

The ADORE Class Factory Evaluator (CFE)

Class Factory Evaluator | 2026-01-22

Jasper Bussemaker (jasper.bussemaker@dlr.de)

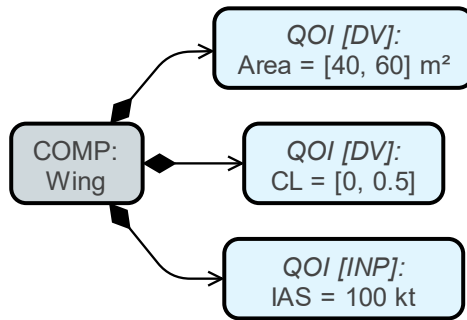
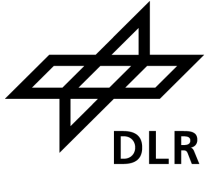
DLR Institute of System Architectures in Aeronautics (SL), Hamburg, Germany



<https://adore.mbse-env.com/docs/>



The Class Factory Evaluator (CFE)



```
from typing import *
from dataclasses import dataclass
import adore as ad
```

```
@dataclass
class Wing:
    area: float
    cl: float
    ias: float
```

```
class MyClassFactoryEvaluator(ad.ClassFactoryApiEvaluator):
    """Problem-specific Python-based Class Factory Evaluator."""
```

```
def get_class_factories(self) -> List[ad.ClassFactory]:
    return [ad.ClassFactory(
        el=ad.ExternalComponentDef(name='Wing', auto=True),
        cls=Wing,
        props={
            'area': ad.ExternalQOIDef(
                name='Area', auto=True, units='m^2'),
            'cl': ad.ExternalQOIDef(
                name='Lift coefficient', auto_match_pattern='CL'),
            'ias': ad.ExternalQOIDef(
                name='IAS', auto=True, units='kts'),
        }
    )]
```

Class Factories

Rules for linking to ADORE elements and instantiating objects for each occurrence in an architecture instance.

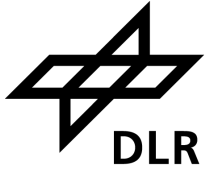
```
classes = {list: 1} [Wing(area=50.0, cl=0.25, ias=100.0)]
  0 = {Wing} Wing(area=50.0, cl=0.25, ias=100.0)
    01 area = {float} 50.0
    01 cl = {float} 0.25
    01 ias = {float} 100.0
    01 __len__ = {int} 1
```

```
def _evaluate(self, architecture, arch_qois, **_):
    # Get Wing objects from the architecture
    wings = self.instantiate(
        architecture, factories=self.get_factories_by_el_name('Wing'))
    ...
```

Instantiation

Instantiate objects in the `_evaluate` function.

Defining Class Factories



1. Assumption: the evaluation framework is available and its input is defined using Python classes

2. Create an evaluator inheriting from ClassFactoryApiEvaluator

3. Define ClassFactory's

- Note: values from QOIs are normally of **float** type; however they will be **strings** for:
 - Input parameter (string)
 - Design variable (discrete)

4. Define a MetricsFactory

```
from dataclasses import dataclass
from adore.optimization.api.factory_evaluator import *
from adore.api.schema import *
```

```
@dataclass
class Wing:
    area: float # m2
    cl: float
    ias: float # kts

    def calc_lift(self):
        ias_ms = self.ias/1.94384
        rho0 = 1.225 # kg/m3
        return self.cl * .5*rho0*ias_ms**2 * self.area
```

“Evaluation framework”

Inputs

Calculations

```
class WingClassFactoryEvaluator(ClassFactoryApiEvaluator):
```

```
@staticmethod
def get_class_factories() -> List[ClassFactory]:
    return [
        ClassFactory(
            el=ExternalComponentDef(name='Wing', n_inst=[1]),
            cls=Wing,
            props={
                'area': ExternalQOIDef(name='Wing Area', qoi_type=QOIType.DESIGN_VAR, bounds=(40., 60.)),
                'cl': ExternalQOIDef(name='Lift Coefficient', qoi_type=QOIType.DESIGN_VAR, bounds=(0., .5)),
                'ias': ExternalQOIDef(name='Indicated Airspeed', qoi_type=QOIType.INPUT_PARAM, value=100.),
            },
        )
    ]
```

ClassFactory
Defines a component
With properties (QOIs)

```
@staticmethod
def get_metrics_factory() -> MetricsFactory:
    return MetricsFactory(metrics={
        'lift': ExternalQOIDef(name='Lift', qoi_type=QOIType.CONSTRAINT, ref_value=15000*9.81, pos_better=True),
    })
```

MetricsFactory
Defines outputs (QOIs)

Automatically Linking Elements

1. Factories can be automatically linked to design space elements using **auto match patterns**

- Other links can still be created manually in the GUI

2. Patterns are applied on the *name* and *ref* fields of design space elements, and can be defined as:

- Wildcard patterns (case-insensitive): * or ? mean any or 1 character, respectively
- [Regular expressions](#): /expr/ (or /expr/i for case-insensitive checking)
- Same name as external element def: auto=True

```
class WingClassAutoFactoryEvaluator(ClassFactoryApiEvaluator):
```

```
@staticmethod
def get_class_factories() -> List[ClassFactory]:
    return [
        ClassFactory(
            el=ExternalComponentDef(name='Wing', n_inst=[1], auto_match_pattern='/W.*/' ),
            cls=Wing,
            props={
                'area': ExternalQOIDef(
                    name='Wing Area', qoi_type=QOIType.DESIGN_VAR, bounds=(40., 60.), auto_match_pattern='area' ),
                'cl': ExternalQOIDef(
                    name='Lift Coefficient', qoi_type=QOIType.DESIGN_VAR, bounds=(0., .5),
                    auto_match_pattern=['cl', 'L* Coefficient'])
                'ias': ExternalQOIDef(
                    name='Indicated Airspeed', qoi_type=QOIType.INPUT_PARAM, value=100. auto_match_pattern='i?s'
                )
            }
        ),
    ]

@staticmethod
def get_metrics_factory() -> MetricsFactory:
    return MetricsFactory(metrics={
        'lift': ExternalQOIDef(
            name='Lift', qoi_type=QOIType.CONSTRAINT, ref_value=15000*9.81, pos_better=True auto=True
        )
    })
```

3. Linking can be done from within Python (no need to switch between Python and the GUI)

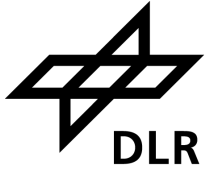
- After instantiating the evaluator, use the `update_external_databases` function to add the database and auto-match elements
- Save to file to inspect the project in the GUI
- Check console output to check matching was correct

```
if __name__ == '__main__':
    # 1) Instantiate evaluator
    evaluator = WingClassAutoFactoryEvaluator.from_file('Class_Factory_Evaluator_Unlinked.adore')

    # 2) Auto match elements
    evaluator.update_external_database()
    evaluator.to_file('auto_linked.adore')
```

```
Linking report: design space elements --> external element definitions
System.Wing --> <AUTO> Wing
System.Wing.Area --> <AUTO> Wing Area (input)
System.Wing.CL --> <AUTO> Lift Coefficient (input)
System.Wing.IAS --> <AUTO> Indicated Airspeed (input)
System.Produce Lift.Lift --> <AUTO> Lift (output)
```

Implementing the Evaluation Function



1. Extend the `_evaluate` function

- Use `instantiate` and `process_results` as needed
- You can select specific factories to be instantiated:
`wings = self.instantiate(architecture, factories=[self._class_factories[0]])`
- Or select the factories to be instantiated by name:
`wings = self.instantiate(architecture, factories="Wing")`

```
def _evaluate(self, architecture: Architecture, arch_qois: List[ArchQOI]) -> Dict[ArchQOI, float]:  
    # Instantiate classes: we expect one instance of Wing  
    objects = self.instantiate(architecture)  
    wing = objects[0]  
    assert isinstance(wing, Wing)  
  
    # Perform calculations  
    results = {  
        'lift': wing.calc_lift(),  
    }  
  
    # Interpret results so that ADORE can process them  
    return self.process_results(architecture, arch_qois, results)
```

1. Run architecture evaluation as it would be in an optimization loop

- Architecture can also be a manually generated architecture

```
evaluator = WingClassFactoryEvaluator.from_file('Class_Factory_Evaluator_Example.adore')  
architecture, dv, is_active = evaluator.get_architecture(evaluator.get_random_design_vector())  
evaluator.evaluate(architecture)
```

3. Check results

```
> dv = {list: 2} [42.05919669032886, 0.4295716065789159]  
> is_active = {list: 2} [True, True]
```

```
architecture = {Architecture} Architecture(33, name='New Design Problem: 42.06, 0.43')  
constraints = {list: 1} [ArchConstraint(45, name='Lift')]  
0 = {ArchConstraint} ArchConstraint(45, name='Lift')  
  arch_qoi_id = {int} 37  
  constraint_id = {int} 32  
  id = {int} 45  
  name = {str} 'Lift'  
  notes = {NoneType} None  
  pos_feasible = {bool} True  
  ref = {str} 'lift'  
  ref_value = {float} 147150.0  
  value = {float} 29287.45304160498
```

Advanced: Class Factory Settings



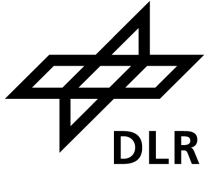
- A ClassFactory “means”:
 - Instantiate cls **for each** architecture element linked to el
 - With properties defined by props
- Notes:
 - props is optional
 - Values can also be hardcoded
 - Use SpecialValue to assign some value of the main element
 - Property values are taken from “contained” elements (e.g. QOIs)
 - Except if only_contained_props=False

```
@staticmethod
def get_class_factories() -> List[ClassFactory]:
    return [
        ClassFactory(
            el=ExternalComponentDef(name='Wing', n_inst=[1]),
            cls=Wing,
            props={
                'area': ExternalQOIDef(name='Wing Area', qoi_type=QOI
                'cl': ExternalQOIDef(name='Lift Coefficient', qoi_typ
                'ias': ExternalQOIDef(name='Indicated Airspeed', qoi_
            },
        ),
    ]
```

```
@dataclass
class MyQOIClass:
    name: str
    el_id: int
    element: ArchElement
    value: float
    fixed_value: float
```

```
ClassFactory(
    el=ExternalQOIDef(name='QOI'),
    cls=MyQOIClass,
    props={
        'name': SpecialValue.Name,
        'el_id': SpecialValue.Id,
        'element': SpecialValue.El,
        'value': SpecialValue.Value,
        'fixed_value': 50,
    },
    only_contained_props=False,
)
```

Advanced: Control Which Classes Get Instantiated



- You can instantiate only selected classes by passing the appropriate class factories to instantiate (or instantiate_by_owner)
- You can use self.get_factories_by_el_name function to search for class factories by name; pattern matching is supported (e.g. "Component *")
- You can also directly access available class factories in self._class_factories

```
@dataclass
class Engine:
    rpm: float
```

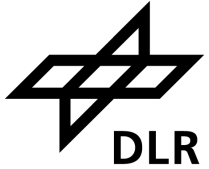
```
@dataclass
class Fuselage:
    length: float
```

```
class MultiClassFactoryEvaluator(ClassFactoryApiEvaluator):
```

```
    @staticmethod
    def get_class_factories() -> List[ClassFactory]:
        return [
            ClassFactory(
                el=ExternalComponentDef(name='Engine Comp'),
                cls=Engine,
                props={
                    'rpm': ExternalQOIDef(
                        name='RPM QOI', qoi_type=QOIType.DESIGN_VAR,
                        bounds=(2000, 3000)),
                },
                only_contained_props=True,
            ),
            ClassFactory(
                el=ExternalComponentDef(name='Fuselage Comp'),
                cls=Fuselage,
            ),
        ]
```

```
    def _evaluate(self, architecture: Architecture, arch_qois: List[ArchQOI]) -> Dict[ArchQOI, float]:
        engines = self.instantiate(architecture, factories=[self._class_factories[0]])
        fuselages = self.instantiate(architecture, factories=[self._class_factories[1]])
```

Advanced: Overriding Element Properties



```
@staticmethod
def get_class_factories() -> List[ClassFactory]:
    return [
        ClassFactory(
            el=ExternalComponentDef(name='Wing', n_inst=[1]),
            cls=Wing,
            props={
                'area': ExternalQOIDef(name='Wing Area', qoi_type=QOIType.DESIGN_VAR, bounds=(40., 60.)),
                'cl': ExternalQOIDef(name='Lift Coefficient', qoi_type=QOIType.DESIGN_VAR, bounds=(0., .5)),
                'ias': ExternalQOIDef(name='Indicated Airspeed', qoi_type=QOIType.INPUT_PARAM, value=100.),
            },
        ),
    ]

@staticmethod
def get_metrics_factory() -> MetricsFactory:
    return MetricsFactory(metrics={
        'lift': ExternalQOIDef(name='Lift', qoi_type=QOIType.CONSTRAINT, ref_value=15000*9.81, pos_better=True),
    })
```

Note: if you don't restrict the type of QOI, only **input** types (if defined as part of a class factory) or **output** types (if defined as part of the metrics factory) can be selected for the linked QOI

Name
Wing

Reference
wing

Linked external element
Wing X

☐ Disable element
Removes the element from the design space

☐ External element
Places the element in the system context

Number of possible instances
1

Quantity of Interest type
Design Variable

☐ Positive is better
Interpret greater numbers as better

Reference value

Design variable bounds
40,60 X

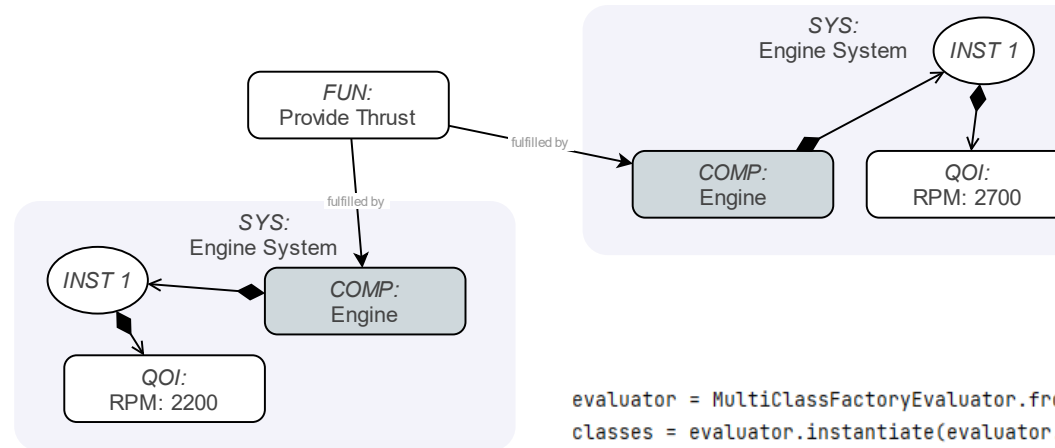
Quantity of Interest type
Constraint

☒ Constrain greater than
Constraint is satisfied if the value is greater than the reference

Reference value
147150 X

Advanced: Track Element Owner Chain

- QOIs are owned by functions or components
- Elements are owned by systems
- Systems are owned by systems
- Use `instantiate_by_owner` to get ownership chain information
- Same example, but now with 2 system instances

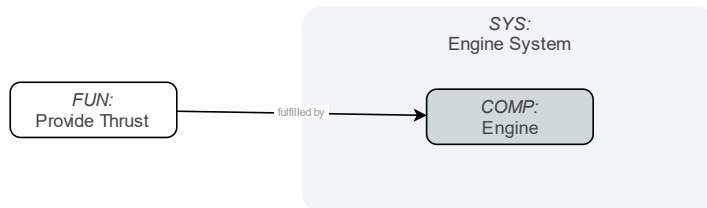


```
evaluator = MultiClassFactoryEvaluator.from_file('Multi_System_Element_Example.adore')
classes = evaluator.instantiate(evaluator.project.architectures[0])
```

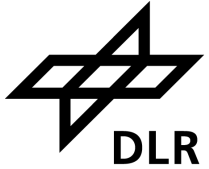
```
> classes = {list: 2} [Engine(rpm=2700.0), Engine(rpm=2200.0)]
> 0 = {Engine} Engine(rpm=2700.0)
> 1 = {Engine} Engine(rpm=2200.0)
01 _len_ = {int} 2
```

```
evaluator = MultiClassFactoryEvaluator.from_file('Multi_System_Element_Example.adore')
classes_and_ownership = evaluator.instantiate_by_owner(evaluator.project.architectures[0])
```

```
> classes_and_ownership = {list: 2} [(Engine(rpm=2700.0), [Architecture(63, name='New Architecture'), ArchSystem(65, name='System'), ArchSystem(67, name='Engine System'), ArchComponent(69, name='Engine')]),
  (Engine(rpm=2200.0), [Architecture(63, name='New Architecture'), ArchSystem(65, name='System'), ArchSystem(68, name='Engine System'), ArchComponent(72, name='Engine')])]
> 0 = {tuple: 2} (Engine(rpm=2700.0), [Architecture(63, name='New Architecture'), ArchSystem(65, name='System'), ArchSystem(67, name='Engine System'), ArchComponent(69, name='Engine')])
> 0 = {Engine} Engine(rpm=2700.0)
> 1 = {list: 4} [Architecture(63, name='New Architecture'), ArchSystem(65, name='System'), ArchSystem(67, name='Engine System'), ArchComponent(69, name='Engine')]
01 _len_ = {int} 2
> 1 = {tuple: 2} (Engine(rpm=2200.0), [Architecture(63, name='New Architecture'), ArchSystem(65, name='System'), ArchSystem(68, name='Engine System'), ArchComponent(72, name='Engine')])
> 0 = {Engine} Engine(rpm=2200.0)
> 1 = {list: 4} [Architecture(63, name='New Architecture'), ArchSystem(65, name='System'), ArchSystem(68, name='Engine System'), ArchComponent(72, name='Engine')]
01 _len_ = {int} 2
01 _len_ = {int} 2
```



Advanced: Sharing External Element Definitions



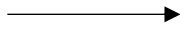
- You can share external element definitions between class factories
- The shared element will only be added to the external database once

```
class SharedClassFactoryEvaluator(ClassFactoryApiEvaluator):  
  
    @staticmethod  
    def get_class_factories() -> List[ClassFactory]:  
  
        shared_qoi = ExternalQOIDef(name='Shared Input')  
  
        return [  
            ClassFactory(  
                el=ExternalComponentDef(name='Component1'),  
                cls=MyObject,  
                props={  
                    'input': shared_qoi,  
                },  
            ),  
            ClassFactory(  
                el=ExternalComponentDef(name='Component2'),  
                cls=MyObject,  
                props={  
                    'input': shared_qoi,  
                },  
            ),  
        ]
```


- ```

 'arch_el': SpecialValue.El,
 'all': ConnectionValue(conn_target_def=all_defs),
 'inputs': ConnectionValue(conn_target_def=all_defs),
 'outputs': ConnectionValue(conn_target_def=all_defs)
 }),
 ClassFactory(el=comp_def, cls=ConnObj, props={...}),
 ClassFactory(el=concept_def, cls=ConnObj, props={...}),
 ClassFactory(el=de_def, cls=ConnObj, props={...}),
 ClassFactory(el=nf_def, cls=ConnObj, props={...}),
 ClassFactory(el=mf_def, cls=ConnObj, props={...}),

```



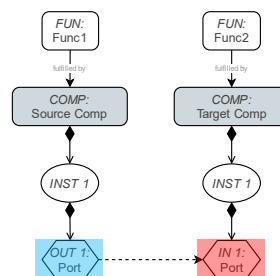
```

1 comp_conn_objs = [list: 3] [ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0'), all=[ConnObj{arch_el=ArchFunction(24, name='Function 3'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0')
2 | inputs = [list: 2] [ConnObj{arch_el=ArchFunction(24, name='Function 3'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0')
3 | outputs = [list: 1] [ConnObj{arch_el=ArchFunction(26, name='Function 5')}, all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0')
4 | 1 = (ConnObj) ConnObj{arch_el=ArchComponentInstance(30, name='Component 1 (x2) 1'), all=[ConnObj{arch_el=ArchComponentInstance(30, name='Component 1 (x2) 1')
5 | 2 = (ConnObj) ConnObj{arch_el=ArchComponentInstance(32, name='Component 2 0'), all=[ConnObj{arch_el=ArchComponentInstance(32, name='Component 2 0')
6 | _len_ = (int) 3
7
8 func_conn_objs = [list: 5] [ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1')
9 | 0 = (ConnObj) ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1'), all=[ConnObj{arch_el=ArchFunction(22, name='Function 1')
10 | 1 = (ConnObj) ConnObj{arch_el=ArchFunction(23, name='Function 2'), all=[ConnObj{arch_el=ArchFunction(23, name='Function 2'), all=[ConnObj{arch_el=ArchFunction(23, name='Function 2'), all=[ConnObj{arch_el=ArchFunction(23, name='Function 2')
11 | 2 = (ConnObj) ConnObj{arch_el=ArchFunction(24, name='Function 3'), all=[ConnObj{arch_el=ArchFunction(24, name='Function 3'), all=[ConnObj{arch_el=ArchFunction(24, name='Function 3'), all=[ConnObj{arch_el=ArchFunction(24, name='Function 3')
12 | all = [list: 3] [ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0')
13 | arch_el = (ArchFunction) ArchFunction(24, name='Function 3')
14 | inputs = [list: 1] [ConnObj{arch_el=ArchDecomposition(34, name='Decomposition'), all=[ConnObj{arch_el=ArchDecomposition(34, name='Decomposition'), all=[ConnObj{arch_el=ArchDecomposition(34, name='Decomposition')
15 | outputs = [list: 2] [ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0'), all=[ConnObj{arch_el=ArchComponentInstance(29, name='Component 1 (x2) 0')
16 | 3 = (ConnObj) ConnObj{arch_el=ArchFunction(25, name='Function 4'), all=[ConnObj{arch_el=ArchFunction(25, name='Function 4'), all=[ConnObj{arch_el=ArchFunction(25, name='Function 4')
17 | 4 = (ConnObj) ConnObj{arch_el=ArchFunction(26, name='Function 5'), all=[ConnObj{arch_el=ArchFunction(26, name='Function 5'), all=[ConnObj{arch_el=ArchFunction(26, name='Function 5')
18 | _len_ = (int) 5

```

# Advanced: Port Connections

- Used for tracking port connections
- Use ConnectionValue to set a class property to a list of instantiated classes corresponding to:
  - Component connection targets
  - Or ports if return\_ports=True
- Optionally filter by input or output connections using input\_conn
- Note: components and ports should all be correctly linked to their respective external elements



```
class PortExampleEvaluator(ClassFactoryApiEvaluator):

 @staticmethod
 def get_class_factories() -> List[ClassFactory]:
 port_def = ExternalPortDef(name='ExtPort')
 return [
 ClassFactory(
 el=port_def, cls=PortObj, props={
 'name': SpecialValue.Name,
 },
),
 ClassFactory(
 el=ExternalComponentDef(name='Source'), cls=SourceObj,
 props={
 'targets': ConnectionValue(conn_target_def=port_def, input_conn=False),
 'target_ports': ConnectionValue(conn_target_def=port_def, return_ports=True),
 },
),
 ClassFactory(
 el=ExternalComponentDef(name='Target'), cls=TargetObj,
 props={
 'sources': ConnectionValue(conn_target_def=port_def, input_conn=True),
 'source_ports': ConnectionValue(conn_target_def=port_def, return_ports=True),
 },
),
]

 def _evaluate(self, architecture: Architecture, arch_qois: List[ArchQOI]) -> Dict[ArchQOI, float]:
 port_objs = self.instantiate(architecture, factories=self.get_factories_by_el_name('ExtPort'))
 src_objs = self.instantiate(architecture, factories=self.get_factories_by_el_name('Source'))
 tgt_objs = self.instantiate(architecture, factories=self.get_factories_by_el_name('Target'))
```

```
> port_objs = {list: 1} [PortObj(name='Port')]
> 0 = {PortObj} PortObj(name='Port')
01 _len_ = (int) 1
> src_objs = {list: 1} [SourceObj(targets=[TargetObj(sources=[...], source_ports=[PortObj(name='Port')]), target_ports=[PortObj(name='Port')])])
> 0 = {SourceObj} SourceObj(targets=[TargetObj(sources=[...], source_ports=[PortObj(name='Port')]), target_ports=[PortObj(name='Port')])])
> target_ports = {list: 1} [PortObj(name='Port')]
> targets = {list: 1} [TargetObj(sources=[SourceObj(targets=[...], target_ports=[PortObj(name='Port')]), source_ports=[PortObj(name='Port')])])
01 _len_ = (int) 1
> tgt_objs = {list: 1} [TargetObj(sources=[SourceObj(targets=[...], target_ports=[PortObj(name='Port')]), source_ports=[PortObj(name='Port')])])
> 0 = {TargetObj} TargetObj(sources=[SourceObj(targets=[...], target_ports=[PortObj(name='Port')]), source_ports=[PortObj(name='Port')])])
> source_ports = {list: 1} [PortObj(name='Port')]
> sources = {list: 1} [SourceObj(targets=[TargetObj(sources=[...], source_ports=[PortObj(name='Port')]), target_ports=[PortObj(name='Port')])])
01 _len_ = (int) 1
```