# About OPLA-ArchStyles Project

Available in https://github.com/thainamariani/OPLA-ArchStyles.git

## Dependencies

This project is a module of OPLA-Tool, which is needed to make OPLA-ArchStyles works. You must download the OPLA-Tool project at https://github.com/edipofederle/architecture-representation.git

## Organization of Packages

**config, templates and perfis:** have some files needed for OPLA-Tool to work;

**lib:** has the needed jars;

**test:** has the project tests;

**src:** this package is divided in the following packages:

- **project:** has all the essential classes of the project;
- **gui:** at the time, there is no graphic interface to support OPLA-ArchStyles. That way, this package contains some support classes with static information to do, for now, the graphic interface role;
- **experiment:** has some support classes to execute the experiments;
- **results:** has some support classes to calculate the results.

## Running

For running, you must pass to the experiment class (src.experiment.Experiment), respectively, the following information as parameters:

- Population size (example: 100);
- Max Evaluations (example: 10000);
- Mutation Probability (example: 0.9);
- Absolute PLA path (example: /home/plas/agm/agm.uml);
- The architectural style to be used. Such value must be layer, clientserver, aspect, or any word if you do not want to use styles (example: layer).
- The folder to be saved (example: test).

If you want to preserve aspects in addition to the architectural style given as parameter, then the boolean parameter aspect in the Experiment class must be true. Otherwise, such value must be false.

If the architectural style given as parameter is aspect, this boolean value is unnecessary and ignored.

After that, the architectural style informed is validate in the PLA. If it is not designed correctly, a message is showed as output. Otherwise, the PLA is optimized.

## What is necessary to create a graphic user interface (GUI)?

### Layered Architectural Style

For the layered architectural style the suffixes and/or prefixes contained in the packages to identify the layers are needed. In addition to this, the layers hierarchy must be given. The hierarchy of layers must be upwards. An example is given in the table below, when Layer 1 is the lowest layer and Layer 3 is the highest layer. Also, there might be multiple suffixes and prefixes, but in this example there is only one suffix for each layer.

| Layers | Suffixes | Prefixes |
|---|---|---|
| **Layer 3** | Gui | |
| **Layer 2** | Ctrl | |
| **Layer 1** | Mgr | |

Based on the above table, 3 objects *Layer* must be instantiated as following.

```
LayerIdentification layerIdentification = new LayerIdentification(architecture);
List<Layer> camadas = new ArrayList<>();
Layer layer1 = new Layer();
layer1.setNumero(1);
List<String> sufixos = new ArrayList<>();
List<String> prefixos = new ArrayList<>();
sufixos.add("Mgr");
layer1.setSufixos(sufixos);
layer1.setPrefixos(prefixos);
camadas.add(layer1);

Layer layer2 = new Layer();
layer2.setNumero(2);
List<String> sufixos2 = new ArrayList<>();
List<String> prefixos2 = new ArrayList<>();
sufixos2.add("Ctrl");
layer2.setSufixos(sufixos2);
```

```
layer2.setPrefixos(prefixos2);
camadas.add(layer2);

Layer layer3 = new Layer();
layer3.setNumero(3);
List<String> sufixos3 = new ArrayList<>();
List<String> prefixos3 = new ArrayList<>();
sufixos3.add("Gui");
layer3.setSufixos(sufixos3);
layer3.setPrefixos(prefixos3);
camadas.add(layer3);

boolean isCorrect = layerIdentification.isCorrect(camadas);
```

The method *isCorrect* of the class *LayerIdentification* receive a list of the layers and returns if the style is designed correctly. The *LayerIdentification* class must be instantiated receiving as parameter an *ArchitectureBuilder* object (for more information see the class *src.gui.StyleGui*).

### Client/Server Architectural Style

For the client/server architectural style the suffixes and/or prefixes contained in the packages to identify the clients and servers are needed. An example is given in the table below, where there are 2 servers and 1 client.

| Component | Suffixes | Prefixes |
|---|---|---|
| **Server** | Server1 | |
| **Server** | Server2 | |
| **Client** | Client1 | |

Based on the table above, 2 objects Server and 1 object Client must be instantiated as following.

```
ClientServerIdentification clientServerIdentification = new ClientServerIdentification(architecture);
 List<Client> clients = new ArrayList<>();
 Client client1 = new Client();
 List<String> sufixos = new ArrayList<>();
 List<String> prefixos = new ArrayList<>();
 sufixos.add("Client1");
```

```
client1.setSufixos(sufixos);
client1.setPrefixos(prefixos);
clients.add(client1);

List<Server> servers = new ArrayList<>();
Server server1 = new Server();
List<String> sufixos4 = new ArrayList<>();
List<String> prefixos4 = new ArrayList<>();
sufixos4.add("Server1");
server1.setSufixos(sufixos4);
server1.setPrefixos(prefixos4);
servers.add(server1);

Server server2 = new Server();
List<String> sufixos5 = new ArrayList<>();
List<String> prefixos5 = new ArrayList<>();
sufixos5.add("Server2");
server2.setSufixos(sufixos5);
server2.setPrefixos(prefixos5);
servers.add(server2);

List<Style> clientsservers = new ArrayList<>();
clientsservers.addAll(clients);
clientsservers.addAll(servers);
boolean isCorrect clientServerIdentification.isCorrect(clientsservers);
```

### Aspect Style

For using the aspect style the profile aspect.profile.uml is needed. Such profile can be found in the folder *perfis*, along with the other profiles used by OPLA-Tool. To add this profile in the code the following line is required.

```
ReaderConfig.setPathToProfileAspect(absoluteDirectory +"/aspect.profile.uml")
```

Another way is to add such profile in the application.yaml file. To model PLAs with the aspect.profile.uml follow the same rules of modeling with other profiles. Details about the notation used in the profile can be found in the master thesis:

*T. Mariani. Uma proposta de operadores para preservar o estilo de arquiteturas de linha de produto de software no projeto baseado em busca. Dissertação de mestrado em andamento, Universidade Federal do Paraná, Curitiba, PR, 2014.*

### Add the SO4ARS operators

If the style used (layer, client/server or aspect) is designed correctly according to the methods of the identification classes (*LayerIdentification*, *ClientServerIdentification*, *AspectIdentification*), the SO4ARS operators must be added. The class of the operator that extends the *Mutation* class is *PLAFeatureMutationConstraints*. To add this operator the following line must be executed, where the style parameter can be “layer”, “clientserver” or “aspect”.

```
mutation = MutationFactory.getMutationOperator("PLAFeatureMutationConstraints", parameters, style);
```

---

To see more helpful information on how to create the graphic interface and to add the SO4ARS operator, you can see the class *Experiment* (*src/experiment/Experiment*), focusing on lines 165 to 203.