jettl

An Interface-Composition based LabVIEW Asynchronous Actor Framework

Nathan Davis

June 8, 2025

1 Introduction

Strictly interface composition based asynchronous actor oriented design pattern for LabVIEW Applications. jettl also has the newer banners for vis. Easy adoption for the new age of LV developers. State pattern with decorators. It is interface composition, so stick with the same rule set for naming methods

2 Philosophy

2.1 Access Scope

Only public and private. Interfaces, classes, and methods have text in the icon/banner that are black for public and red for private.

3 Class Hierarchy

Fig 1 shows the class hierarchy of jettl. In particular, there exist design patterns such as the Strategy Pattern, State Pattern, and Decorator Pattern.

4 LabVIEW Interfaces

The default implementation idea works so only as there is only one method implementation across all interfaces.

For example,
class implements interface 1 and interface 2
interface 1: method
interface 2: method
This cannot exist unless the class overrides the method. Otherwise, at runtime, LabVIEW does not know to execute interface 1: method or interface 2: method.

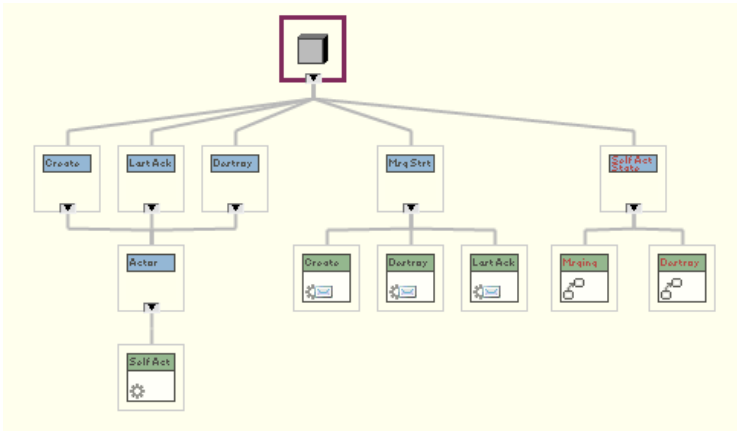

Figure 1: jettl Class Hierarchy.

4.1 Descendants Must Override

Interface methods: check the 'descendants must override' checkbox and see which errors pop up. Attempt to fix them, and if you can't, then justify the reasoning.

For example, in the Self Actor State.lvclass, the DD methods are not required to be overridden. This is because the Self Actor State.lvclass has default functionality in the interface methods. Maybe this is bad practice, but it is a design choice showing only the methods that are required to be overridden with different functionality.

Another example, the DD methods in the Dev Actor.lvclass are not required to be overridden. This is because the methods here are all decorator methods that are not required to be overridden. Using the Actor.lvclass methods are sufficient for the Dev Actor.lvclass. These will be found in the palette.

5 Future Scope

Creating an actor is NOT connected to the actor that creates it. Rather, this actor exists on its own in the "liquid" message transport. The overall "application" has access to the actor's reference (unique message address)

5.1 Pub-Sub Messaging

As shown in Fig 2, it is as though the actors are just objects that "float" in this "liquid" messaging transport. So that way you can perform this pub-sub messaging. The objects are linked together by "who launches who".

Same as with the pub-sub pattern, everything is a publisher-subscriber. It's just that most relationships are just one way. Same with Git, the philosophy is that all branches are created equal

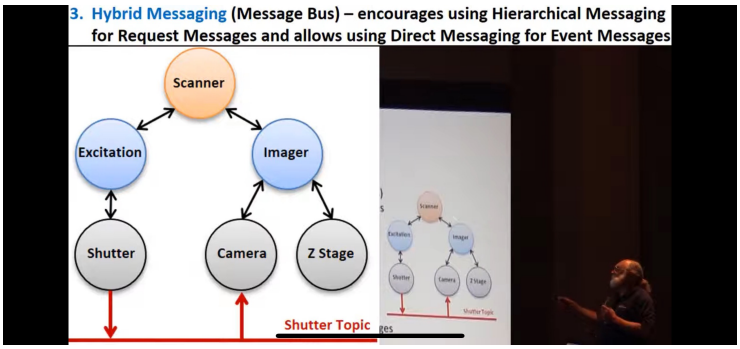

Figure 2: LabVIEW Actor Framework Message Transport.

6 Debugging

6.1 Displaying Method Execution

What a log for which methods are executed, instead of the dialog popups

Two project conditional blocks:

1. Occurs in all methods
2. Occurs only in message specific methods

These conditionals occur in Self Actor.lvclass. If either conditional is true, debug panel that displays the names of actors (columns), along with timestamps (rows) of when methods (data) are executed. Think discrete time water fall display.

7 Git

This might be of interest for submodules in git for LabVIEW. Find here: Git Submodules: An Alternative Approach to Code Reuse - Greg Payne - GDevCon2.

In the repos, use the tag to have different "stable" versions of the repo such as v0.1.1 or v3.8.3 This allows others to easily look at the different versions of the repo without much thought. This could also help with submodules that are referenced in other repos. Check this video near the end for reference: Git and GitHub Tutorial for Beginners

8 Naming Conventions

Noting that Fig 3 shows the naming convention for LabVIEW classes and methods.

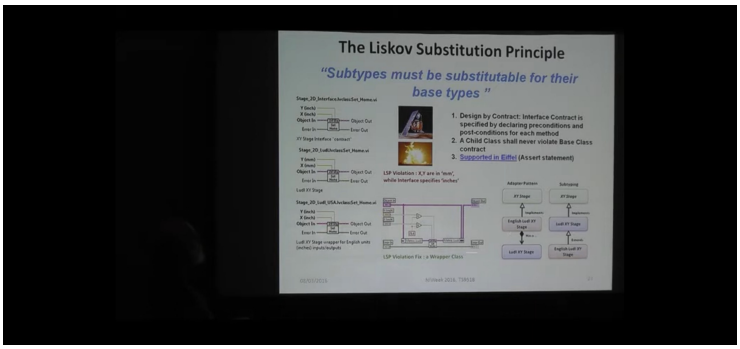

Figure 3: This image shows the naming convention for LabVIEW classes and methods.

- Library-Name.lvlib
- Interface-Name.lvclass
- Class-Name.lvclass (control is capital by default!)
- method-Name.vi
- control-Name.ctl (this control, which is not tied to a class is lowercase!)

Note: Avoid underscores and spaces.

9 Design Patterns

9.1 State Pattern

Context:
    Method.vi (just a wrapper for Method.vi "State").[1]
State.lvclass (interface):
    Concrete State.lvclass
    Method.vi "State"

10 External Packages

IG OOPanel

[1] Best Practice: use this Method.vi in other methods, rather than the Method.vi "State" itself

10.1 Panels

Actor helpers, not helper actors. Helper Loop → Async Actor Helper. These are helper loops that are created async which help an actor with its task. This is to declutter the actor by offloading certain tasks such as panel display or subprocesses that do not require an additional actor to be created.

10.2 UI Events

Indicators: Instead of generating an event, bundle in the reference of the indicator and have a property node (write)in the subVI.

The same can be said for the control, which has a right click menu, that bundles in the reference for the control and creates a subVI that writes the value to the front panel. Or creates a signal, something that automatically puts the control in the event loop and bundles in the reference for the control to the subVI. Create control which does the same as the indicators but added the reference of the control to the private data.

In prelaunch, have a subVI that internally has the creation of the user events. Script to create the subVI first.

Interface with DD methods: No need for template to replace the placeholder methods with new ones, just have DD methods as "placeholders" which change their implementation depending on the class that's on the wire

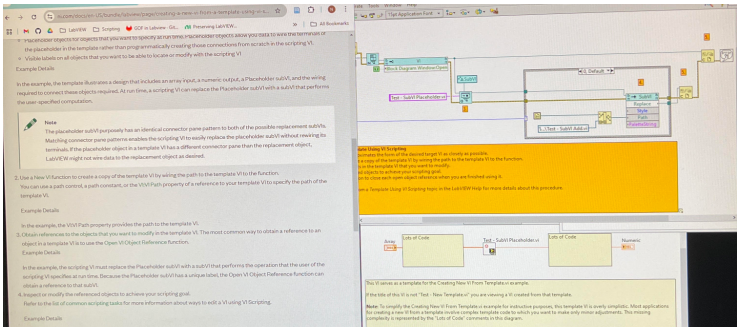

Figure 4: This image shows the UI events in LabVIEW.

Split the events as controls and indicators as shown in Fig 4.

10.3 Right Click creates library with interface

Method in actor:
Creates library containing interface and method along with message class with Send and Do methods

11 Messaging

Actor messages error on generation of message. Error when creating a message and wiring in the interface object as an input, the message scripting doesn't know how to differentiate the class input and the parameter input

12 Scripting

Go through and replace all the Opens and Traverse with the hidden gem.

User groups for the scripting (quick drop, right click, etc.).

13 TODO

Check execution for all methods.

Have the interface checkbox for DD methods not required for override doesn't necessarily break subtype rules!

In the DD methods, because they are not required to be overridden, have functionality within them that calls the Self Actor method (some kind of checking mechanism?).

Actor interface methods (all implemented with default functionality) to have the *new* (Actor Interface) Read Actor DD method that reads from the Dev Actor the Self Actor class and performs that interface function, then bundles back in with the Setup method.

Do this for all for the default behavior.

Note this is breaking the contract for interface DD methods not needing to be overridden.

Instead of the Setup method, create a new Actor Write method which ONLY Writes to the Actor. This can also be put inside the Setup method as a first step, the setup code follows after

State Enter Core and State Exit Core are NOT check marked. That way the developer does not need to override, just to have no functionality anyway. Read State and Write State do because they'll have functionality.

14 Miscellaneous

14.1 MEF (Managed Extensibility Framework)

LabVIEW Interfaces for Satellite Calibration - SLM and McBee: 44:50

15 Errors

In frameworks, errors are fundamental to the program's operation. They are not just incidental issues but rather integral to the design and flow of the application.

jettl has an error object in the private data of the jettl object. At the end of the method, the error object is unbundled and checked for errors before teardown.

Error philosophy: Errors occur ONLY from unexpected events. For example, the error case in Msg.vi: in the case structure, a custom error saying that a message could not be executed occurs when 'message name' occurs for 'actor name'. This is unexpected behavior since this SHOULD be known at edit time. This is a legit error.

'That was handled, or I wouldn't have been called' - SLM (The Errors of our Ways — Stephen Loftus-Mercer GDevCon N.A. 2021: 52:08)

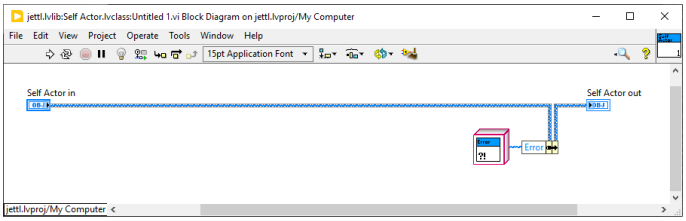

Figure 5: Method template. This has no error terminals, and the error cluster is handled internally. This is a different approach to error handling in LabVIEW methods.

Wouldn't this make the API more beautiful and easy to understand? Having just the object wire come out of the method, and ONLY the object wire coming out of the method? I suppose, adopting the OO paradigm. Instead of having the error cluster inside the objects class data. Instead, it's a dedicated "error cluster" shared for every single object in use. Encourages data flow since unbundling of errors will always occur. It's a step in the right direction having no "error input" for methods. Now it's time to get rid of the error out. It's almost like branching an objects wire, in a way. There should only be one thing coming out of a method.

16 Orderly Shutdown

Boolean flag for orderly shutdown i.e.'destroy nested actors before destroying self'

17 Example

17.1 Dev Actor

No decorators used since they're trivial and only used once.