AZInterface:
OOP Interfaces in LabVIEW
Solution and Implementations

Andrei Zagorodni

2019-04-02 Krakow

This presentation describes version 2.0.0.0 of AZInterface

www.azinterface.net

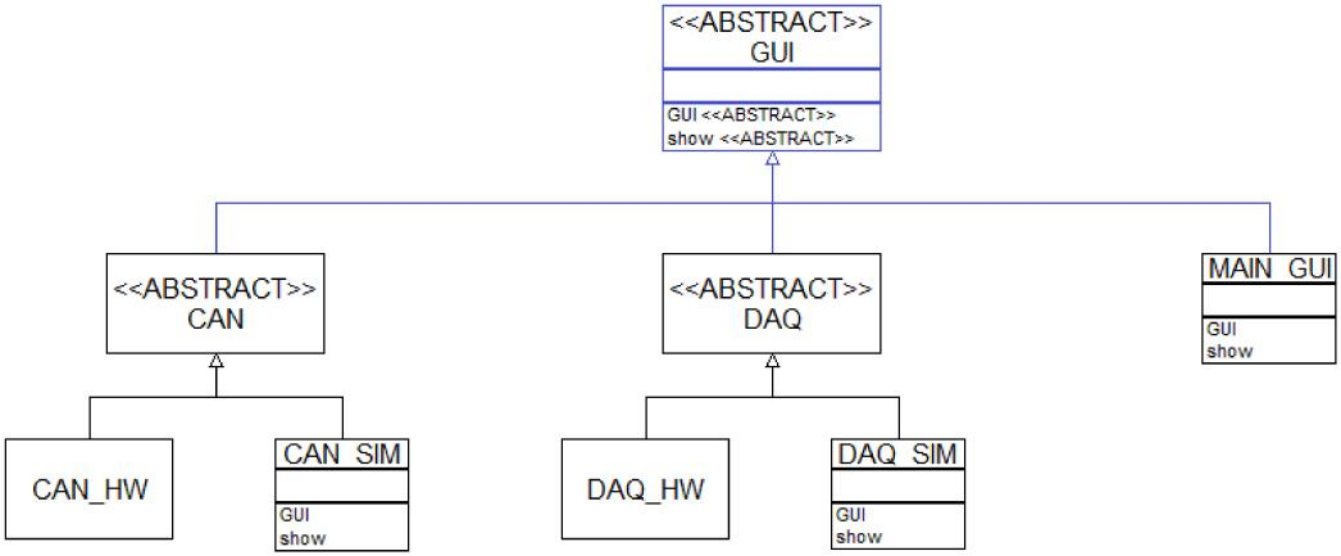
Content

- Interfaces in OOP
  - multiple inheritance
  - concept of interface
  - available solutions
  - benchmarking

- AZInterfaces
  - how to
  - what do we get
  - how does it work
  - how to use
  - pitfalls

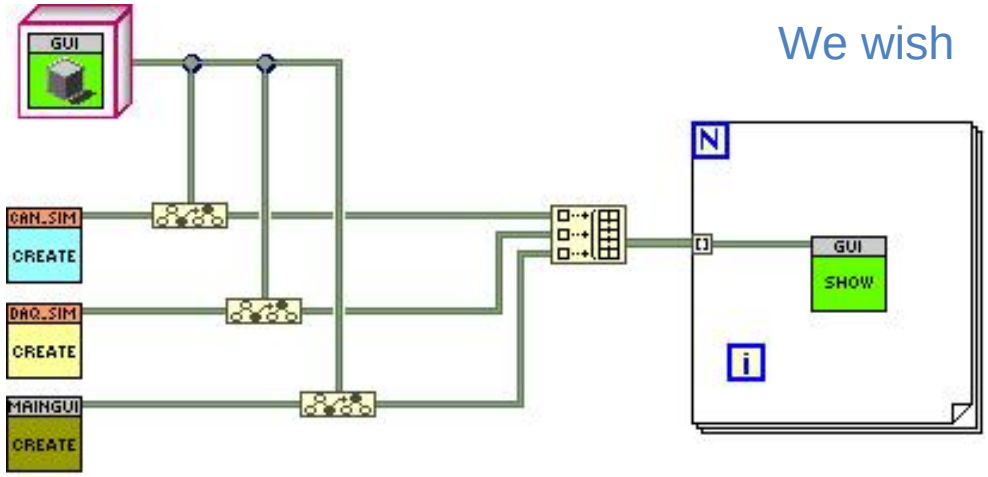
- What do we need more?

Common Abstraction Layer: Superclass



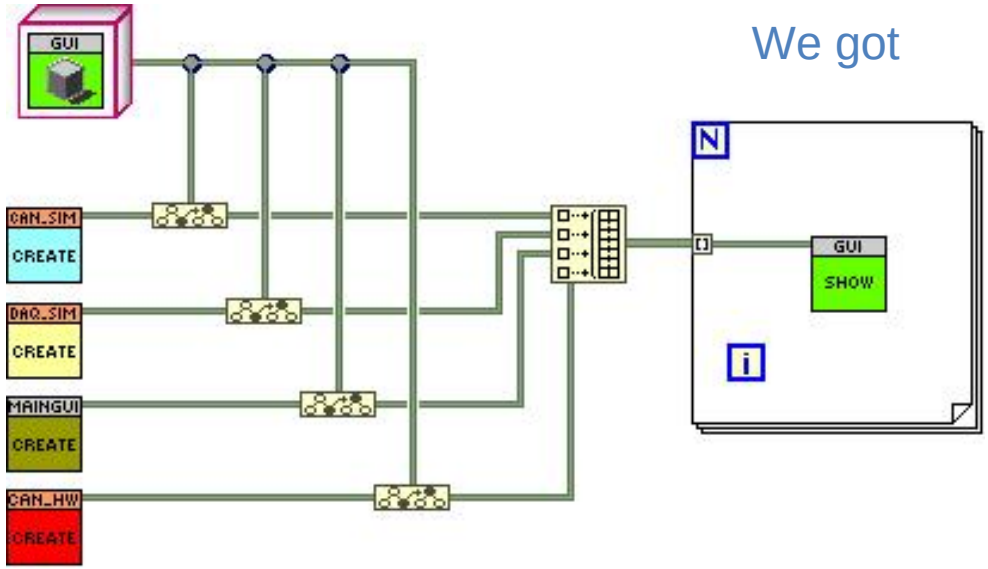
Not a real project, of course.

LabVIEW: Common Superclass



Does it work?

LabVIEW: Common Superclass

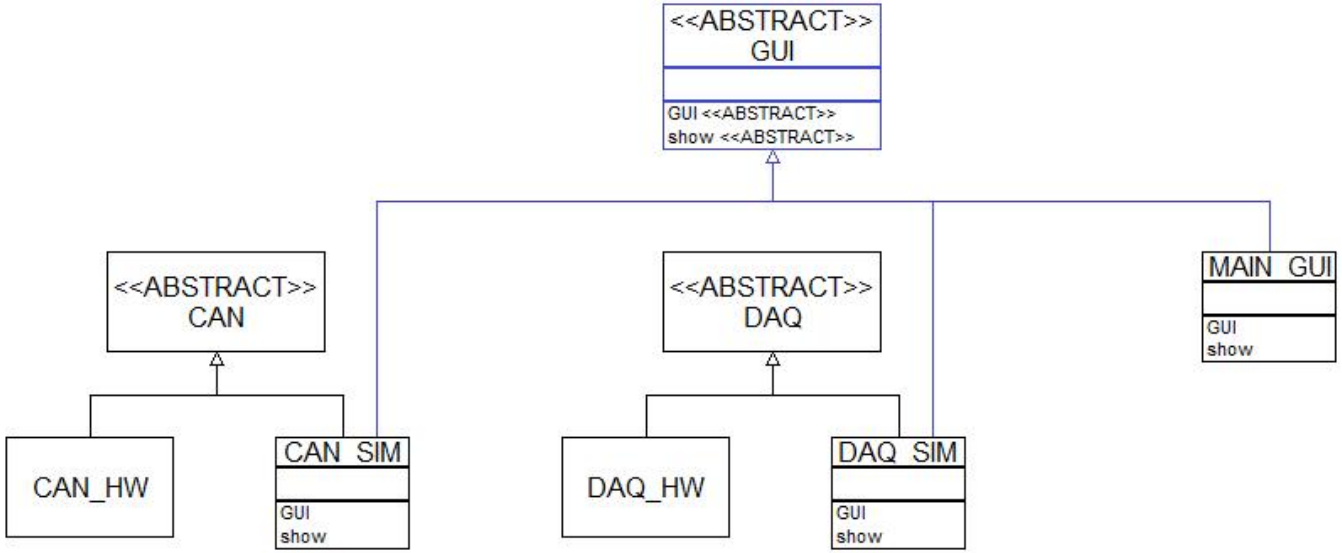


We got

Does it work?

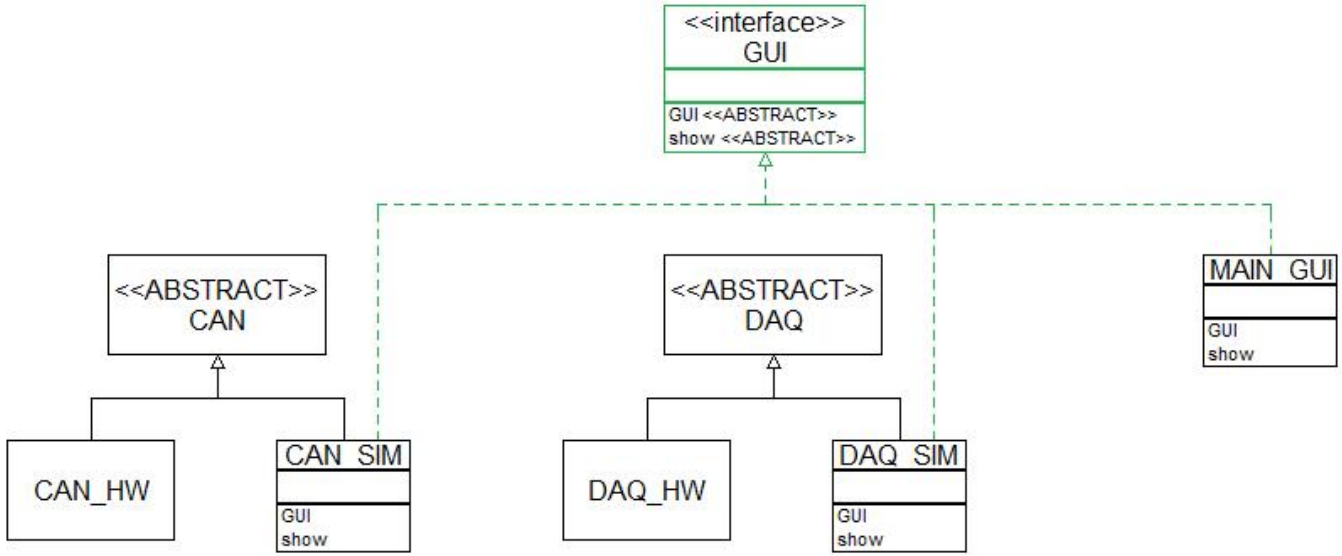
Ya, right…

Multiple Inheritance



Interfaces

Interface can be considered as a class without attributes, and with all methods being abstract.

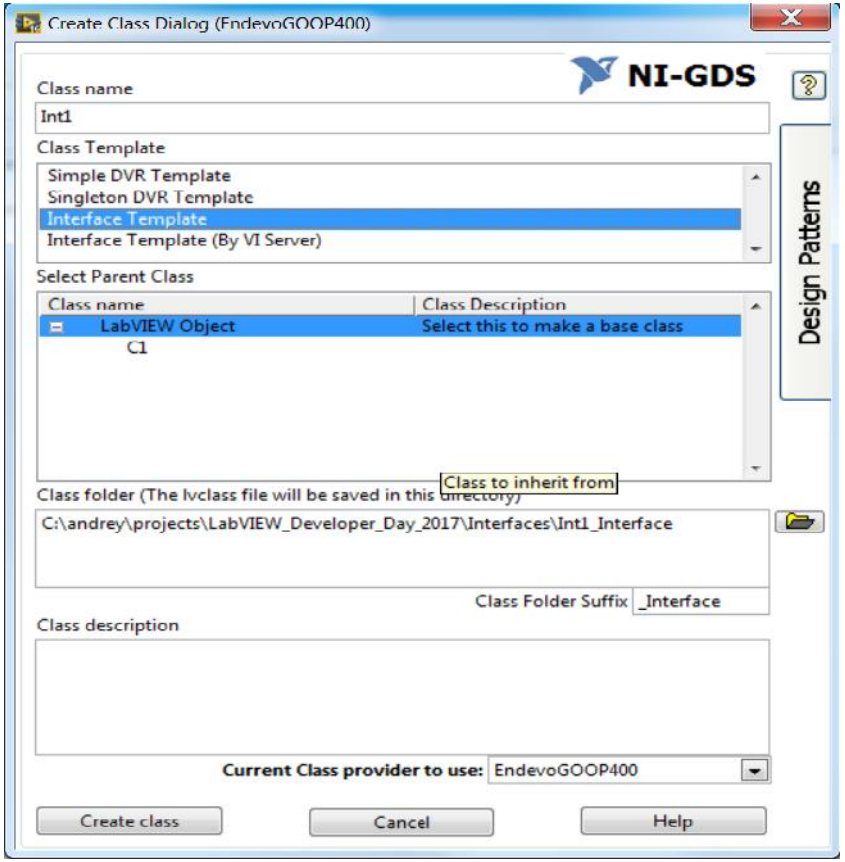


Interface

- Interface is a definition class
  - No code
  - No data structures
  - Only empty methods to override
  - Own data type

Available in LabVIEW?

Creating GOOP Interface

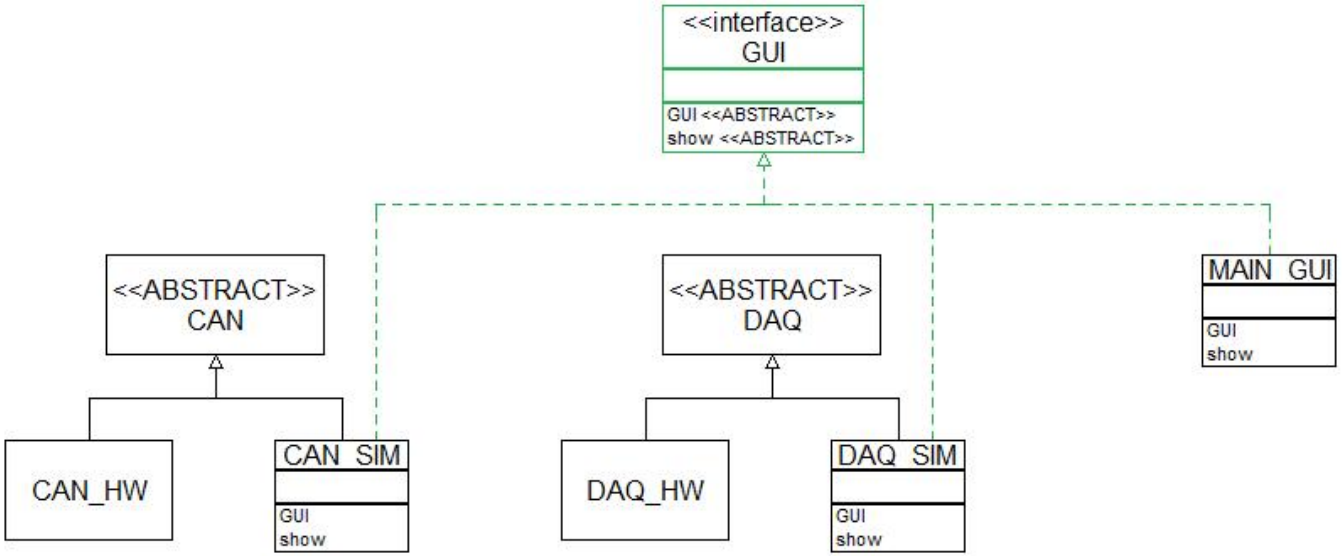


Two types of interfaces

- Why two?

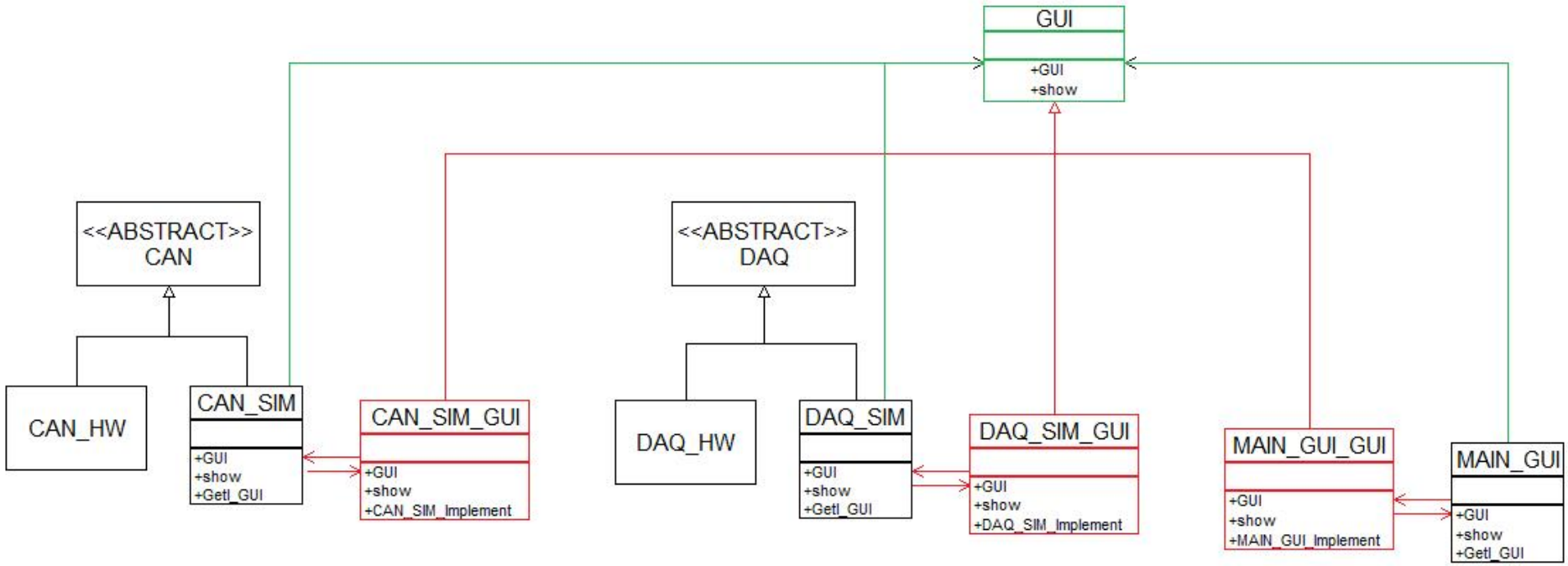
- Because no one of them is straightforward

GOOP Interface by Aggregation



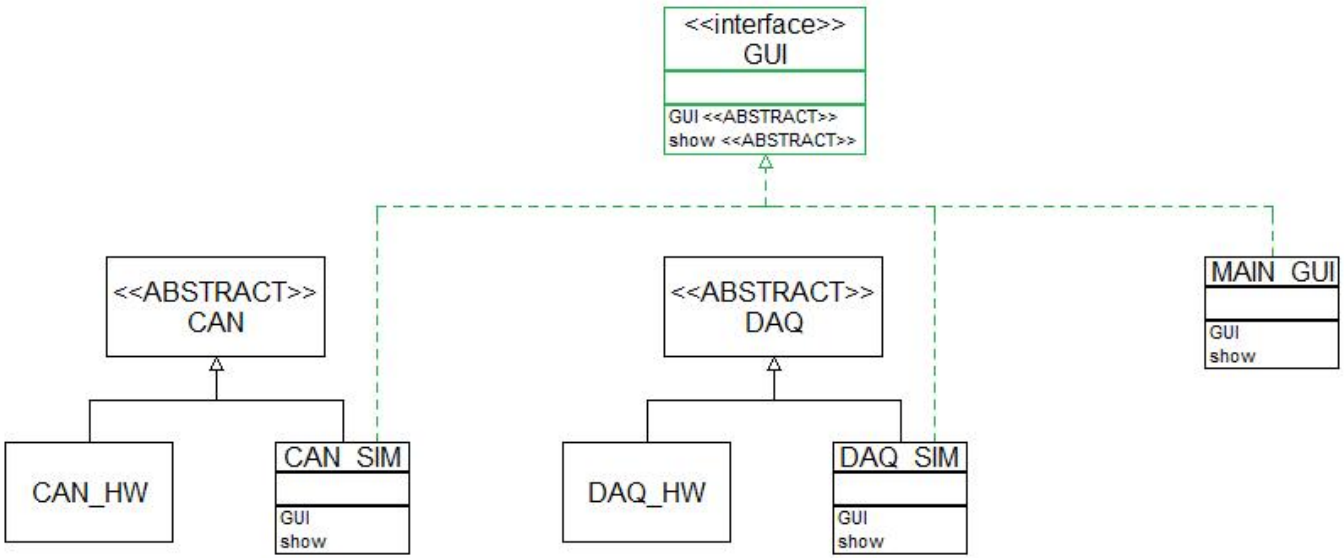
Our wish

GOOP Interface by Aggregation



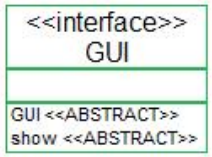
What do we get

GOOP Interface by VI server

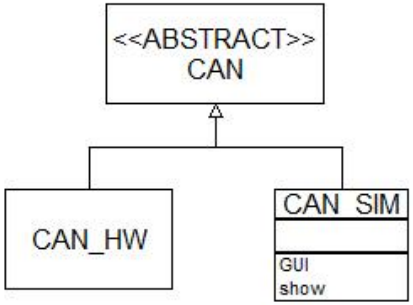


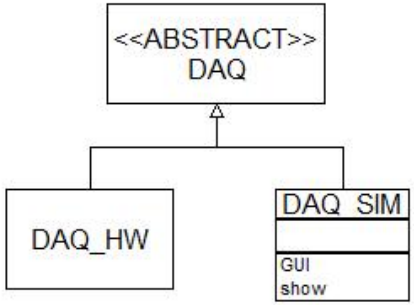
Our wish

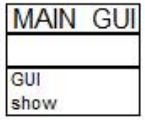
GOOP Interface by VI server



There is no interface data type

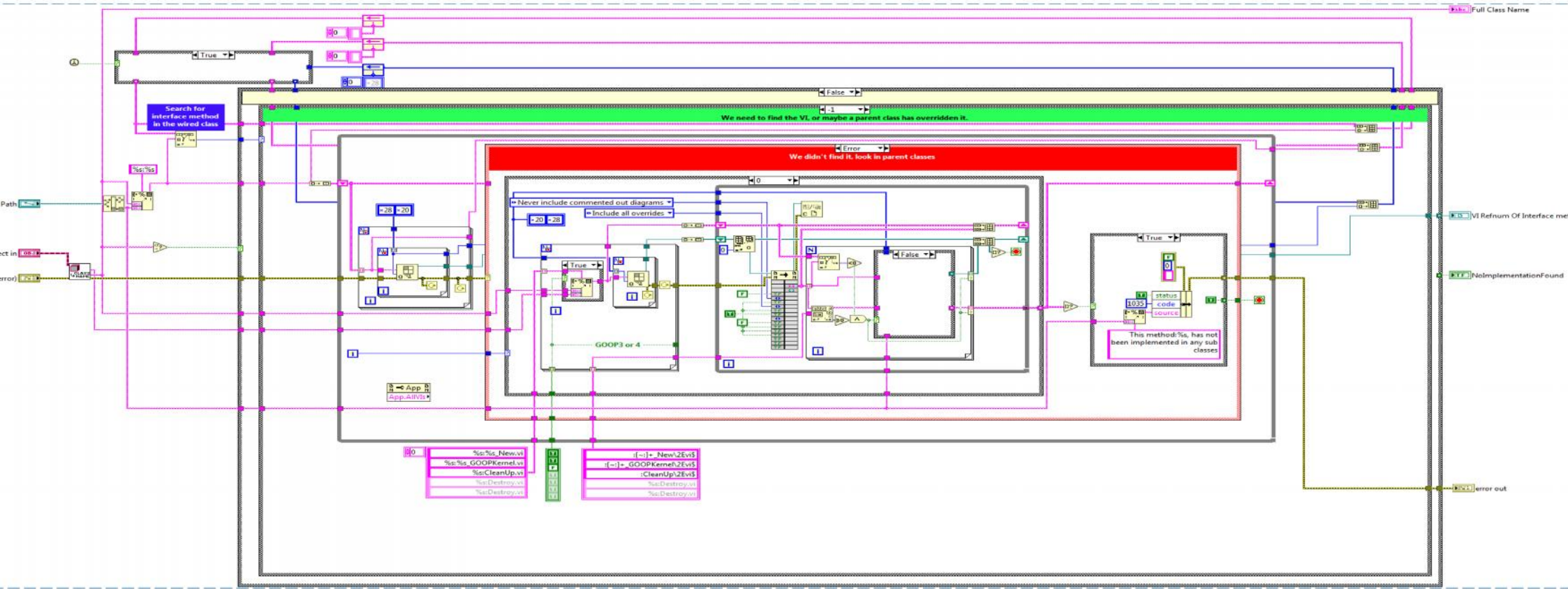




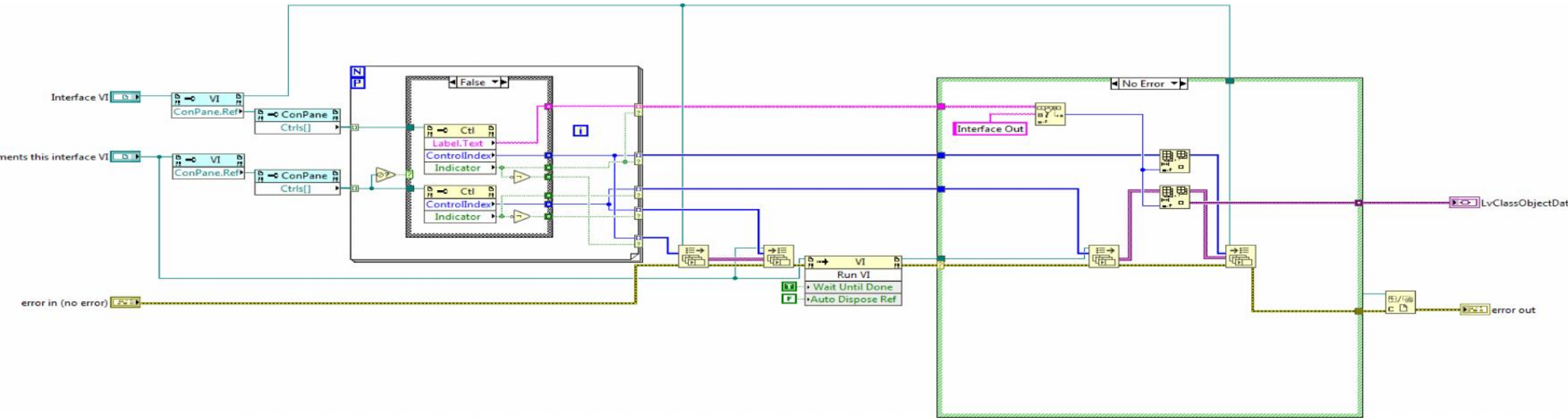


We get. It works! But…

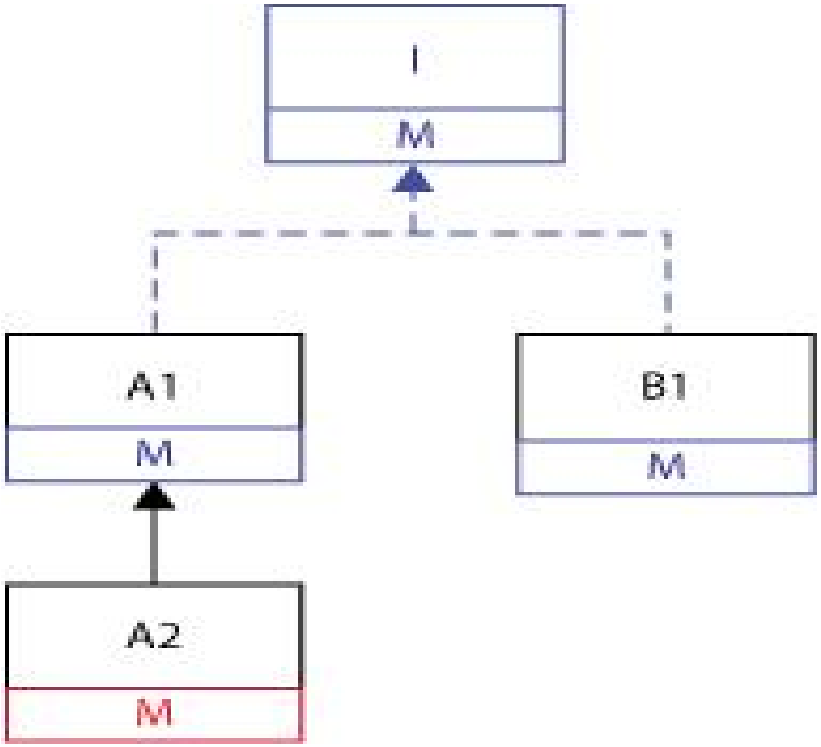
Interface by VI Server: Service Code



Interface by VI Server: Scripting



Benchmarking



Comparing Different Solutions

	GOOP by Aggregation	GOOP by VI Server	AZInterface v.2
Benchmarking: time spent	OK 100 %	Super-slow 30000 %	?
Interface data type	Yes	No	?
Dependency	Straight	No	?
Additional code	Too much	No	?
Readability	Bad	Good	?
Code in VI.lib	No	Yes	?

Common superclass (GOOP): Benchmarking 25-30 %

Disclaimer

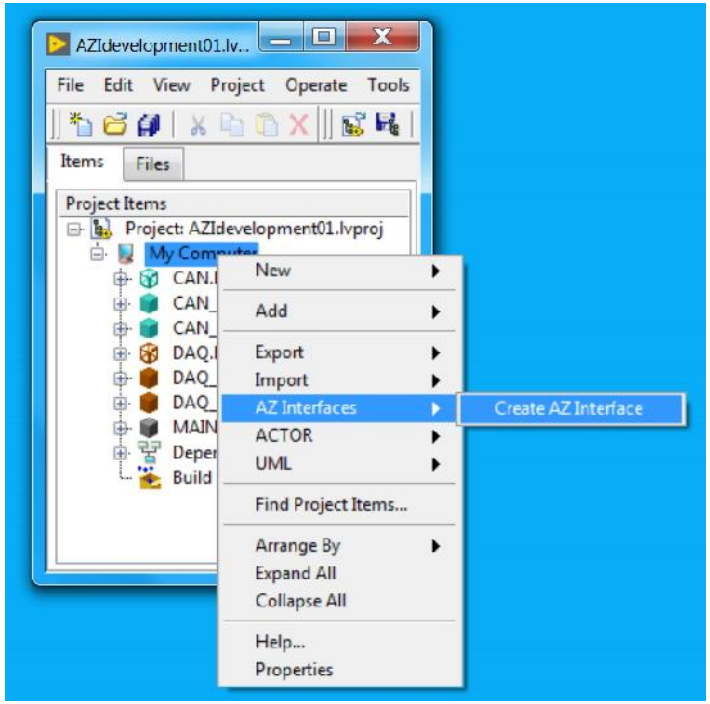
- What a LabVIEW programmer does when he is not working writing LabVIEW programs?
  - He writes hobby LabVIEW programs
    - AZInterface is a hobby project with all fun and consequences☺

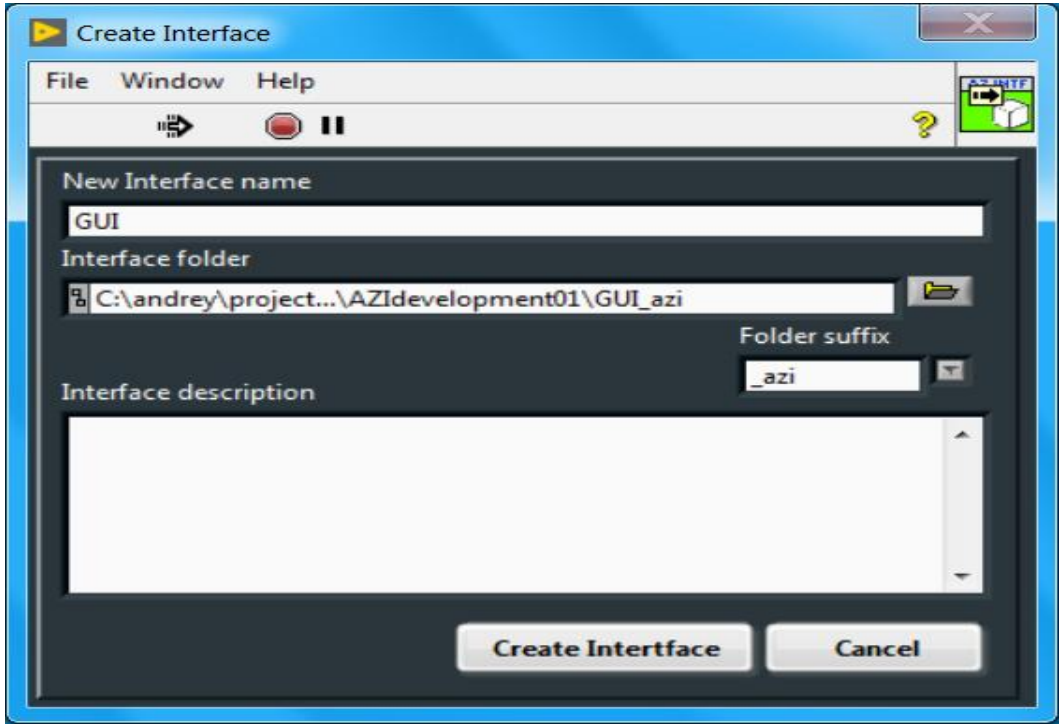
www.azinterface.net

AZ Interfaces

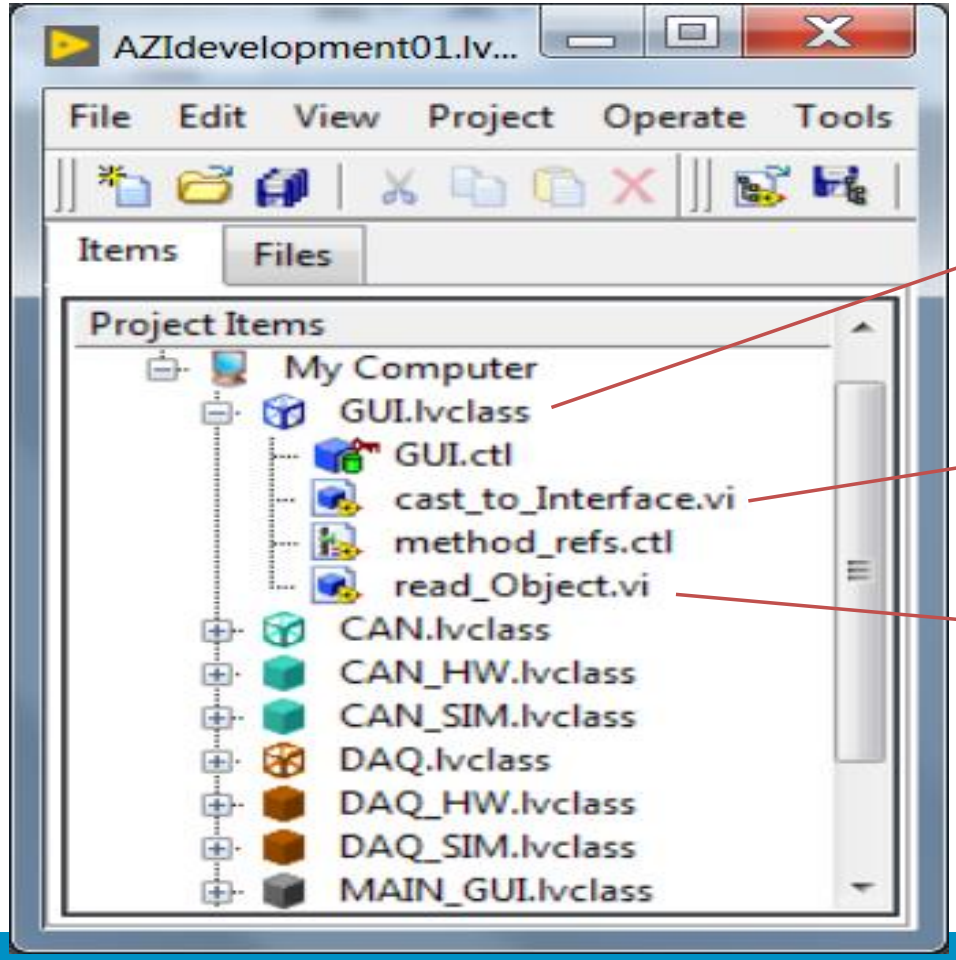
- LVOOP class is defined as an Interface.
  - Otherwise it is a class supplied with few specific members.
- Any class can implement such an Interface.
- Relationships between classes and Interfaces are defined by Community scope. Interface is added in list of Friends of the interface-implementing class.
  - Anyhow, I newer met a developer who used Community scope for anything but quick fixes ☺

Create Interface





New AZInterface

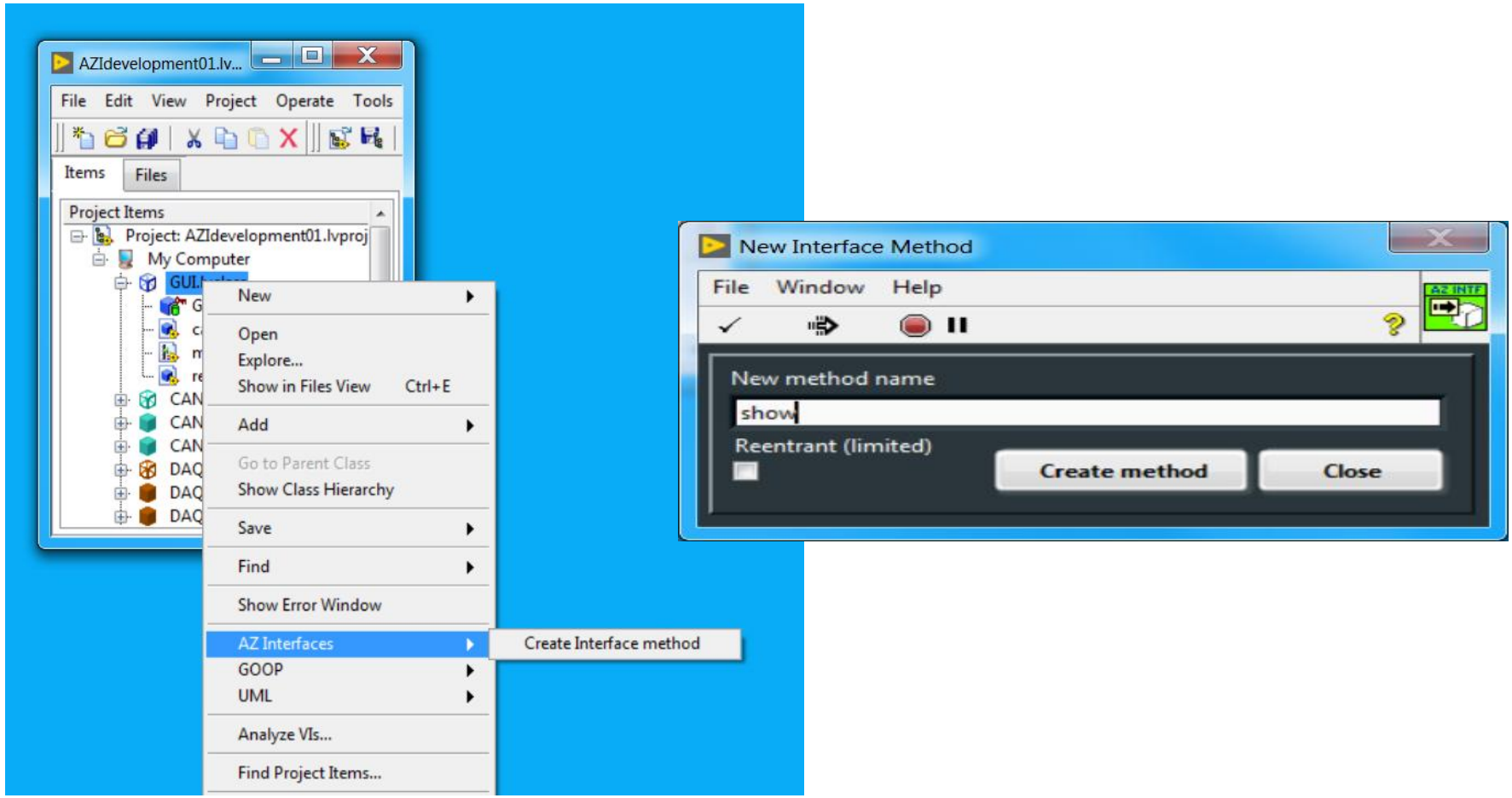


Interface

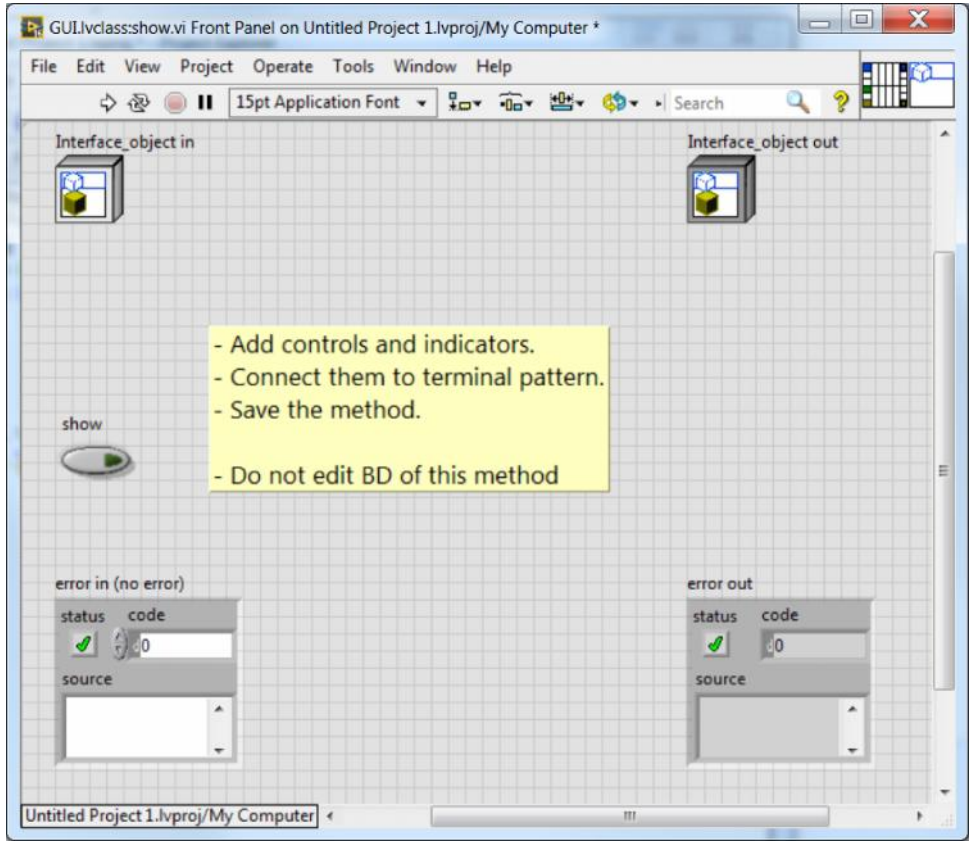
`cast_to_Interface` – method automatically included in BD of corresponding methods of interface-implementing classes

`read_Object` – double-purpose method: provides access to original class instance and destroys interface instance (only if the interface has reentrant methods).

New Interface Method: Create



New Interface Method: FP

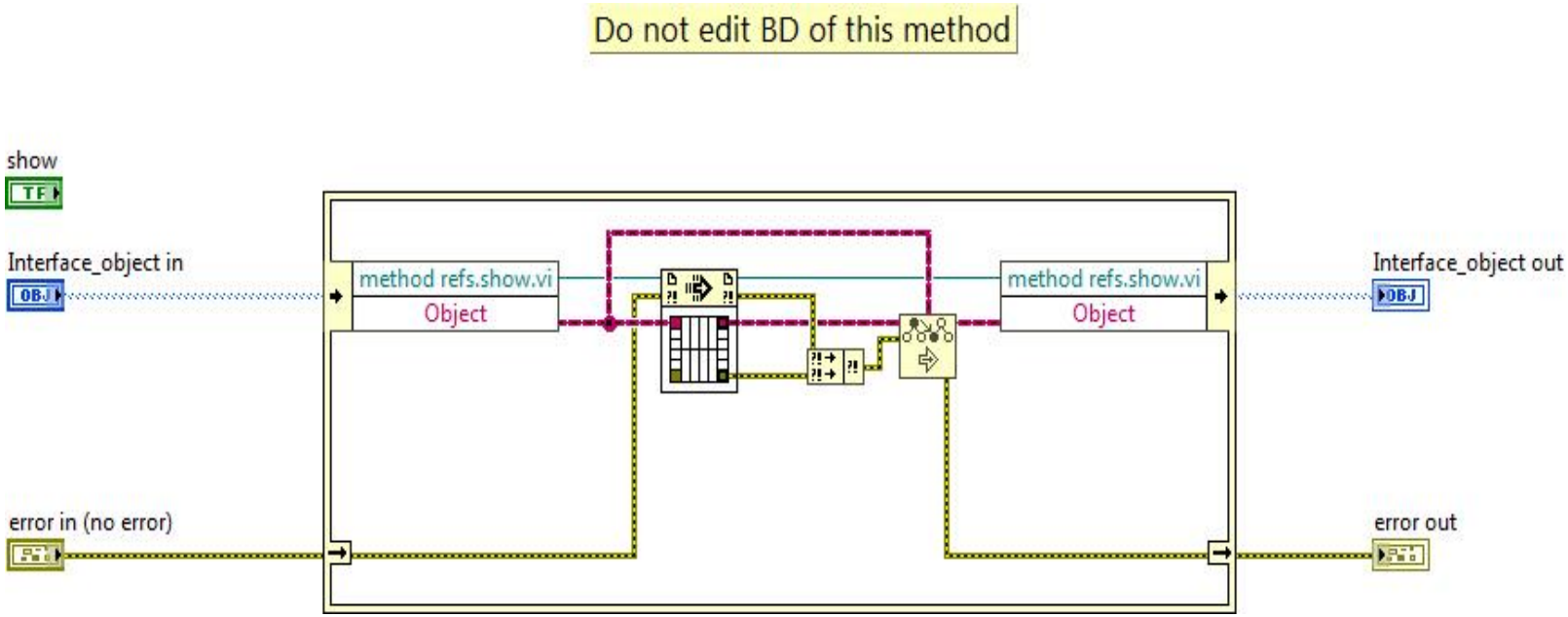


- Think about terminal pattern twice.
- It will be difficult to alter at later stages.

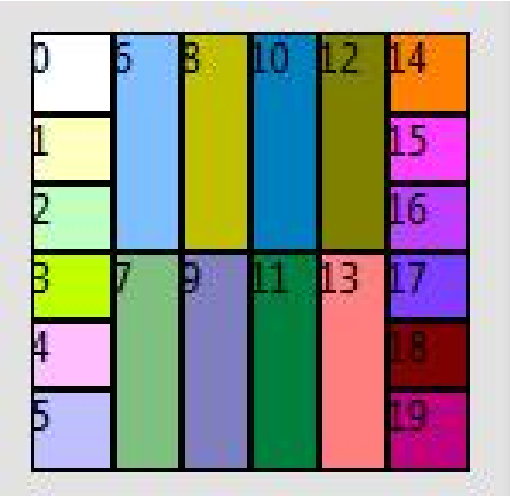
- As usual in OOP ☺

New Interface Method: BD

modified when first implemented in a class

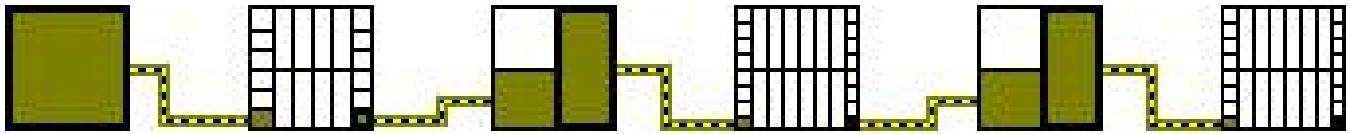


Terminal pattern limitation

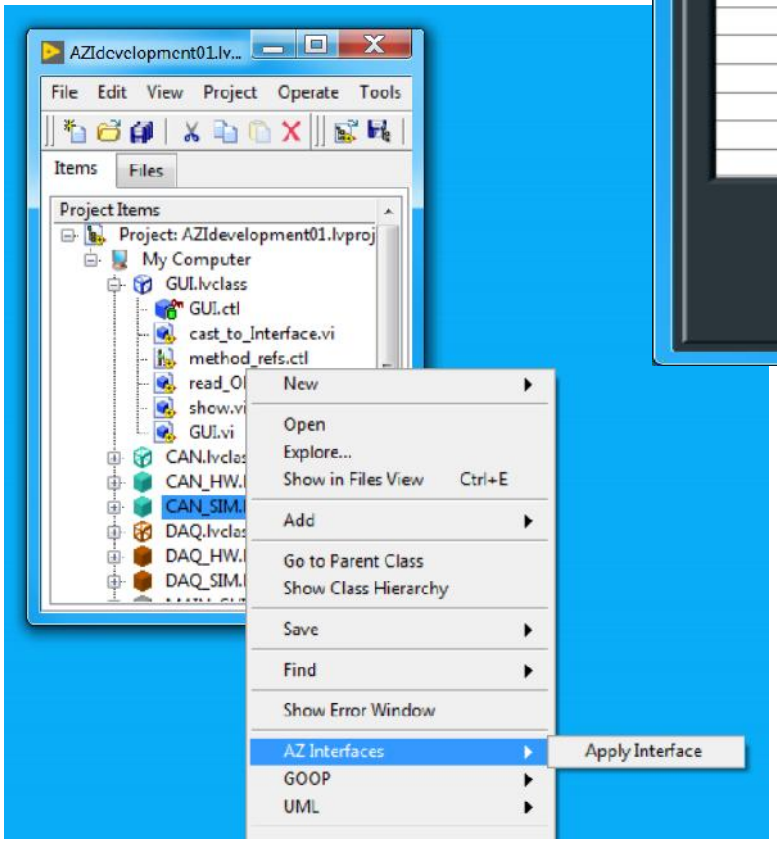


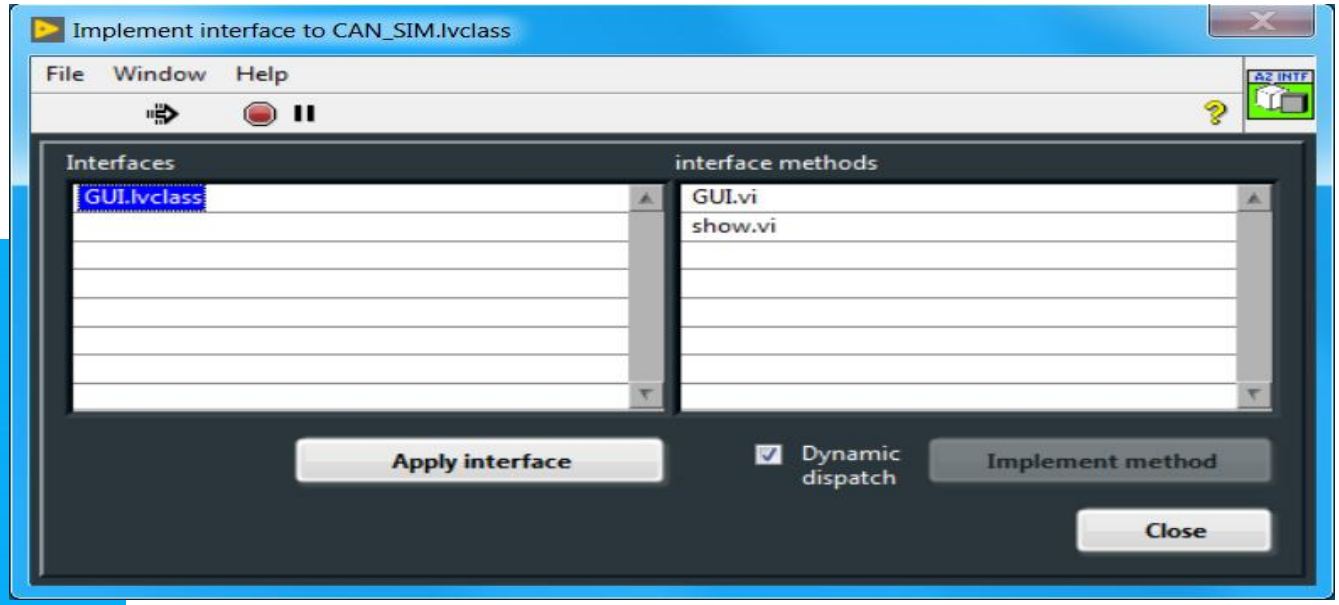
- Use only 20-connetor terminal patterns in all AZI methods!
- Anyhow, using the same pattern through the whole project is a good practice.

- And I hate such a mess.



Apply Interface

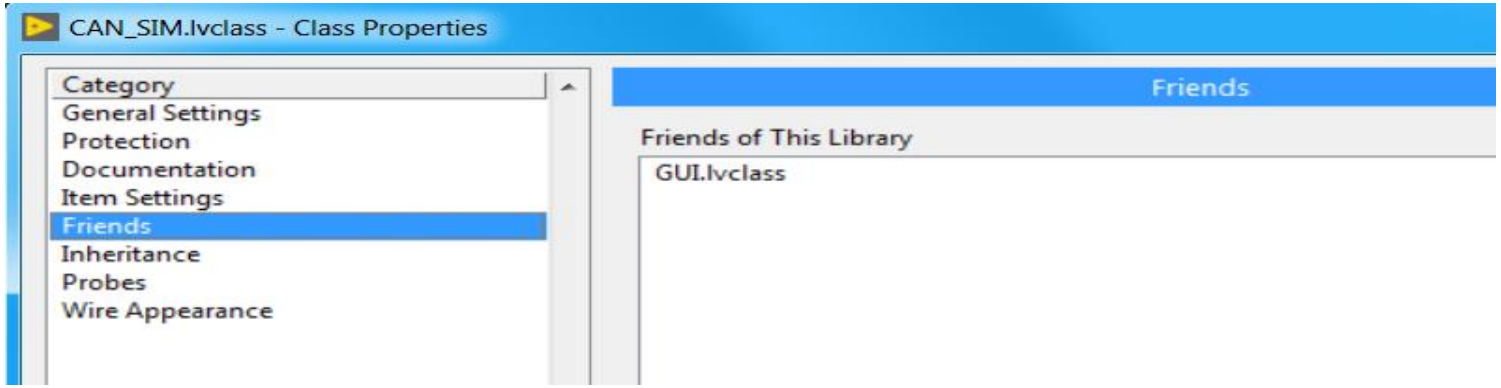




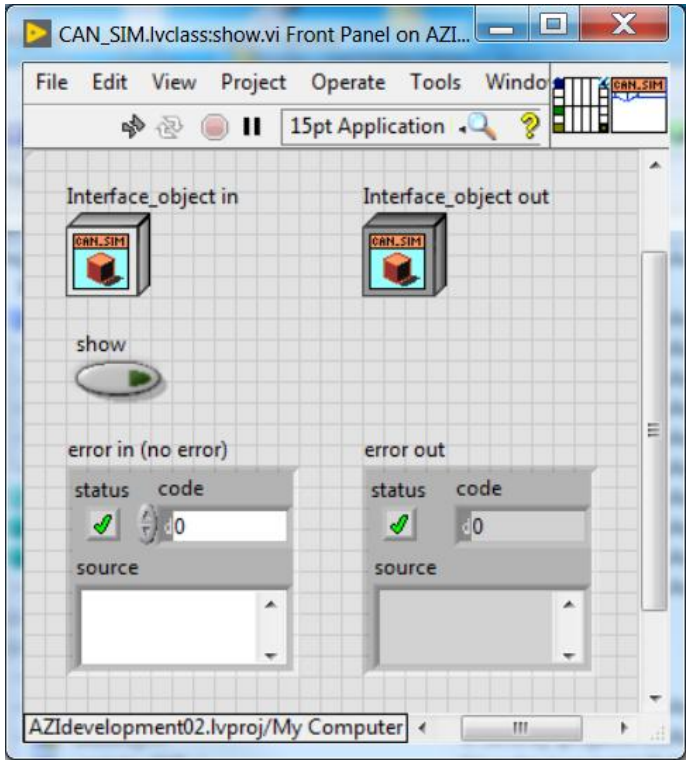
Attention!

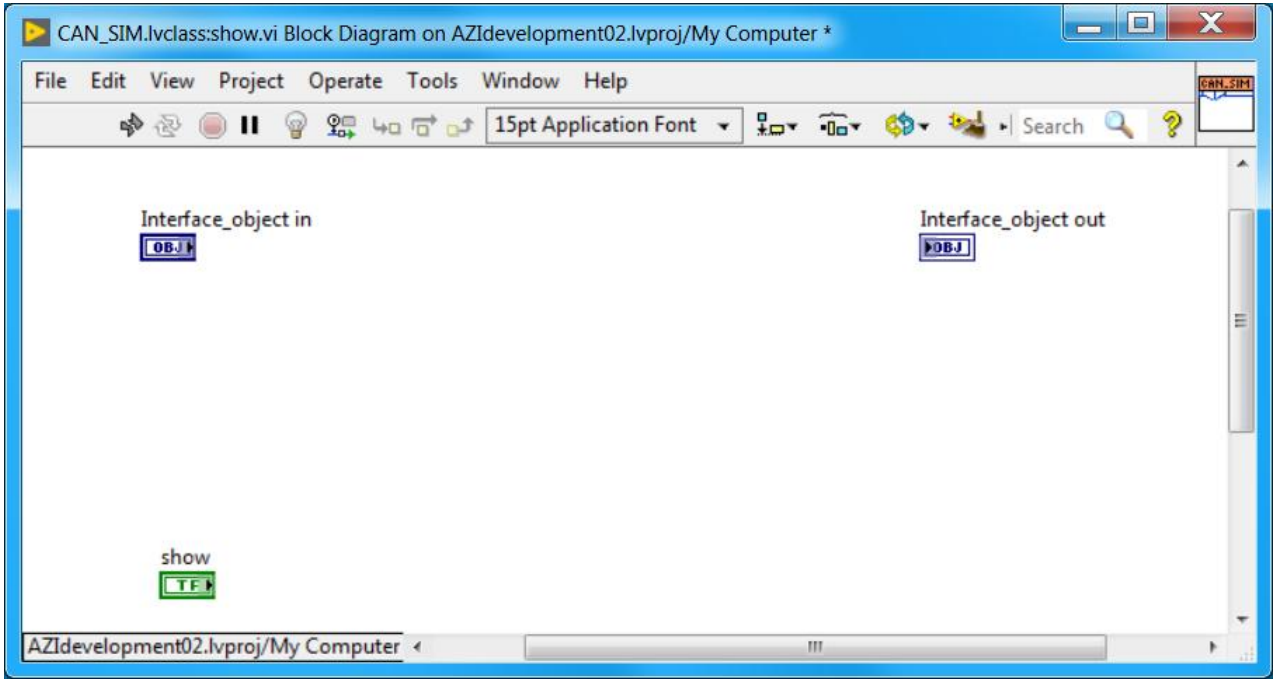
- If you implement interface method but the method already exists in the class, the methods **must** have complimentary terminal patterns.
- There is no build-in protection from errors here (v.2.0.0)!
- I believe this issue should be prioritized in next version.

Applied Interfaces defined as Friends



Method Implemented in Class

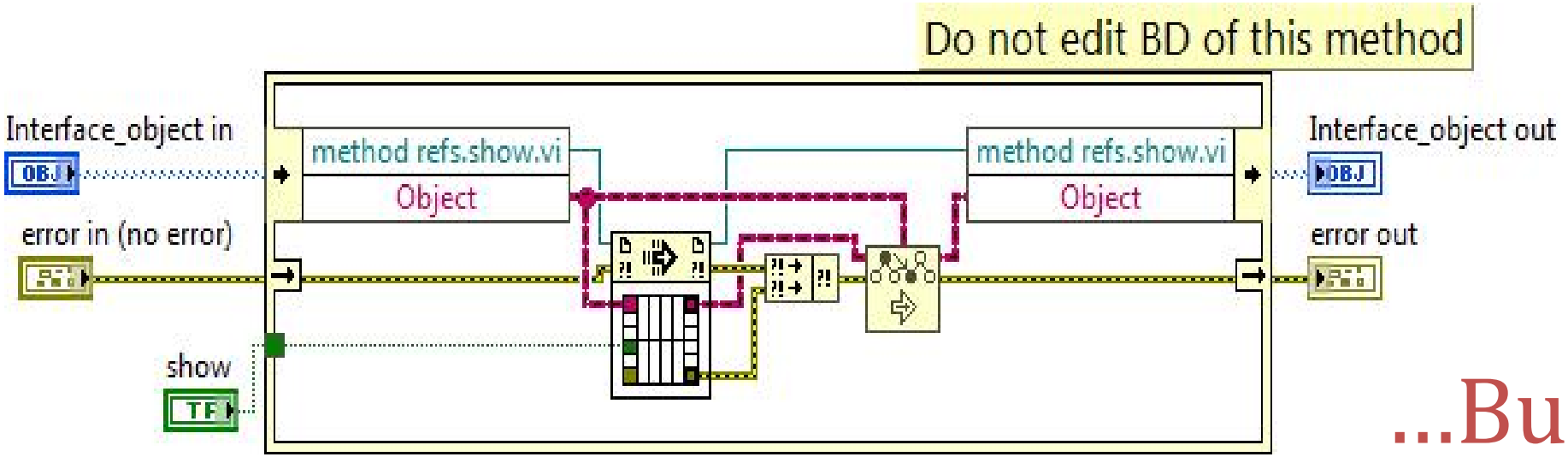




BD is empty

Code of the Interface Method

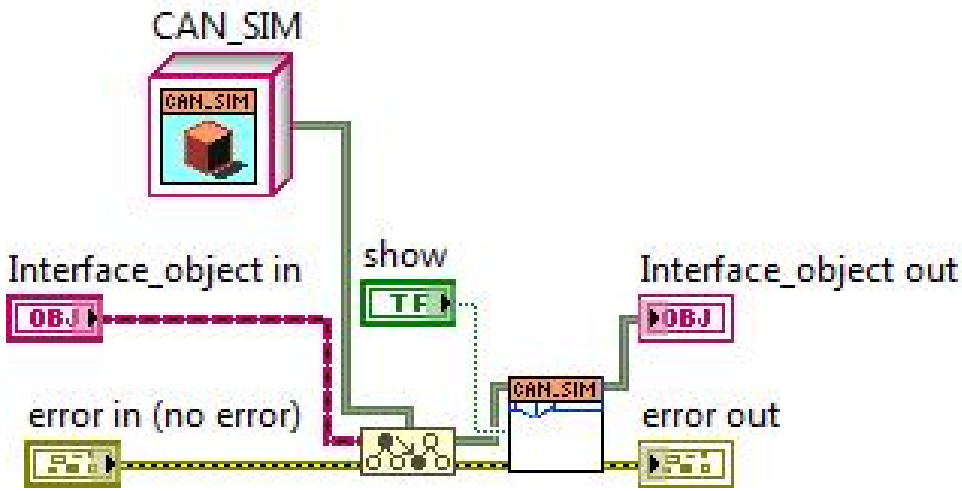
modified when first implemented in a class



...But?

- Created class method has object terminals of the class
- Interface method keeps object cast to LabVIEW Object type
- Call By Reference does not support Dynamic Dispatch

Wrapping utility method (“middleman”)

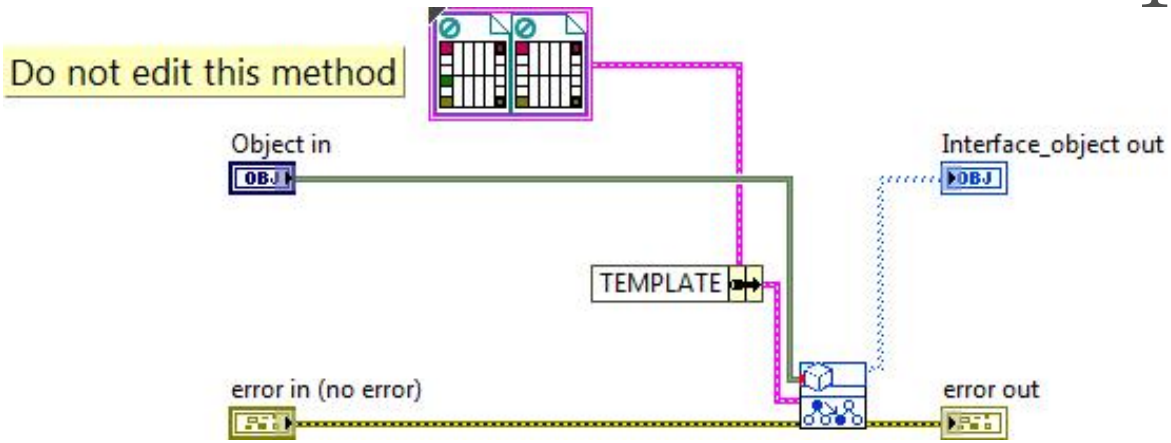


Transfers call from interface to class

“Provides” terminals of necessary data type

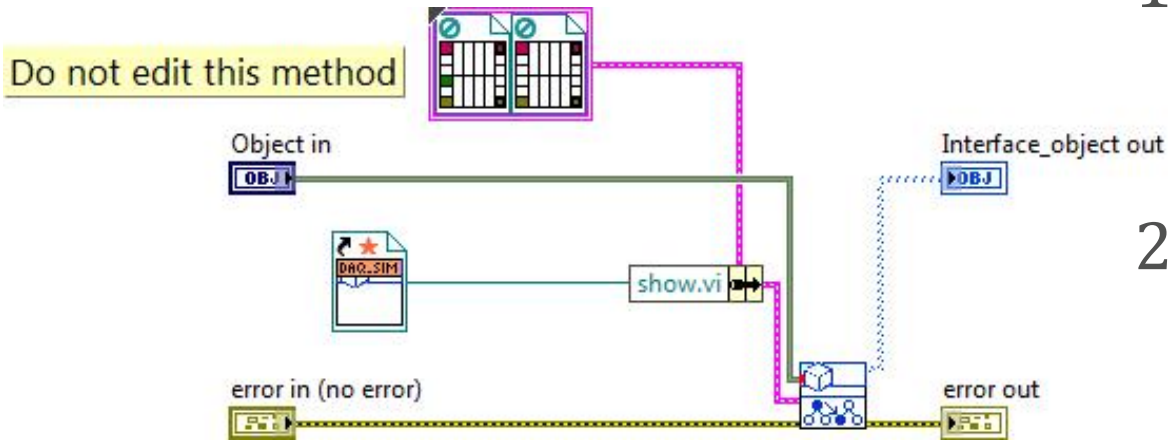
- Belongs to the class (CAN_SIM:util_GUI_cls_show.vi)
- Has object terminals of LabVIEW Object type
- Invoked by interface method (GUI:how.vi)
- Invokes actual class method (CAN_SIM:show.vi)

Method cast_to_<interface name>.vi



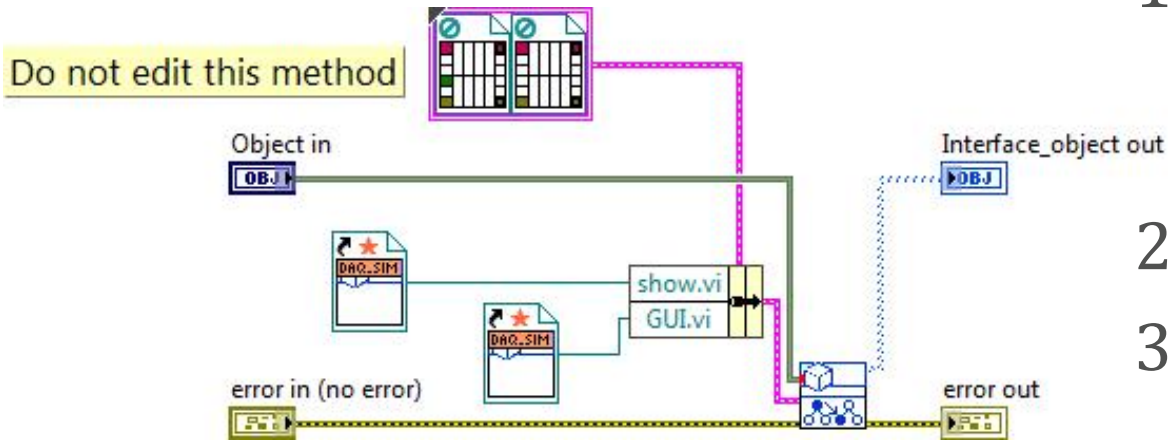
1. Interface is applied to class
   - The method is created but broken so far

Method cast_to_<interface name>.vi



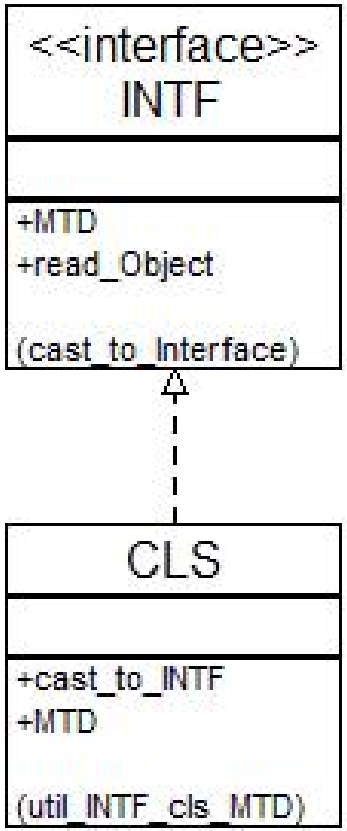
1. Interface is applied to class
   - The method is created but broken so far
2. First method is implemented

Method cast_to_<interface name>.vi



1. Interface is applied to class
   - The method is created but broken so far
2. First method is implemented
3. Second method is implemented
4. And so on…

Intermediate summary

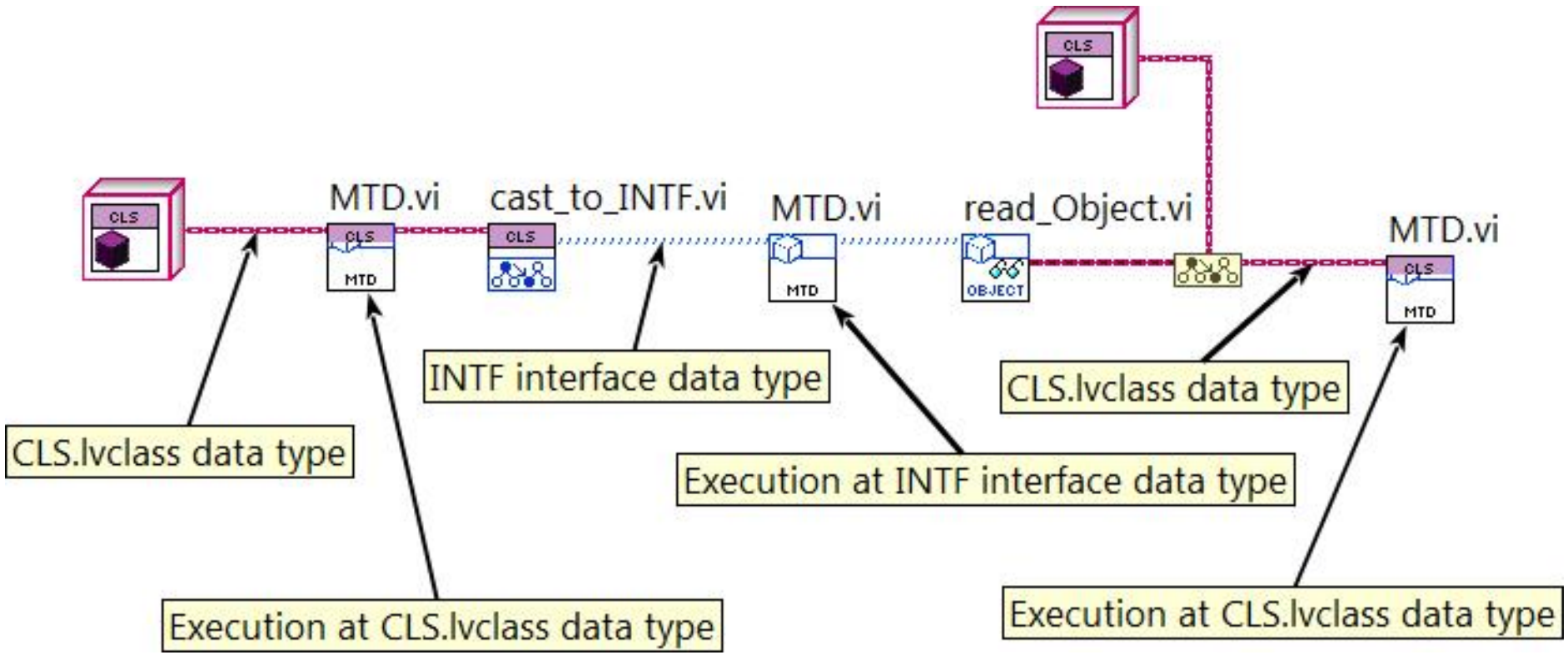


- Interface (INTF.lvclass)
  - Custom methods
  - Method read_Object
  - Utility cast_to_Interface
- Class (CLS.lvclass)
  - Custom methods
  - Method cast_to_INTF
  - Utilities util_INTF_cls_MTD

- Utilities must not be customized
  - They are automatically created
  - They are automatically modified
  - They are automatically included in the code

But?! How to use?

Method execution at interface abstraction level



Comparing Different Solutions

	GOOP by Aggregation	GOOP by VI Server	AZInterface 2.0.0
Benchmarking: time spent	OK 100 %	Super-slow 30000 %	Good 20 %
Interface data type	Yes	No	Yes
Dependency	Straight	No	Straight
Additional code	Too much	No	Some
Readability	Bad	Good	Good
Code in VI.lib	No	Yes	No

Common superclass (GOOP): Benchmarking 25-30 %

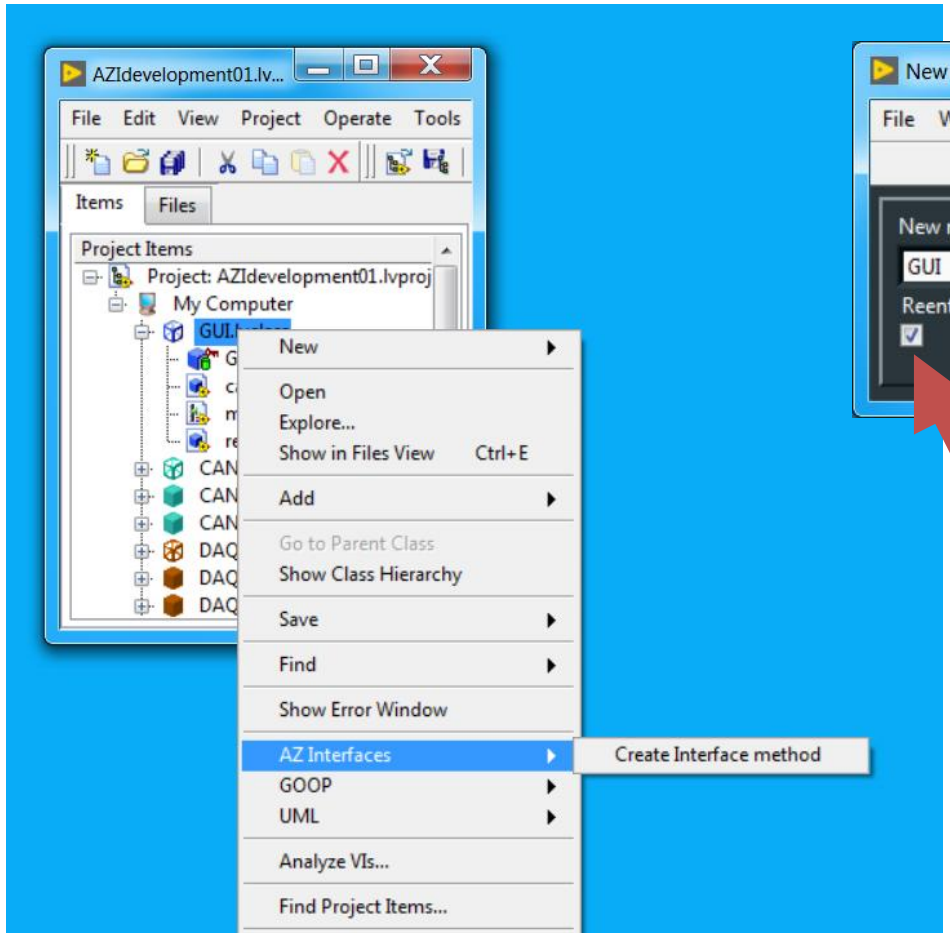
Is it it?

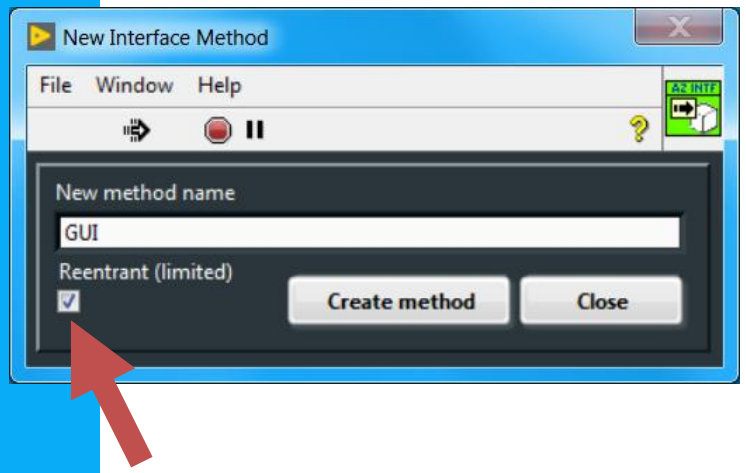
Not yet

Reentrant VI-s

Real challenge!

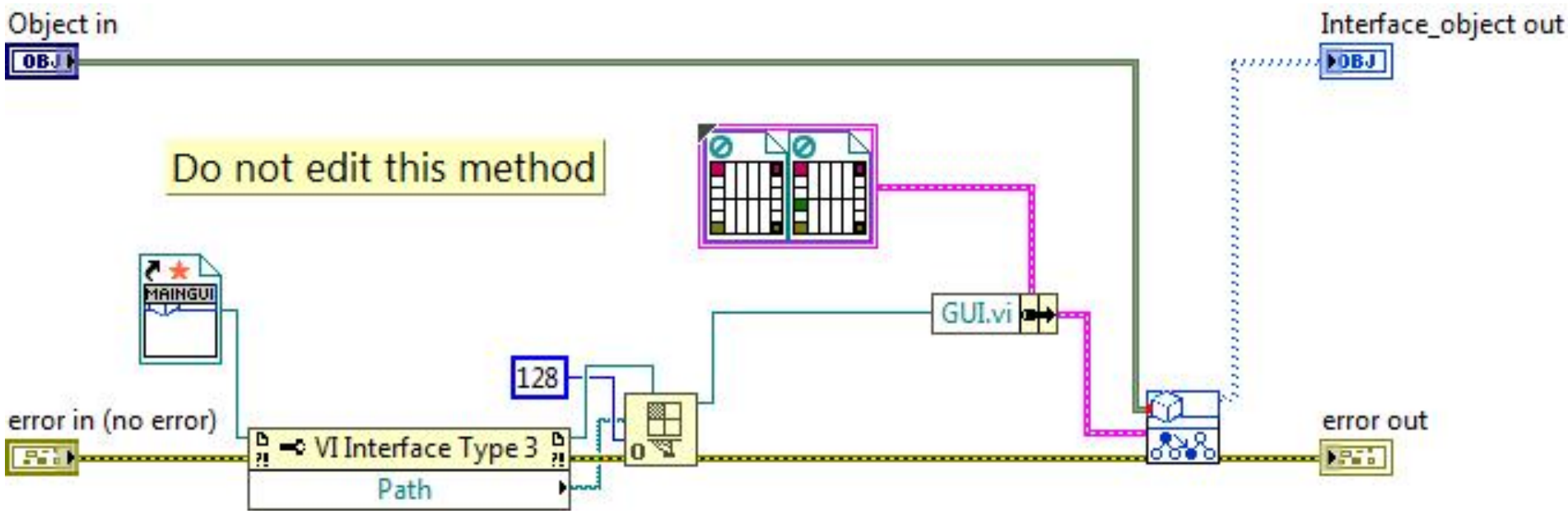
New Reentrant Interface Method: Create



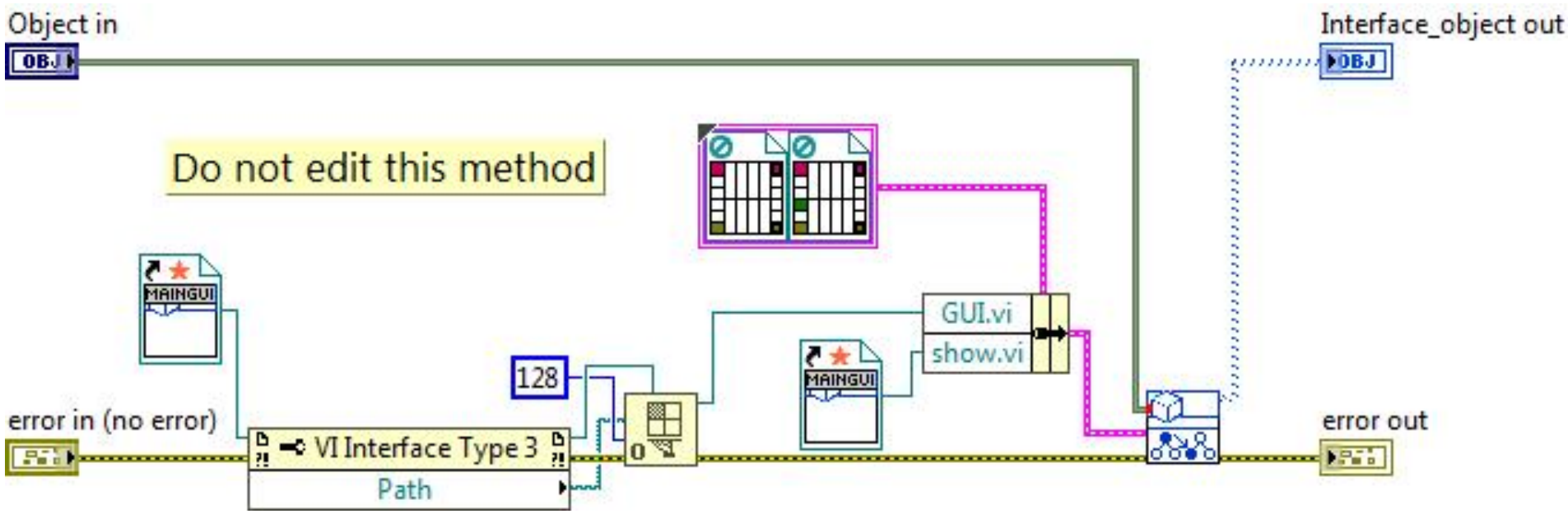


- Shared clone reentrant execution is selected for this method
- All corresponding class methods shall have the same reentrancy setting
  - That is good to be frank

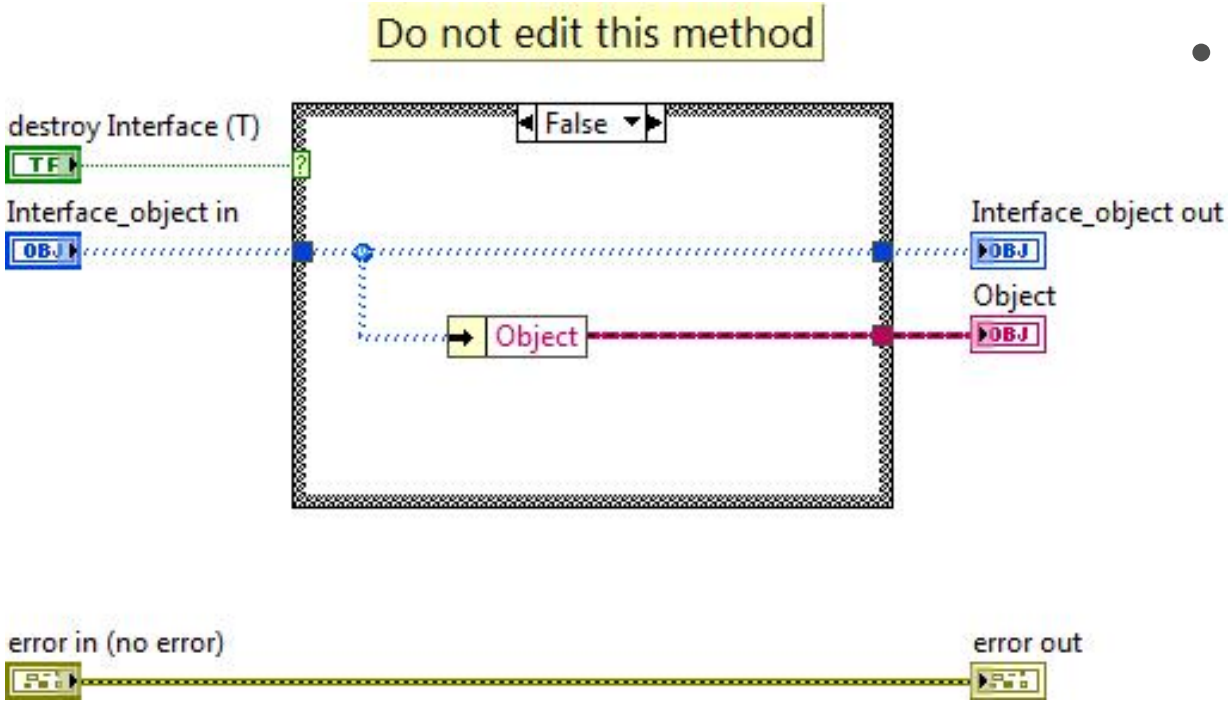
Method cast_to_<interface name>.vi



Method cast_to_<interface name>.vi

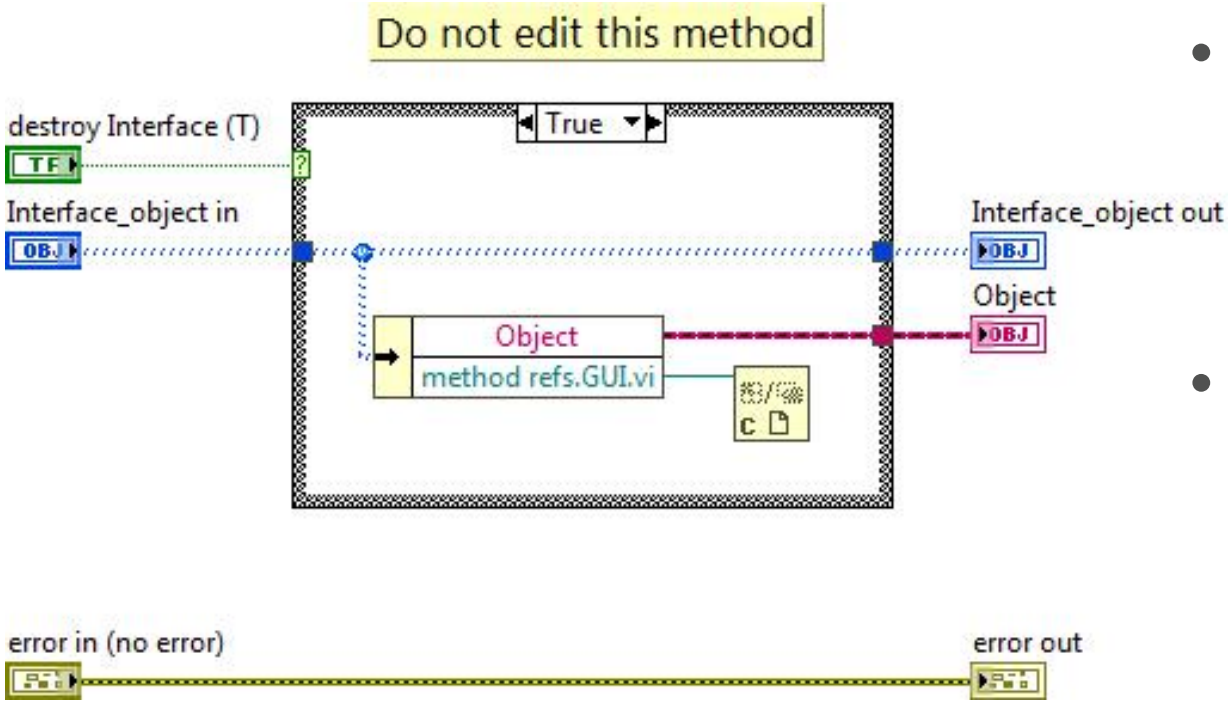


Interface method read_Object.vi



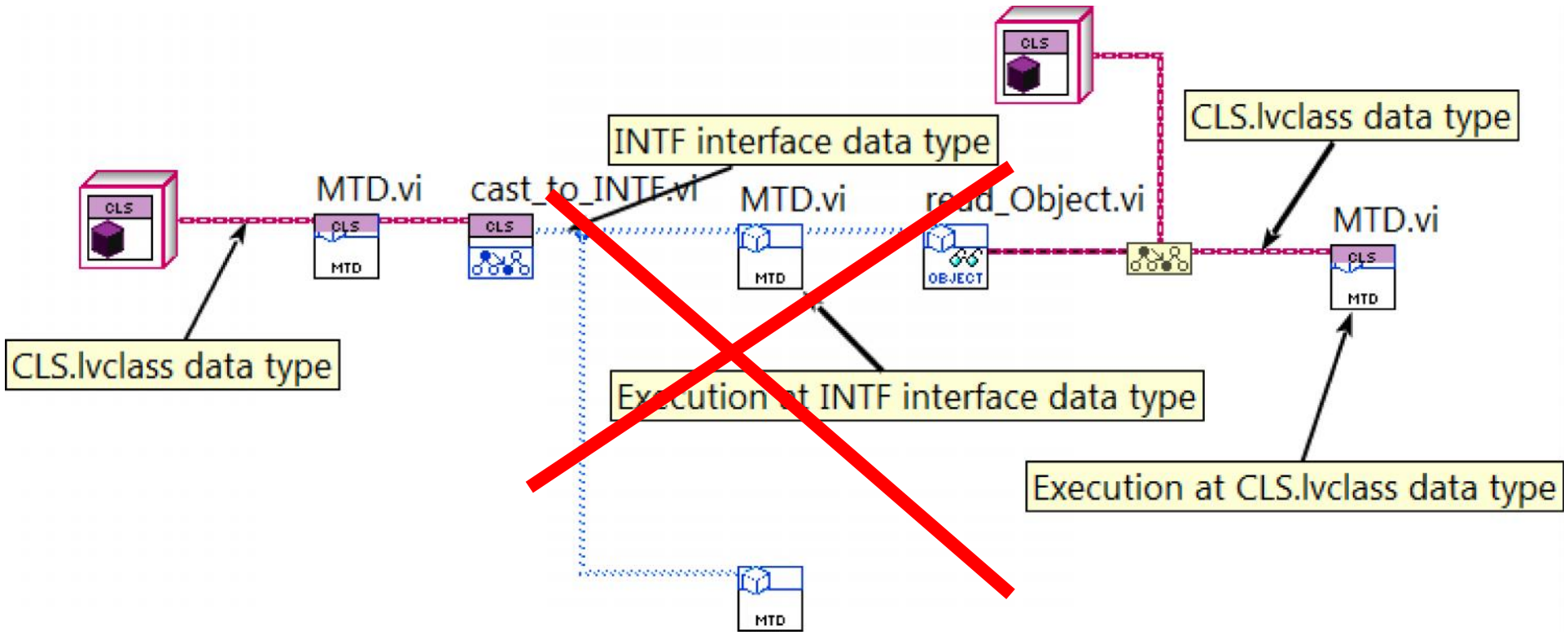
- If Interface has no reentrant methods, both cases (False and True) contain the same code.

Interface method read_Object.vi

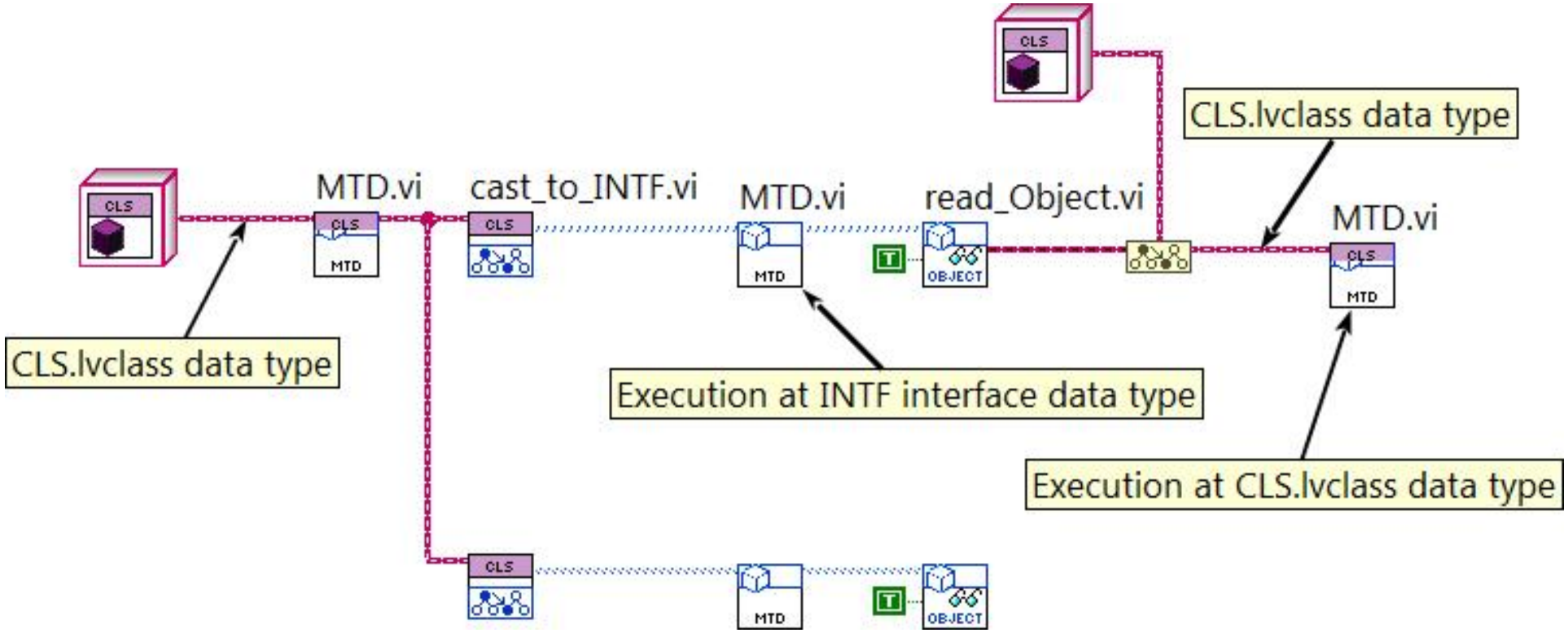


- If Interface has no reentrant methods, both cases (False and True) contain the same code.
- Case True closes references to reentrant methods
  - Thus read_Object.vi works as destructor
  - It MUST be used when the interface instance is not needed any more

Reentrant method execution at interface abstraction level



Reentrant method execution at interface abstraction level



We create another copy that must be destroyed

Is it it?

Almost

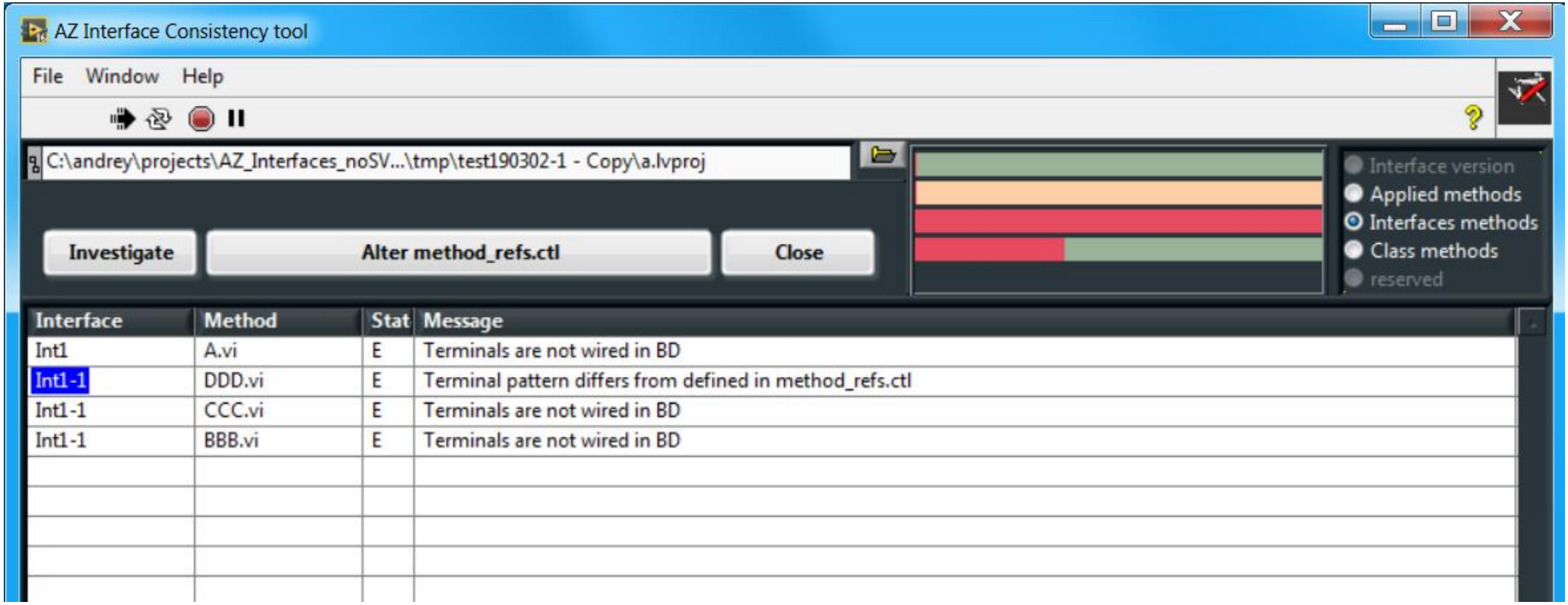
One more advantage

- You can apply the same Interface within the same project to any type of class
  - GOOP4 classes
  - GOOP3 classes
  - G# classes
  - Native LVOOP classes
- This means you can create common abstraction level for any combination of OOP models

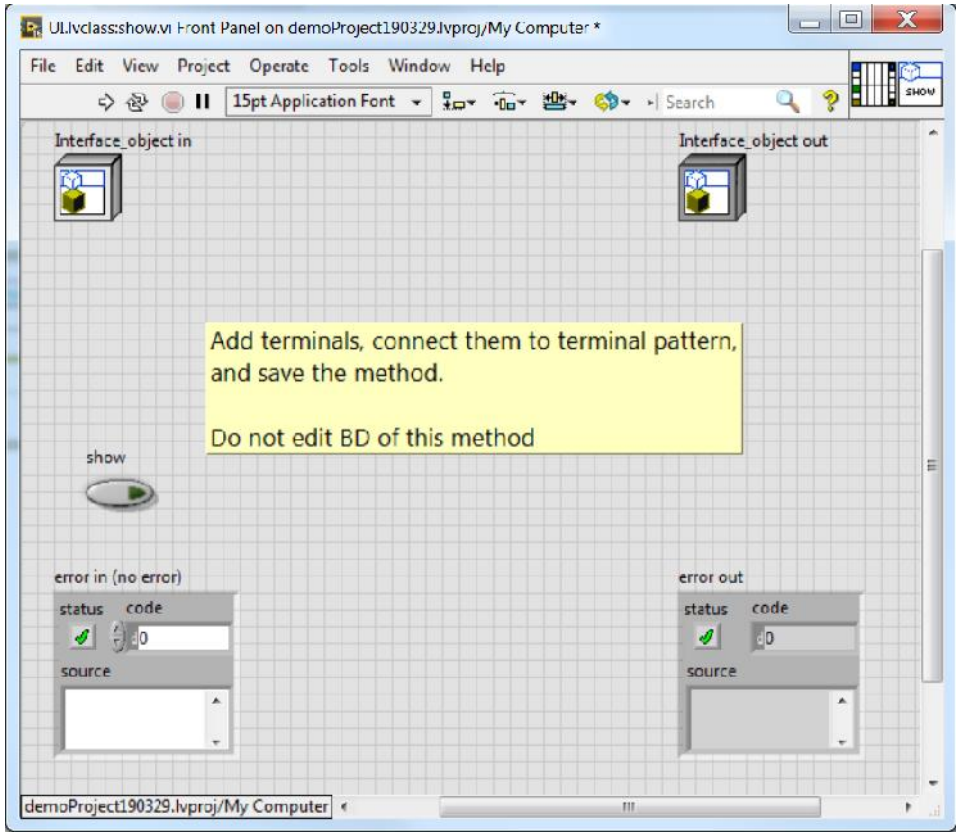
Consistency Tool

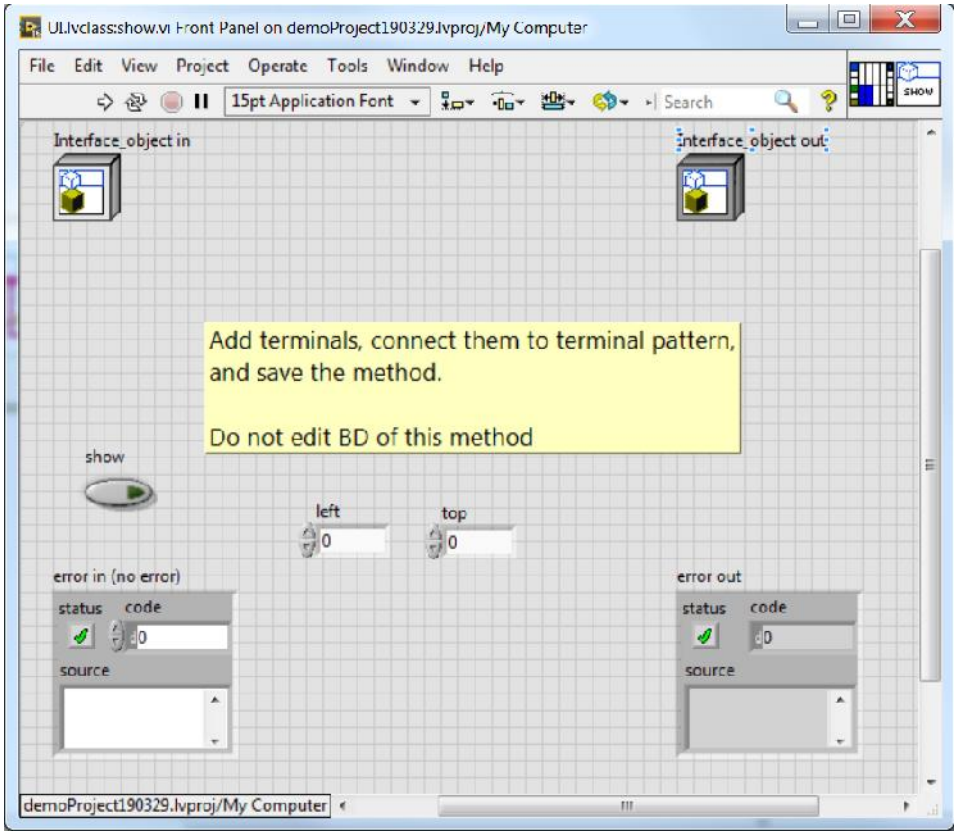
- I had presented Consistency tool when we discussed prototype version (v.0.0.0) of AZInterface at CLA-E 2018.
- There was no Consistency tool any more in v.1 and v.2.0 because there was no need in it.
- But probably we need it.
- There is Consistency tool in v.2.1.

Consistency Tool

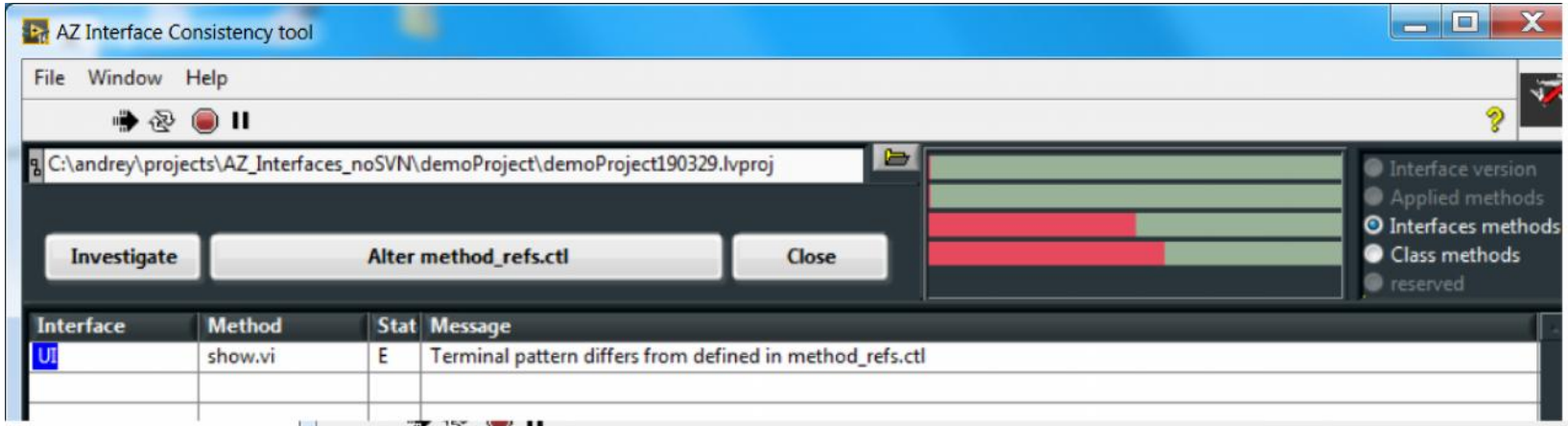


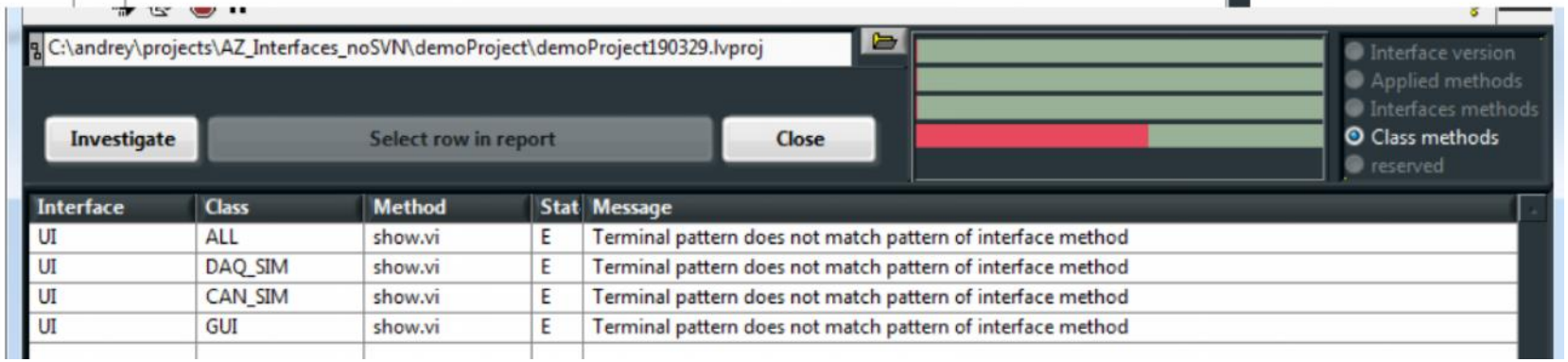
Consistency Tool: Alter terminal pattern of AZI method





Fix Interface method first and Class methods after





It is it

Thank you!

www.azinterface.net