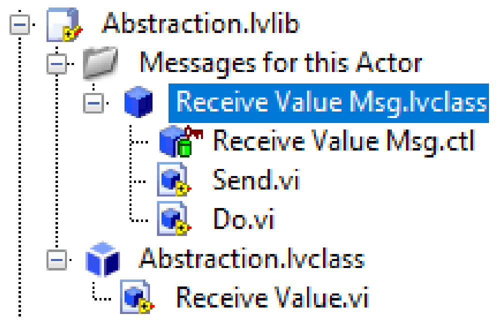


Interfaces

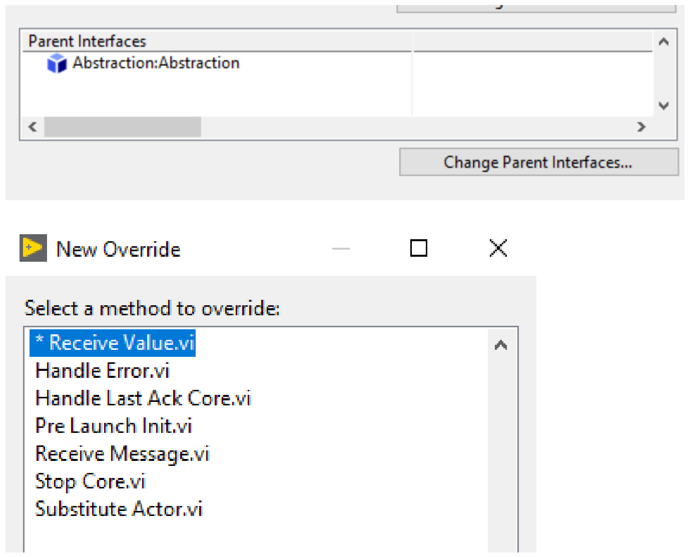
- Simplify drastically the decoupled messaging pattern

  "This technique completely replaces the abstract message pattern used in LV 2019 and earlier."

- Updated the implementation of "my broker" pattern
- Receiver is a child of an Interface



```
Abstraction.lvlib
  Messages for this Actor
    Receive Value Msg.lvclass
      Receive Value Msg.ctl
      Send.vi
      Do.vi
  Abstraction.lvclass
    Receive Value.vi
```



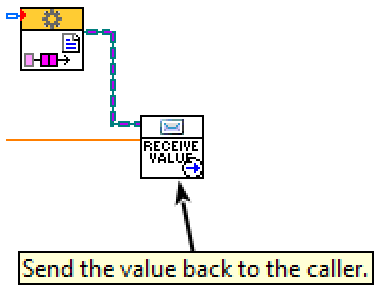
Parent Interfaces
Abstraction:Abstraction

Change Parent Interfaces...

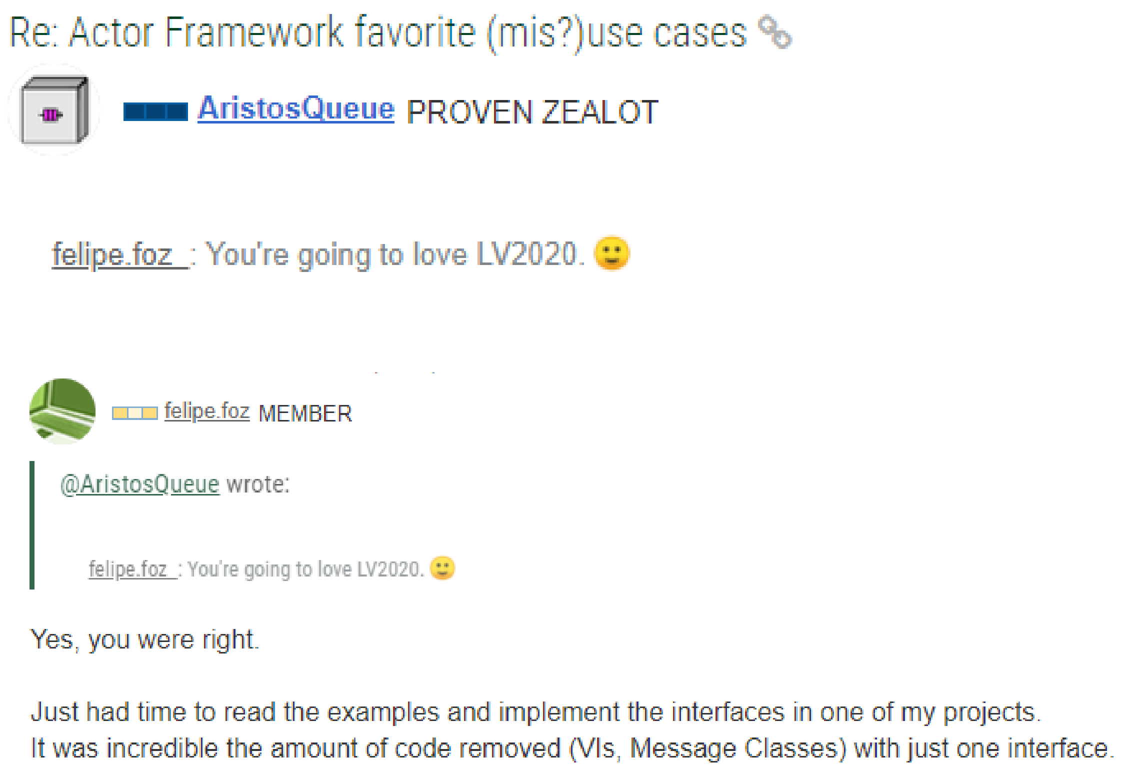
New Override

Select a method to override:

* Receive Value.vi
Handle Error.vi
Handle Last Ack Core.vi
Pre Launch Init.vi
Receive Message.vi
Stop Core.vi
Substitute Actor.vi



Interfaces



Re: Actor Framework favorite (mis?)use cases

AristosQueue PROVEN ZEALOT

felipe.foz : You're going to love LV2020. 🙂

felipe.foz MEMBER

> @AristosQueue wrote:
>
> felipe.foz : You're going to love LV2020. 🙂

Yes, you were right.

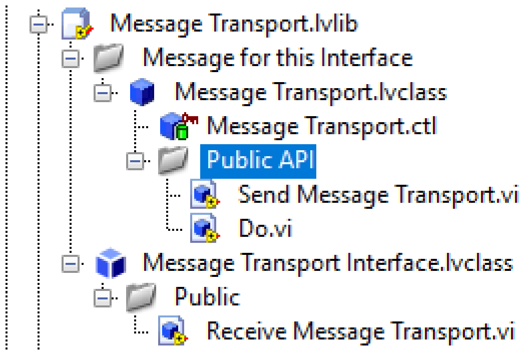
Just had time to read the examples and implement the interfaces in one of my projects.
It was incredible the amount of code removed (VIs, Message Classes) with just one interface.

Agenda

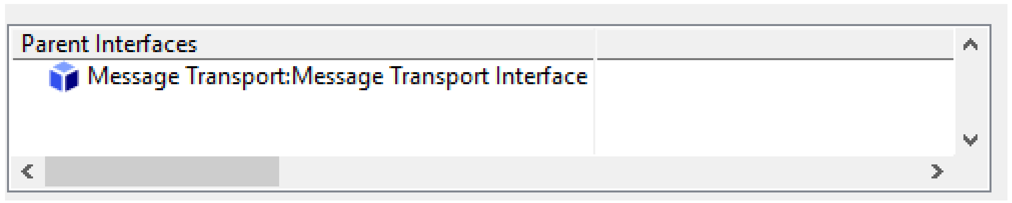
- What is and what is not this presentation…
- Broker “Framework” Idea
- Message Patterns and the Broker Concept
- Abstract Message/Interfaces
- **Implementation**
- Future Work

Implementation – Message and Subscriber

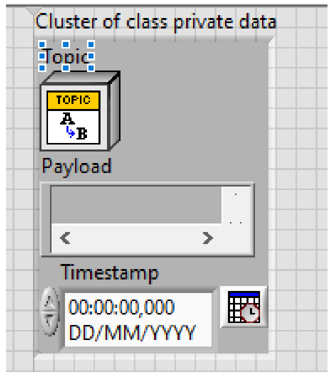
- Abstract Message with an Interface
  - Any class intending to receive msg (subscribe) must inherit from this interface
  - Override the Abstract Method (Receive Message Transport).



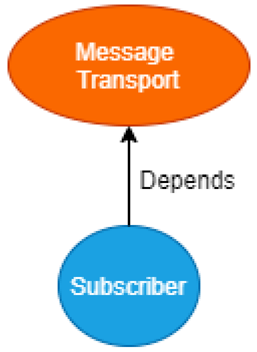
```
Message Transport.lvlib
  Message for this Interface
    Message Transport.lvclass
      Message Transport.ctl
      Public API
        Send Message Transport.vi
        Do.vi
  Message Transport Interface.lvclass
    Public
      Receive Message Transport.vi
```



Parent Interfaces
Message Transport:Message Transport Interface

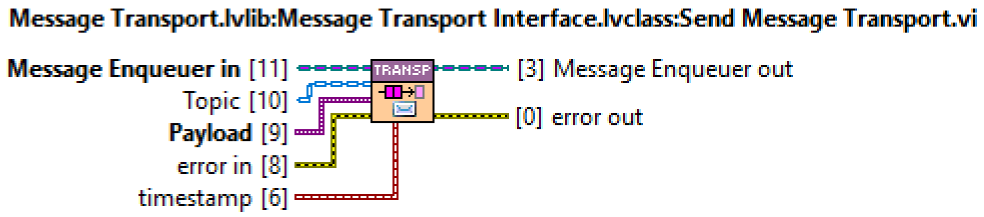


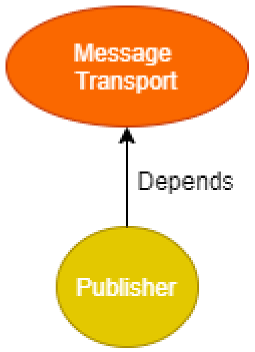
Cluster of class private data
Topic
Payload
Timestamp
00:00:00,000
DD/MM/YYYY



Implementation - Publisher

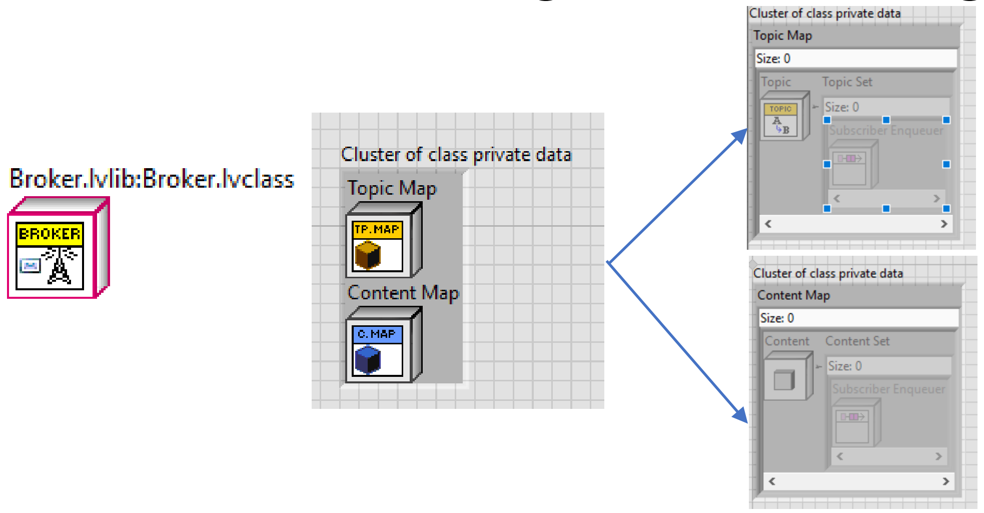
- Uses the Send from Message Transport
  - Sets a topic (optional with classes as payload)
  - Send a payload (variant)

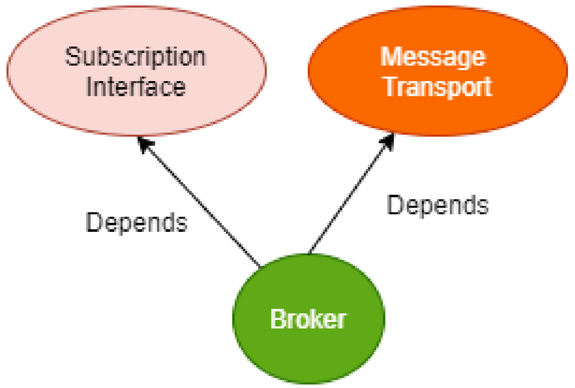




Implementation - Brokers

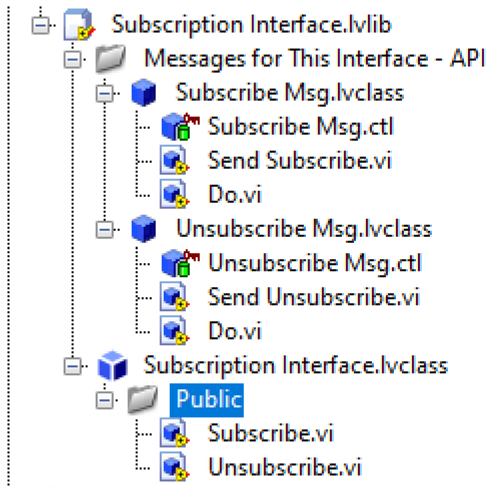
- Broker is a Parent Class
  - Any actor acting as broker (caller) must inherit from this parent class
  - “Nested” broker may also inherit to implement decentralized brokers
  - Other actors have no relation to the broker class
  - Runs the logic in the message forwarding using two maps.



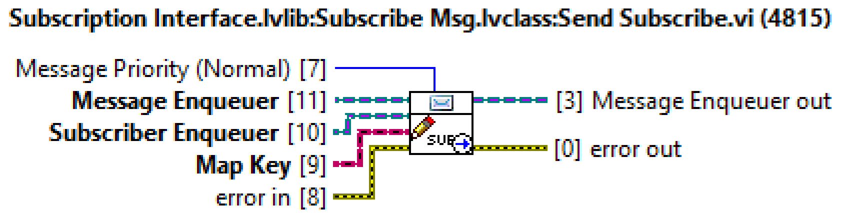


Implementation - Subscription

- Mandatory for brokers
  - Uses to subscribe/unsubscribe from topic or content
- Optional for other actors
  - Special Use Case: Subscribe or Unsubscribe on the Fly.



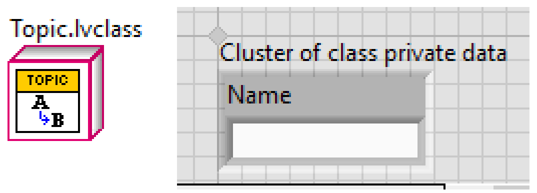
```
Subscription Interface.lvlib
    Messages for This Interface - API
        Subscribe Msg.lvclass
            Subscribe Msg.ctl
            Send Subscribe.vi
            Do.vi
        Unsubscribe Msg.lvclass
            Unsubscribe Msg.ctl
            Send Unsubscribe.vi
            Do.vi
    Subscription Interface.lvclass
        Public
            Subscribe.vi
            Unsubscribe.vi
```

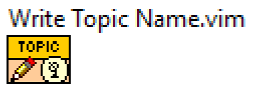


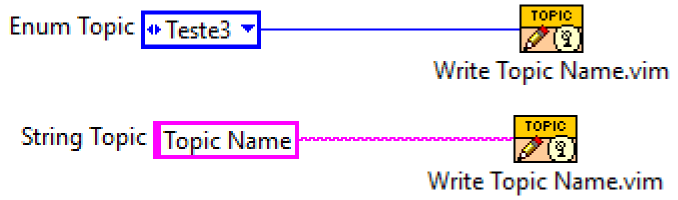
Subscription Interface.lvlib:Subscribe Msg.lvclass:Send Subscribe.vi (4815)

Implementation - Topic

- Encapsulate a string
- Vim accepts:
  - String or Enum
- Converts all to a String
- Enums are encouraged (mistyping)

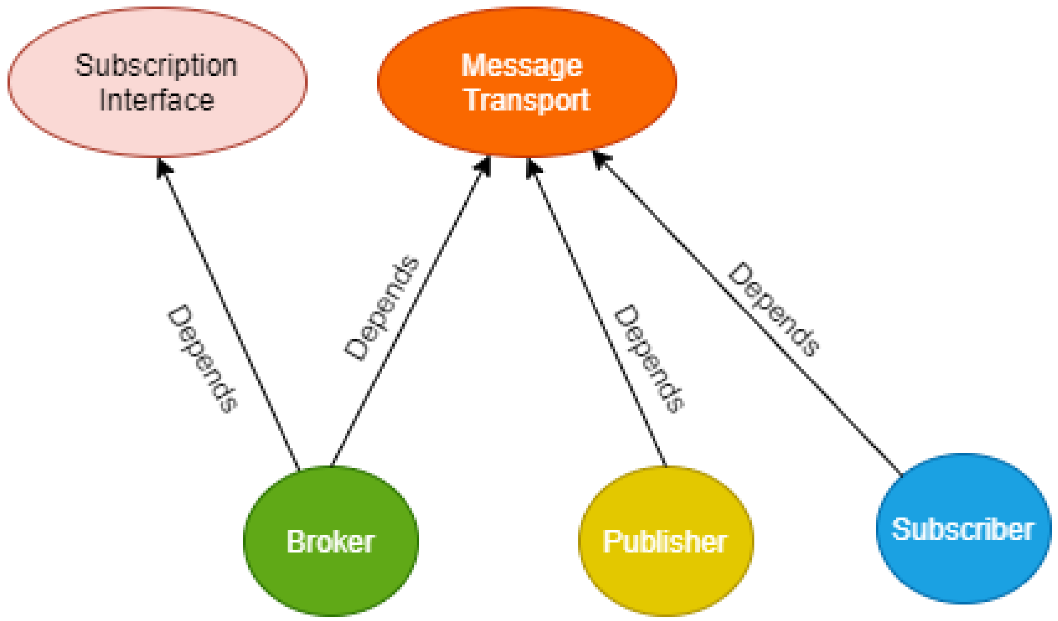


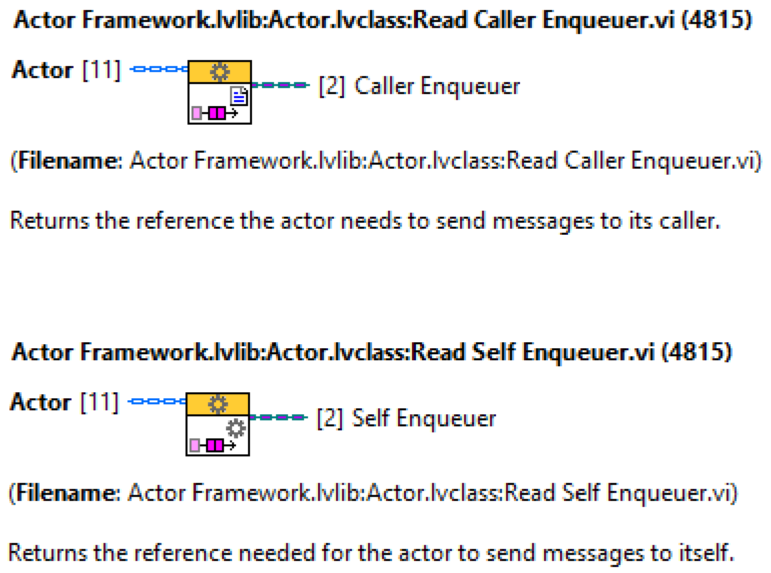




Implementation – Dependency Tree

- Dependency Inversion
- Not directly storing enqueuers





Actor Framework.lvlib:Actor.lvclass:Read Caller Enqueuer.vi (4815)

Actor [11] → [2] Caller Enqueuer

(**Filename**: Actor Framework.lvlib:Actor.lvclass:Read Caller Enqueuer.vi)

Returns the reference the actor needs to send messages to its caller.

Actor Framework.lvlib:Actor.lvclass:Read Self Enqueuer.vi (4815)

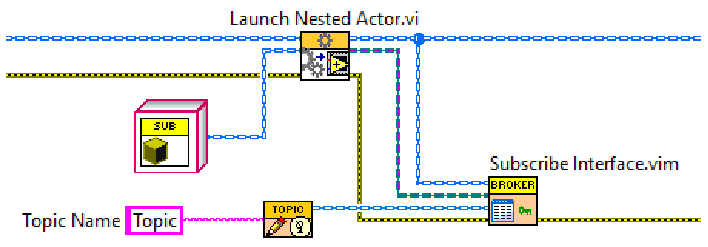
Actor [11] → [2] Self Enqueuer

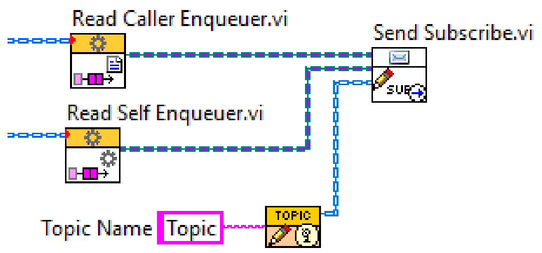
(**Filename**: Actor Framework.lvlib:Actor.lvclass:Read Self Enqueuer.vi)

Returns the reference needed for the actor to send messages to itself.

Implementation – Use Case 1

- A subscriber wants to receive information about a topic
- It subscribes to the topic using:
  - Caller subscribes it during any of its tasks
  - Send method

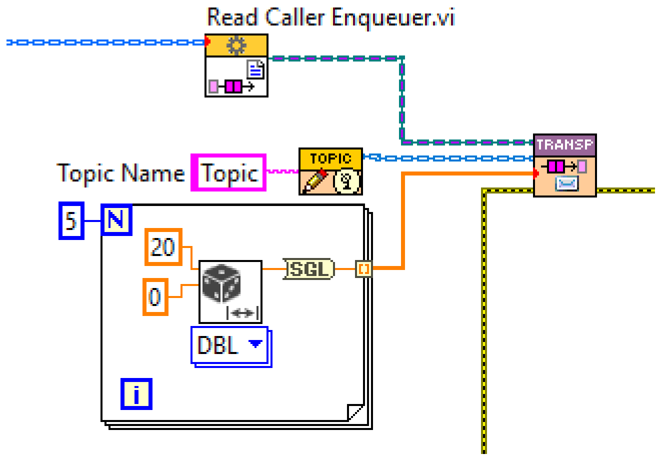


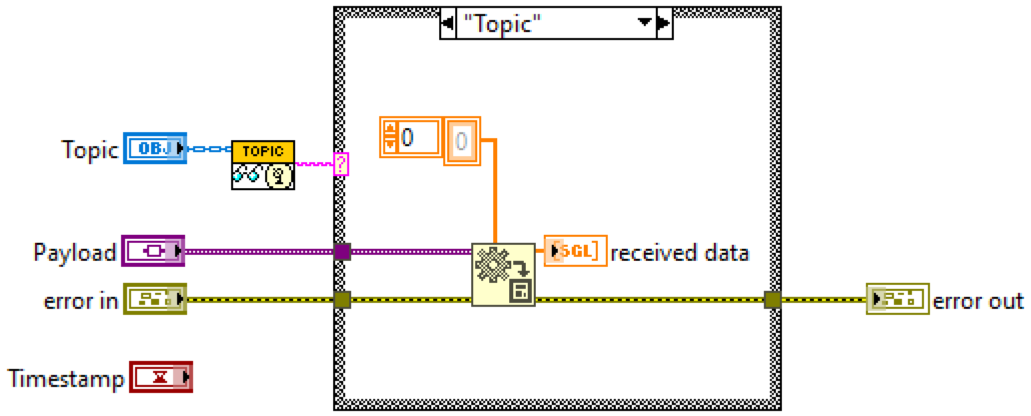


- Same for Unsubscribing

Implementation – Use Case 1

- Publisher Send a message with topic name “topic”
- Broker looks in map and forwards the message
- Subscriber receives and convert to use.

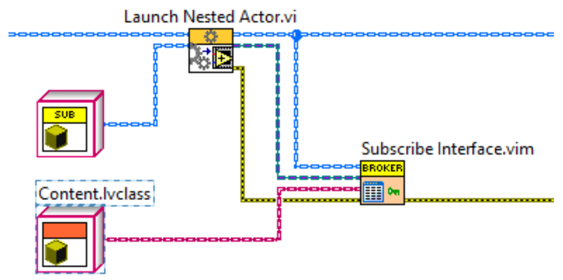


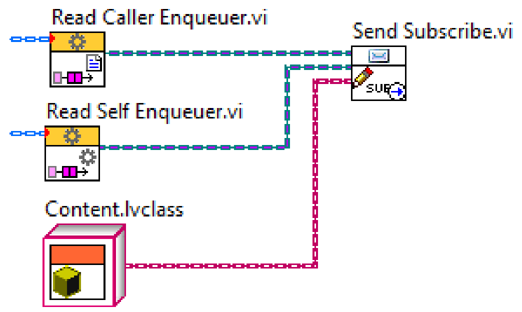


- Both must agree on the data type.

Implementation – Use Case 2

- A subscriber wants to receive information about a content (lvclass)
- It subscribes to the content using:
  - Caller subscribes it during any of its tasks
  - Send method

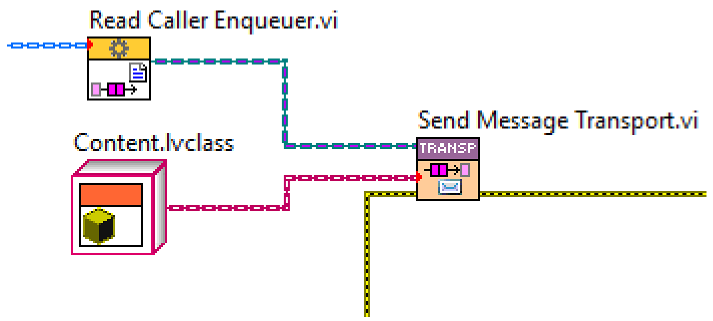


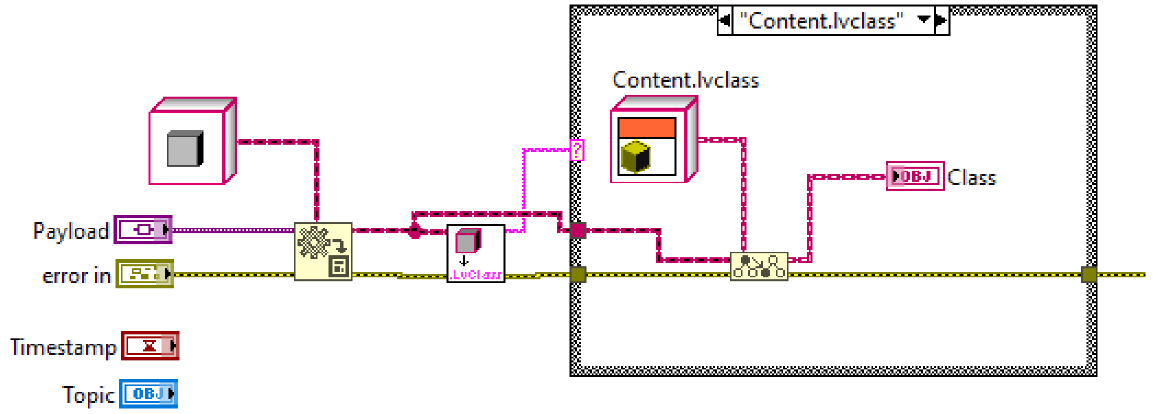


- Same for Unsubscribing

Implementation – Use Case 2

- Publisher Send a message with a content (lvclass)
- Broker looks in map and forwards the message.
- Subscriber receives and convert to use.





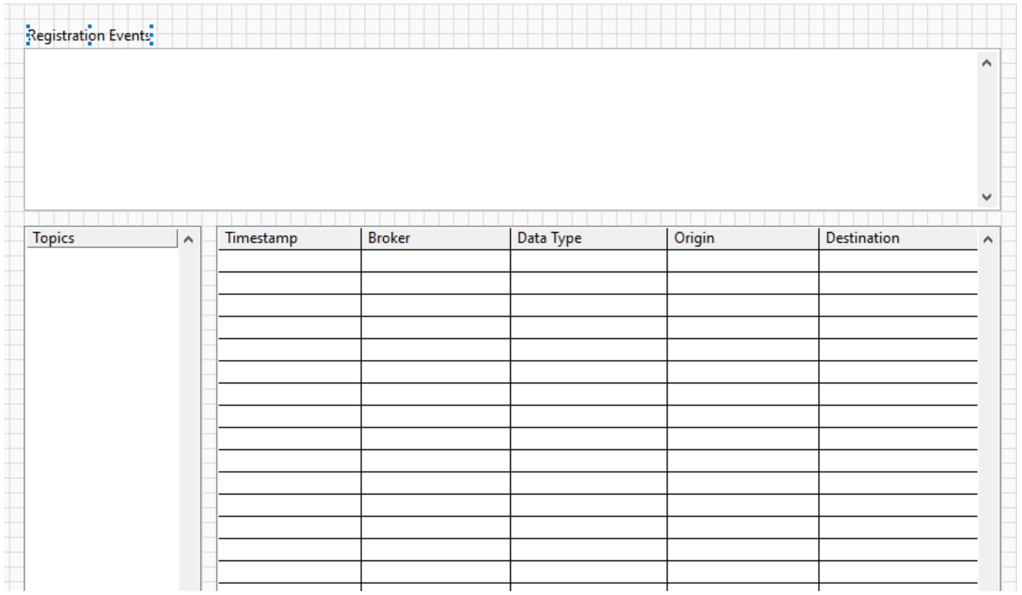
- Both must agree on the data type.

Implementation – Considerations

- Subscriber doesn’t know about the publisher
  - Although they must agree on the data type
- Publisher and Subscriber don’t know about the broker
  - They only know that it is their caller.
- Publisher and Subscriber depends only on the abstraction
- Broker most of the times know about the callees.
  - It needs to launch the nested actors
  - Runtime injection can be used to change this.
- Topics can be used as private data.
  - Decoupling topics from nested actors.

Implementation – Other Features

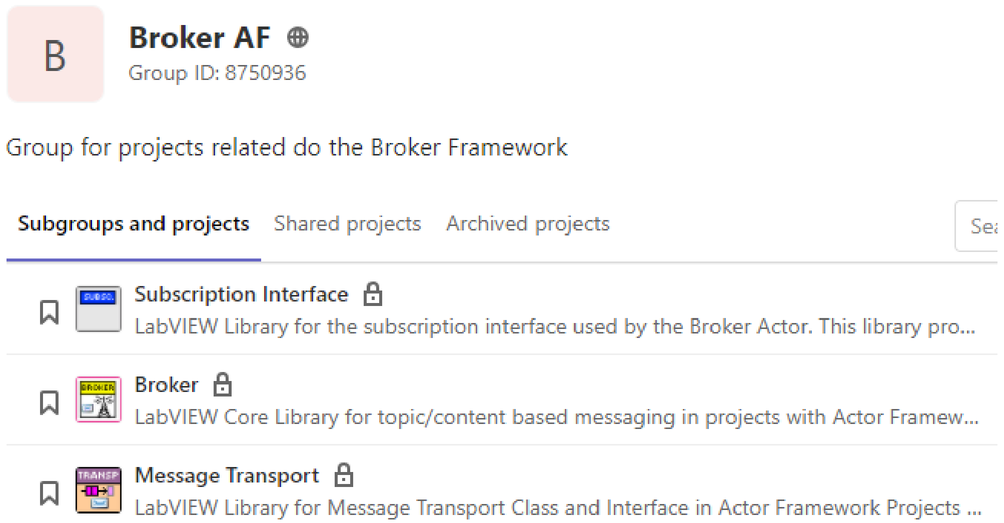
- Timestamp
  - All messages are sent with timestamp that can be used for anything.
- Debug Console
  - Use to track message path (working feature).


Registration Events

Timestamp	Broker	Data Type	Origin	Destination

Implementation – Code Repository

- Gitlab Code Repository


B

Broker AF

Group ID: 8750936

Group for projects related do the Broker Framework

Subscription Interface
LabVIEW Library for the subscription interface used by the Broker Actor. This library pro...

Broker
LabVIEW Core Library for topic/content based messaging in projects with Actor Framew...

Message Transport
LabVIEW Library for Message Transport Class and Interface in Actor Framework Projects ...

https://gitlab.com/broker_af

Agenda

- What is and what is not this presentation…
- Broker "Framework" Idea
- Message Patterns and the Broker Concept
- Abstract Message/Interfaces
- Implementation
- **Future Work**

Future Work

- Scripting
  - Some scripting for the actors and make life easier
  - Creating topics, publishing, subscribing methods
- Documentation
  - Include more examples
  - Extend the current VI Documentation
- Debugging
  - Improve constantly the debugging tool

Resources

- Actor Framework Learning
  - Actor Framework from Basic to PPL Plugins
  - Introduction to LabVIEW Interfaces
  - Actor Framework Video Tutorials
  - LabVIEW Wiki Actor Framework
- Actor Framework Best Practices
  - What a Software Architect Needs to Know
  - Best Practice Guidance Sibling Actor Messaging - Forums
- Actor Framework Messaging Content
  - Fun with Actor Framework Pub/Sub
  - Actor Programming without an Actor Framework
  - Event Source Actor Package
- Messaging
  - Message Exchange Patterns
  - Publish Subscribe Pattern

GLA Summit 2020

Questions?

Felipe Pinheiro Silva

felipefoz@gmail.com

http://felipekb.com