AZ Multi-inheritance

version 1.0

Andrei Zagorodni

2022-11-23

Content

1 Introduction

AZ Multi-inheritance (**AZM**) is solution and toolkit for implementing multiple inheritance in LabVIEW OOP projects.

The solution is based on native LabVIEW interfaces; thus, applicable to versions starting from LabVIEW 2020.

Attention! Multiple inheritance is extremely powerful OOP technique. As any powerful programming technique, it opens possibility to create excellent architectural solutions as well as errors that are difficult to identify. Thus using described tools (**AZM**) and approaches is recommended only to advanced users with significant OOP experience.

1.1 Conventions

1.1.1 Lexicon, shortenings, and abbreviators

Abbreviator	Description
Ancestor	Any class from which considered class inherits. The word is convenient to define indirect inheritance; i.e. parents of parents.
AZM	AZ Multi-inheritance
`[name of XYZ]`	User-selected name
BD	Block Diagram
enum	enumerated data type
G#	Alternative toolkit providing LabVIEW with by-reference OOP programming features.
GOOP	Concept and toolkit providing LabVIEW with by-reference OOP programming features.
i-Class	**Interface**-based class specific for **AZM**
Interface	The word is used in this document solely for LabVIEW OOP **Interfaces**.
`[LabVIEW]`	Location of LabVIEW in this computer; for example **`C:\Program Files (x86)\National Instruments\LabVIEW 2020\`**
OOP	Object-Oriented Programming
Parent	Any class from which considered class inherits. The word is convenient to define direct inheritance; i.e. closest parents of the class.
RTE	Run-Time Engine
SW	Software; AZM software

1.1.2 Font conventions

- **Bold** is used for anything that appears literally in a LabVIEW environment or in LabVIEW program. For example, for menu, labels that cannot be altered.
- *Italic* is used for terms and messages.
- `Constant Width` is used for values: paths, names, etc.
- **`Constant Width Bold`** is used for values: paths, names, etc. that cannot or must not be altered.
- `[ ]` – brackets surround selectable parts of paths, names, etc.
- *Green Italic* is used for my personal notes.

1.2 Versions

Version number consists of four values:

1. *version* - altered with major changes causing compatibility and/or conceptual issues;
2. *subversion* – altered with introduction of major changes;
3. *fix* – minor changes, f. ex. a bug fix or minor performance improvement;
4. *build* – has meaning only for developer; f. ex. allows accounting of development packages, special assemblies, etc.

Altered *version* or *subversion* can cause a need in reading updated manual, while altered *fix* or *build* does not affect the way of use.

1.2.1 Version 0.0.0-pre-alpha

First functional version of the toolkit. The version is prepared in connection to presentation at GDevCon #3, Amsterdam, 9th Sept 2022.

Attention! This version is a proof of concept. It could undergo dramatic changes after obtaining feedbacks. *I highly appreciate any feedback /Andrei Zagorodni*

1.2.2 Version 1.0.0-alpha

First release of the toolkit.

2 AZ Multi-inheritance Background Ideas

2.1 Solutions

AZ Multi-inheritance (AZM) is based on following:

- Type-defined attribute cluster is stored as DVR (similarly to **GOOP4**).
- The DVR-s are mapped to object instances.
- The map key has LabVIEW **Interface** datatype.
- Only **GOOP4** objects are cast to the **i-Class** type (**i-Classes** are assembled by **AZM**).
- The map is stored in FGV.

2.2 Features

- AZM provides **GOOP4** classes with multiple parent classes called **i-Classes**.
- Conventional **GOOP4** ancestor class and ancestor **i-Classes** are two types of parentship.
- **GOOP4**, **i-Classes**, and **Interfaces** implemented as parents simultaneously do not create any conflict.
- LabVIEW code created with toolkit can be opened, edited, and run without installation of the toolkit. The code is not limited to LabVIEW development environment; corresponding EXE-files can be run under conventional LabVIEW RTE.

2.3 Limitations

- Current version is tested only for My Computer branch of LabVIEW Project. Use of the toolkit with other targets is implemented but not verified yet.
- **AZM** is applicable only to **GOOP4** classes. It can probably be used with **GOOP3** and G# classes but such feature has not been tested and not implemented yet.
- **AZM** cannot be used with native LabVIEW classes.
- Classes assembled by **AZM** (**i-Classes**) are abstract and cannot be instantiated.
- **i-Classes** and conventional **GOOP4** classes cannot be converted to each other.

3 System Requirements and Installation

3.1 Requirements

- Current version of **AZM** is developed for LabVIEW 2020 and expected to be fully functional with following LabVIEW versions.
- No additional package is required.
- **AZM** concept and toolkit cannot be downgraded to earlier LabVIEW versions.
- Current version of **AZM** works with **GOOP4** classes. Thus, this document is targeted to developers familiar with **GOOP4**.
- Installation of GOOP development suite is not required. *However, I will be very surprised if one uses AZM toolkit without GOOP toolkit /Andrei Zagorodni*

3.2 Installation

No installer is supplied with current version of the toolkit. Files must be manually copied in corresponding LabVIEW directories.

Files belonging to older version of **AZM** must be deleted before installation.

3.2.1 File location

Files must be copied in different directories of LabVIEW. The table below refers to `[LabVIEW]` directory, for example to,

`C:\Program Files (x86)\National Instruments\LabVIEW 2020\`

Content of the following source directories must be copied into corresponding target directories.

Supplied files	Target LabVIEW directory
`GProviders`	`[LabVIEW]`**`\resource\Framework\Providers\GProviders\`**
`Providers`	`[LabVIEW]`**`\resource\Framework\Providers\`**
`Project`	`[LabVIEW]`**`\resource\Framework\project\`**
`help`	`[LabVIEW]`**`\help\`**

3.2.2 Recompiling

In some cases, files of the toolkit must be recompiled after the copying; f. ex. VIs must be re-saved accounting to new locations of sub-VIs.

To do it open consequently three VIs. These VIs are used only for manual installation. Ignore messages concerning altered file locations. Order of opening could be important:

1. Open LabVIEW.
2. Open
   `[LabVIEW]\help\AZ Interfaces\`
   `_1_all_help_AZ-MultiInheritance.vi`
3. Open
   `[LabVIEW]\resource\Framework\Providers\AZ_Interfaces\`
   `_2_all_providers_AZ-MultiInheritance.vi`
4. Open
   `[LabVIEW]\project\AZ-MultiInheritance\`
   `_3_all_project_AZ-MultiInheritance.vi`
5. Click menu **File** > **Save All**.
6. Close all VI-s.
7. Restart LabVIEW.

4 Primary Functions of AZ Multi-inheritance toolkit

Note: When working with **AZM** all involved files must be available for modifications. Remove write-protection from involved **i-Classes**, **GOOP4** classes, and all their members.

4.1 Creating i-Class

1. Create **Interface** or select **Interface** existing in the project.
2. Right-click the Interface and select menu **AZ-Multi-inheritance > Convert to class**.

The **Interface** will be supplied with two virtual folders and five new members:

- `utils/[Interface name]_Attributes.vi` – utility method; holding object attributes in uninitialized shift register (private).
- `utils/[Interface name]_GetAttributes.vi` – attribute accessor with functionality similar to corresponding member of **GOOP4** class (protected).
- `protected/ObjectAttributes.ctl` – cluster defining object attributes in the way similar to **GOOP4** (protected).
- `protected/[Interface name]_Create.vi` – object constructor similar to **GOOP4** (protected).
- `protected/[Interface name]_Destroy.vi` – non-dynamic-dispatch object destructor (protected).

Attention! Conversion of an **Interface** to **i-Class** does not affect classes that already inherit from this **Interface**. Corresponding changes must be implemented in the code using **Consistency tool** (see section 4.3) or manually.

4.2 Setting i-Class as parent class

1. Right-click **GOOP4** class and select menu **AZ-Multi-inheritance > Set parent**.
2. Parent-selection GUI is opened. The GUI is described in section 4.2.1.
3. Select parent and click Set parent

- Inheritance from selected **i-Class** or **Interface** will be set.

- `[i-Class name]_Attributes.vi` with corresponding enum options will be added to code of constructor and destructor BD-s.

4.2.1 Selecting parent

Parent selection GUI is shown in Figure 1.

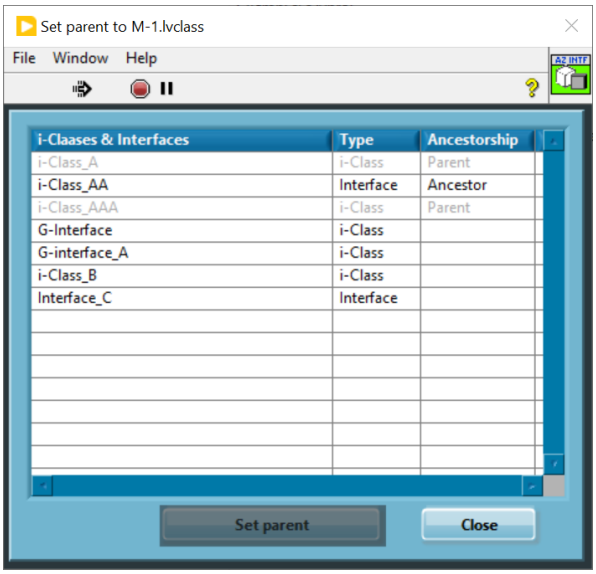

Figure 1 Example of newly created *AZI method*

- Column **i-Classes & Interfaces** shows all **Interfaces** in the project: both “as is” and **converted to i-Classes.**
- Column **Type** indicates this difference.
- **Parent** in Column **Ancestorship** shows if selected class already inherits from the **i-Class**/**Interface**. **Ancestor** in this column indicates that the inheritance is already exists but via one or more members in class hierarchy.

Difference between Parent and Ancestor is illustrated by Figure 2.

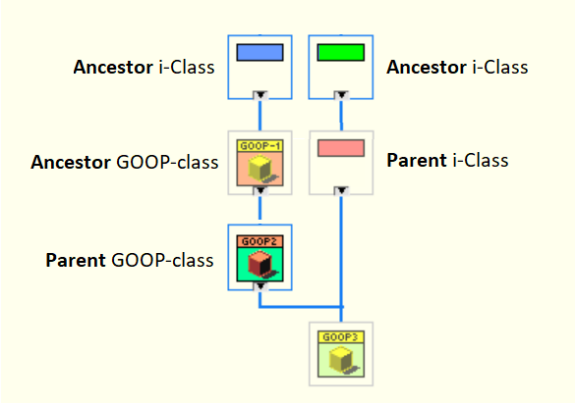

Figure 2 Example of newly created *AZI method*

Note: If column **Ancestorship** contains word **Ancestor**, implementation of direct inheritance (changing ancestor to parent) is not forbidden. However, this does not add any functionality to the class while adding accessors to its constructor and destructor.

4.3 Consistency control

- Each **i-Class** class inheriting from ancestor **i-Class** must call one constructor and one destructor belonging to closest in the class hierarchy **i-Class**. Both (constructor and destructor) are copies of the same accessor, provided with corresponding enum options. They are located in BD of `[successor name]_Attributes.vi`.
- Each **GOOP4** directly inheriting from **i-Class** must call one **i-Class** constructor per class constructor BD and one **i-Class** destructor in the destructor BD.

The consistency can be controlled with **AZM** consistency tool.

Simple cases of inconsistency can be repaired with the same tool. More complicated cases must be fixed manually.

4.3.1 Using consistency tool

1. Select menu **Tools > AZ-Multi-inheritance > Consistency tool…** .
2. Use **opened GUI** to select project and target.

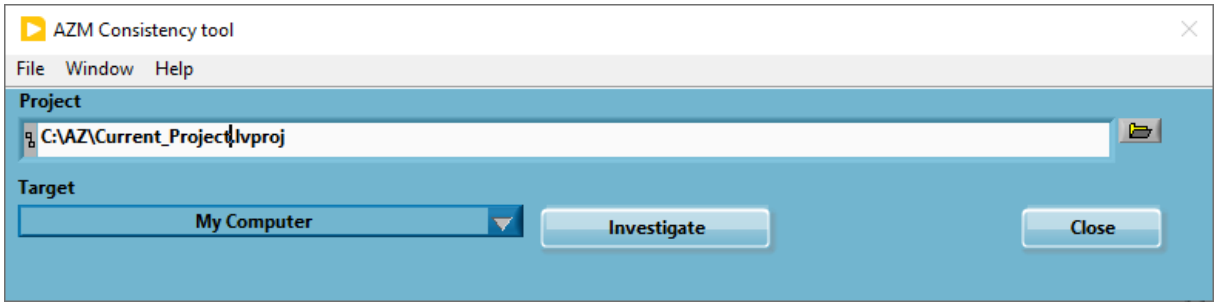

Figure 3 First GUI of Consistency tool

3. Click **Investigate i-Classes** or **Investigate classes**.
4. List of inconsistencies is shown. Corresponding dialog is described in section 4.3.2.
5. Select inconsistency and click **Repair**.

4.3.2 Investigation results GUI

Inconsistency list is shown in Figure 4.

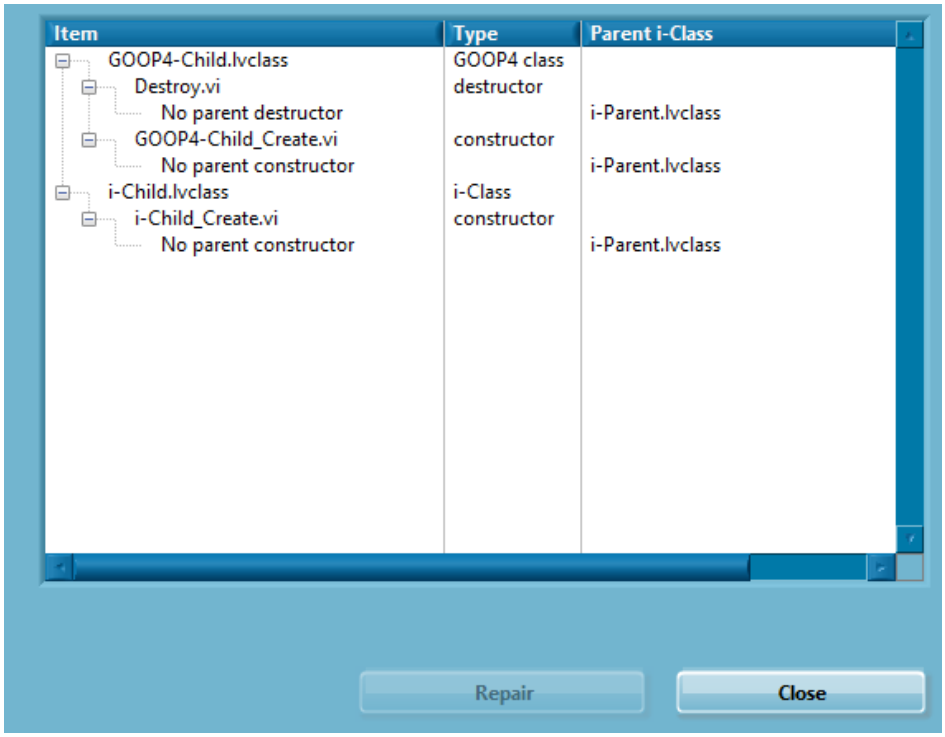

Figure 4 Example of investigation results

- First hierarchical level lists inconsistency-containing classes.
- First hierarchical level lists inconsistency-containing methods.
- Third level lists inconsistencies.

Column **Inconsistency** can have following values:

Inconsistency	Explanation	Expected action
Class has no constructor	Class constructor is missing.	**GOOP4** class can be repaired only manually while **i-Class** can be repaired automatically. The newly created **i-Class** constructor should be attended and further developed manually
Class has no destructor	Class destructor is missing.	**GOOP4** class can be repaired only manually while **i-Class** can be repaired automatically. The newly created **i-Class** destructor should be attended and further developed manually
No parent constructor	Constructor or accessor does not call **i-Class** accessor with option **Create**.	Can be repaired automatically.
No parent destructor	Destructor or accessor does not call **i-Class** accessor with option **Cleanup**.	Can be repaired automatically.
Parent has no constructor	Parent **i-Class** constructor is missing.	Can be repaired automatically selecting corresponding problem presented for the parent **i-Class** in the same GUI.
Parent has no destructor	Parent **i-Class** destructor constructor is missing.	Can be repaired automatically selecting corresponding problem presented for the parent **i-Class** in the same GUI.

5 Using i-Classes

5.1 Specificity of i-Class code

5.1.1 Order of parent constructors and destructors

Calls of parent **i-Class** constructor and destructor are similar to corresponding calls of **GOOP4** class members:

- BD of child class constructor must call parent constructor.
- BD of child class destructor must call parent destructor.

I.e., constructor of any class inheriting from **i-Class** must call constructor of the parent class

AZM toolkit adds parent **i-Class** constructors to child class constructor and parent **i-Class** destructor to child class destructor. This is true for both **GOOP4** and **i-Class** child classes.

If class inherits from an **Interface** while the **Interface** inherits from an **i-Class**, calls to constructor and destructor of the ancestor **i-Class** are added to child class BD-s. I.e., presence of the **Interface** "between" the child and the ancestor is ignored.

5.1.2 Order of parent constructors and destructors

If class inherits from multiple **i-Classes**, order of calls for the parent constructors/destructors is not established by AZM toolkit. If the order is important for particular application, it has to be established manually.

Calls of parent of **i-Class** constructors must be placed after utility `[child_class_name]_New.vi`.

5.1.3 Repeated calls of constructor and destructor

Multiple inheritance allows creation of sophisticated class hierarchy. A common case is a class inheriting from an ancestor **i-Class** "twice or more" through different direct parents, see Figure 5. Repeated calls of **i-Class** constructor/destructor have no effect.

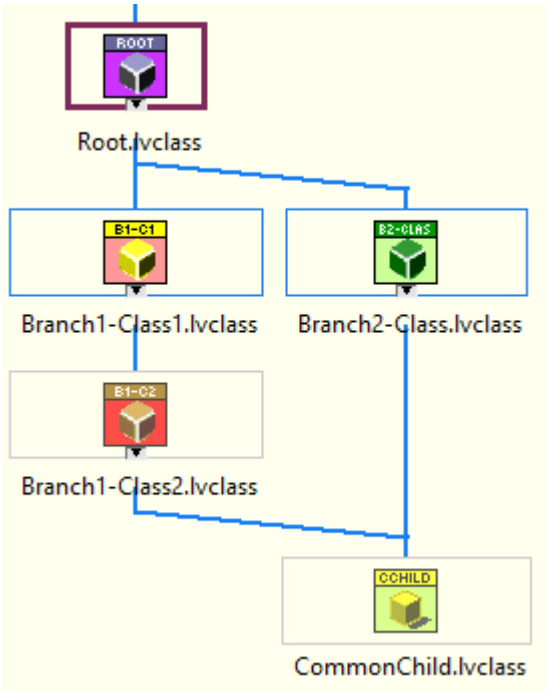

Figure 5 Example a class (`CommonChild`) inheriting from same ancestor (`Root`) two times.

I.e., programmer should not worry about behavior of constructors/destructors in complicated hierarchies.

6 About and Contacts

Figure 6 About

6.1 License Agreement

1 Acknowledgement

CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE. BY USING THIS FREEWARE VERSION YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LIMITED WARRANTY, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT UNLESS YOU HAVE A DIFFERENT LICENSE AGREEMENT SIGNED BY ANDREI ZAGORODNI YOUR USE OF THIS SOFTWARE INDICATES YOUR ACCEPTANCE OF THIS LICENSE AGREEMENT AND WARRANTY. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DELETE THE SOFTWARE FROM ALL STORAGE MEDIA.

2 License

This Freeware License Agreement (the "Agreement") is a legal agreement between you ("Licensee"), the end-user, and developer of AZ Multi-inheritance Toolkit Andrei Zagorodni ("Developer") for the use of this software product ("Software"). Commercial as well as non-commercial use is allowed. By using this Software or storing this program or parts of it on a computer hard drive (or other media), you agree to be bound by the terms of this Agreement. Provided that you verify that you are handling the original freeware version you are hereby licensed to make as many copies of the freeware version of this Software and documentation. You can alter this Software in any way but Developer does not carry any responsibility for consequences.

If you alter and/or further develop this Software, documentation (including "help" and "about") must include reference to original Software, name of its Developer and his contacts.

3 Limited Warranty and Disclaimer of Warranty

The AZ Multi-inheritance Toolkit EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE. THIS SOFTWARE AND THE ACCOMPANYING FILES ARE PROVIDED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED, OR NONINFRINGEMENT. THIS SOFTWARE IS NOT FAULT TOLERANT AND SHOULD NOT BE USED IN ANY ENVIRONMENT WHICH REQUIRES THIS. NO LIABILITY FOR DAMAGES. In no event shall AZ Multi-inheritance Toolkit or its suppliers be liable for any consequential, incidental or indirect damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) resulting of the use of or inability to use this SOFTWARE EVEN IF the Software HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The entire risk resulting of use or performance of the SOFTWARE remains with you.

4 Copyright

6.2 Contacts

Andrei Zagorodni

andrei.zagorodni@novatorsolutions.se

Please write **AZI** or **AZ Multi-inheritances** in subject line.

6.3 Support and communications

I shall appreciate feedback about bugs and bottlenecks identified in this SW.

I promise to read your emails and reply within reasonable time. However, the project is developed in my evenings and weekends. Thus the "reasonable time" will solely depend on my workload.

You are free to modify code of the software. However, I do not promise to support the modified code.

Andrei Zagorodni