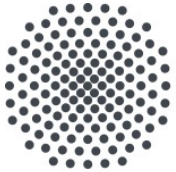
Integrating Artifact Translation into Model Transformation Processes

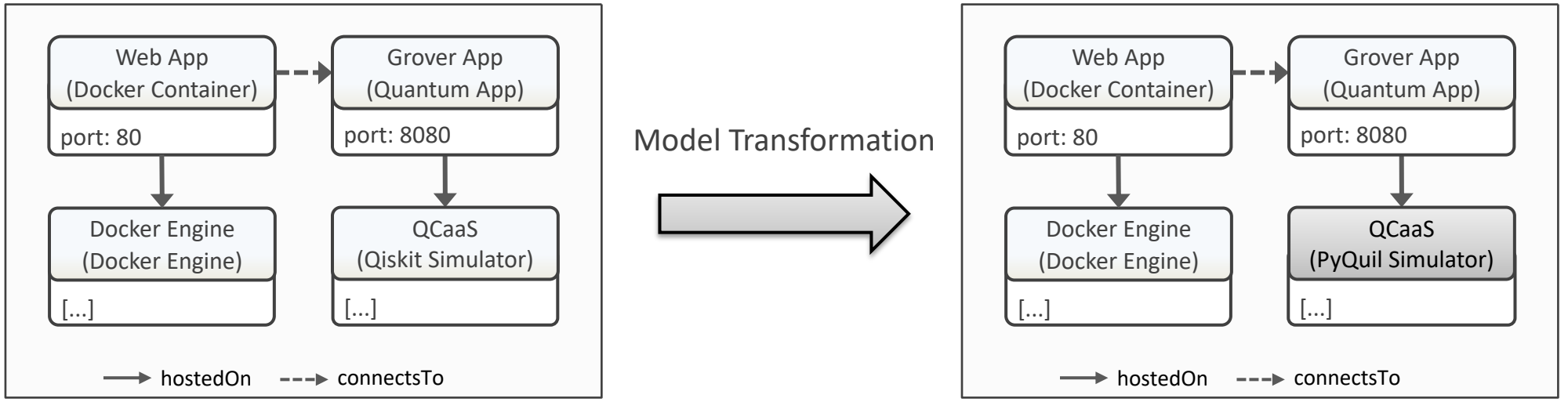


University of Stuttgart

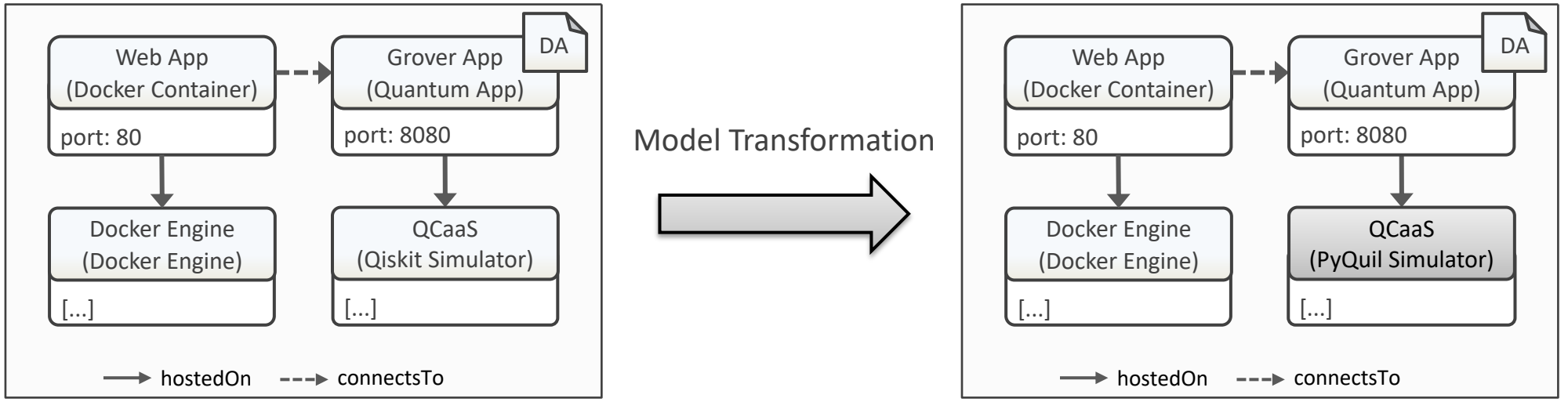
Daniel Vietz, Johanna Barzen, Lukas Harzenetter
Frank Leymann, Benjamin Weder

lastname@iaas.uni-stuttgart.de
Institute of Architecture of Application Systems

Motivational Example

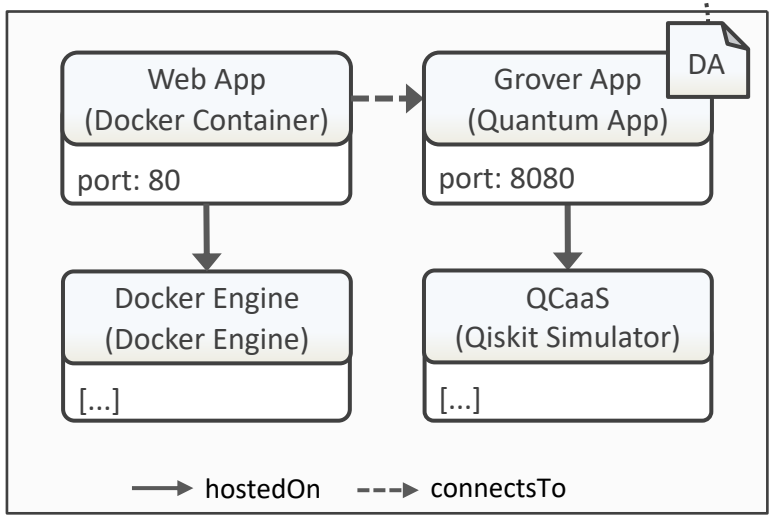


Motivational Example

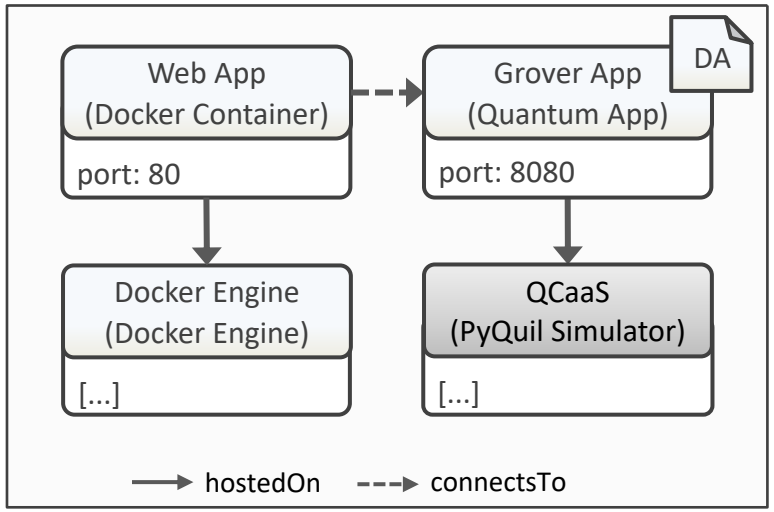


Motivational Example

```
from qiskit import QuantumCircuit,
     QuantumRegister,
ClassicalRegister
from qiskit import execute

q = QuantumRegister(3, ‘q’)
c = ClassicalRegister(3, ‘c’)
circ = QuantumCircuit(q, c)
circ.h(q)
...
```

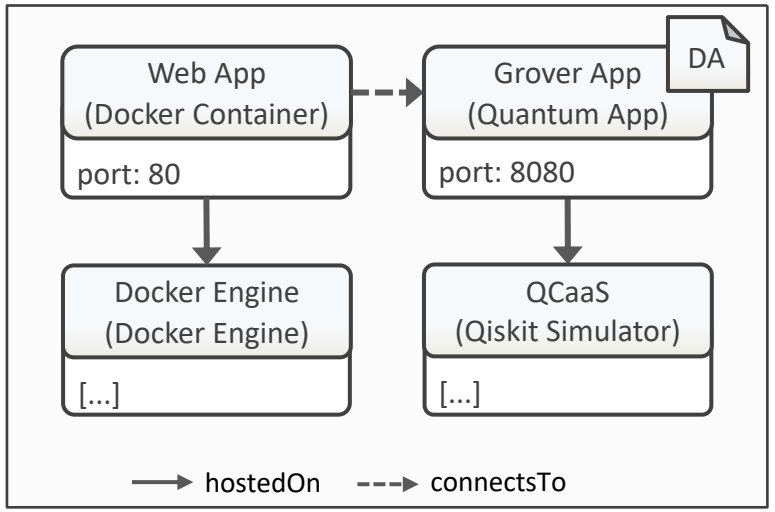


Model Transformation

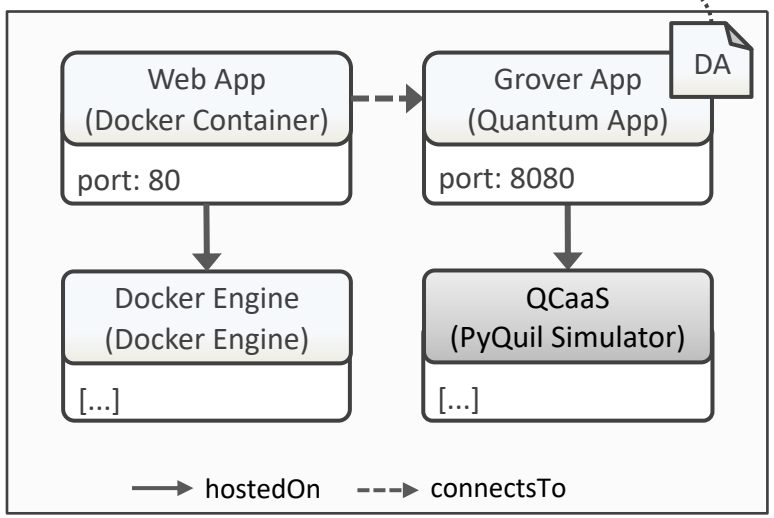


Motivational Example

```
from qiskit import QuantumCircuit,
     QuantumRegister,
ClassicalRegister
from qiskit import execute

q = QuantumRegister(3, ‘q’)
c = ClassicalRegister(3, ‘c’)
circ = QuantumCircuit(q, c)
circ.h(q)
...
```



Model Transformation

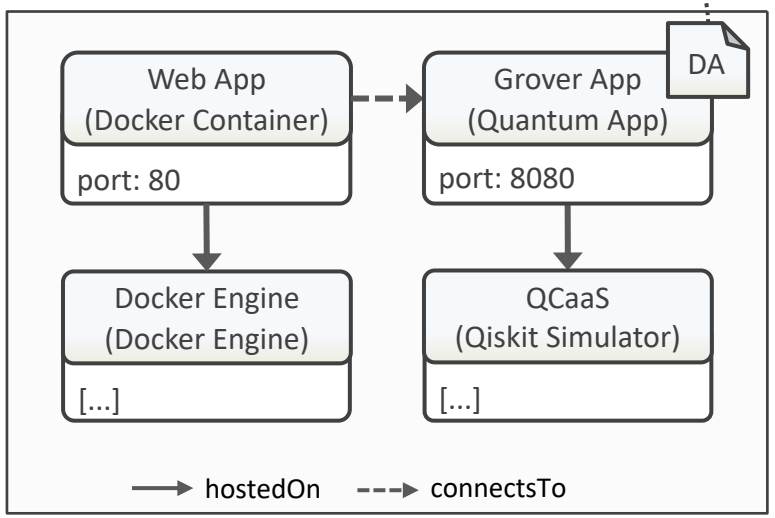


Motivational Example

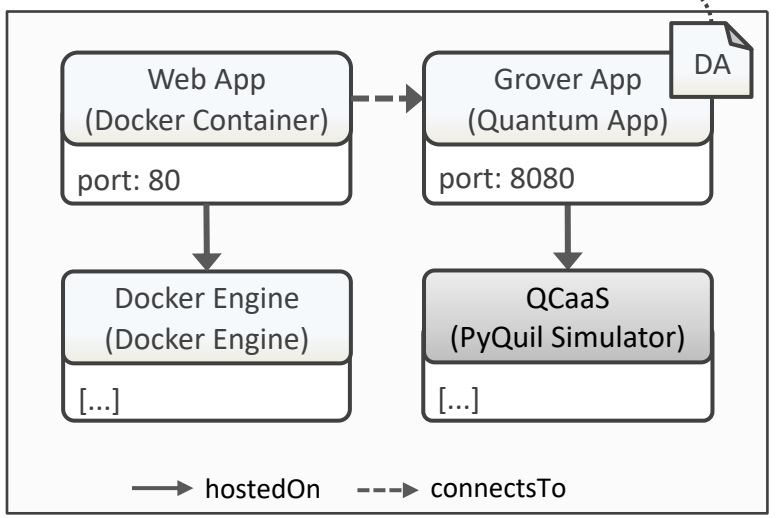
```
from qiskit import QuantumCircuit,
     QuantumRegister,
ClassicalRegister
from qiskit import execute

q = QuantumRegister(3, ‘q’)
c = ClassicalRegister(3, ‘c’)
circ = QuantumCircuit(q, c)
circ.h(q)
...
```

Translate

```
from pyquil import Program, get_qc
from pyquil.gates import H, CNOT

p = Program()
p += H(0)
...
```



Model Transformation



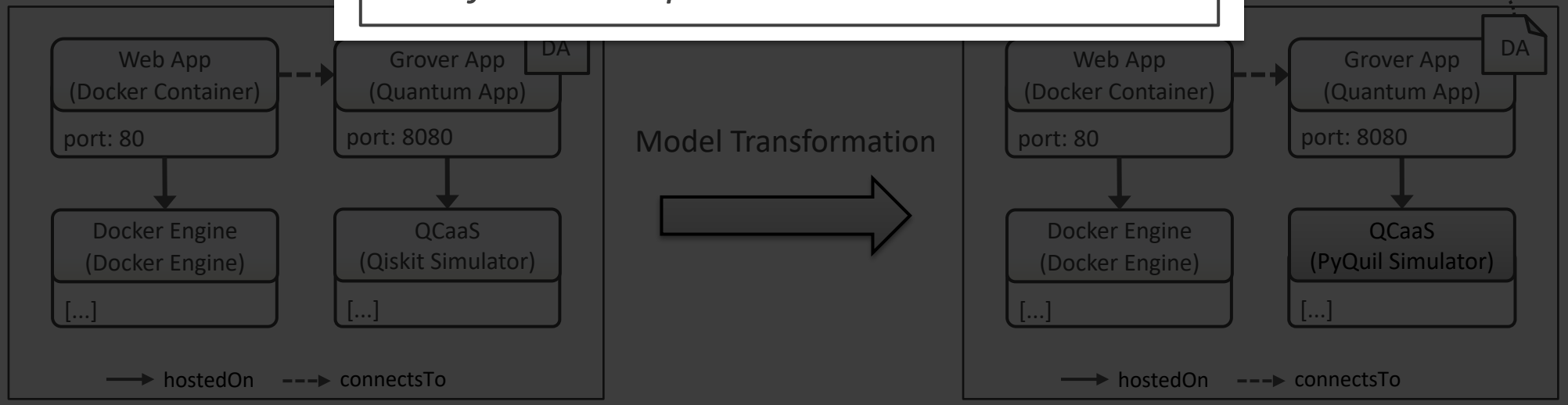
Motivational Example

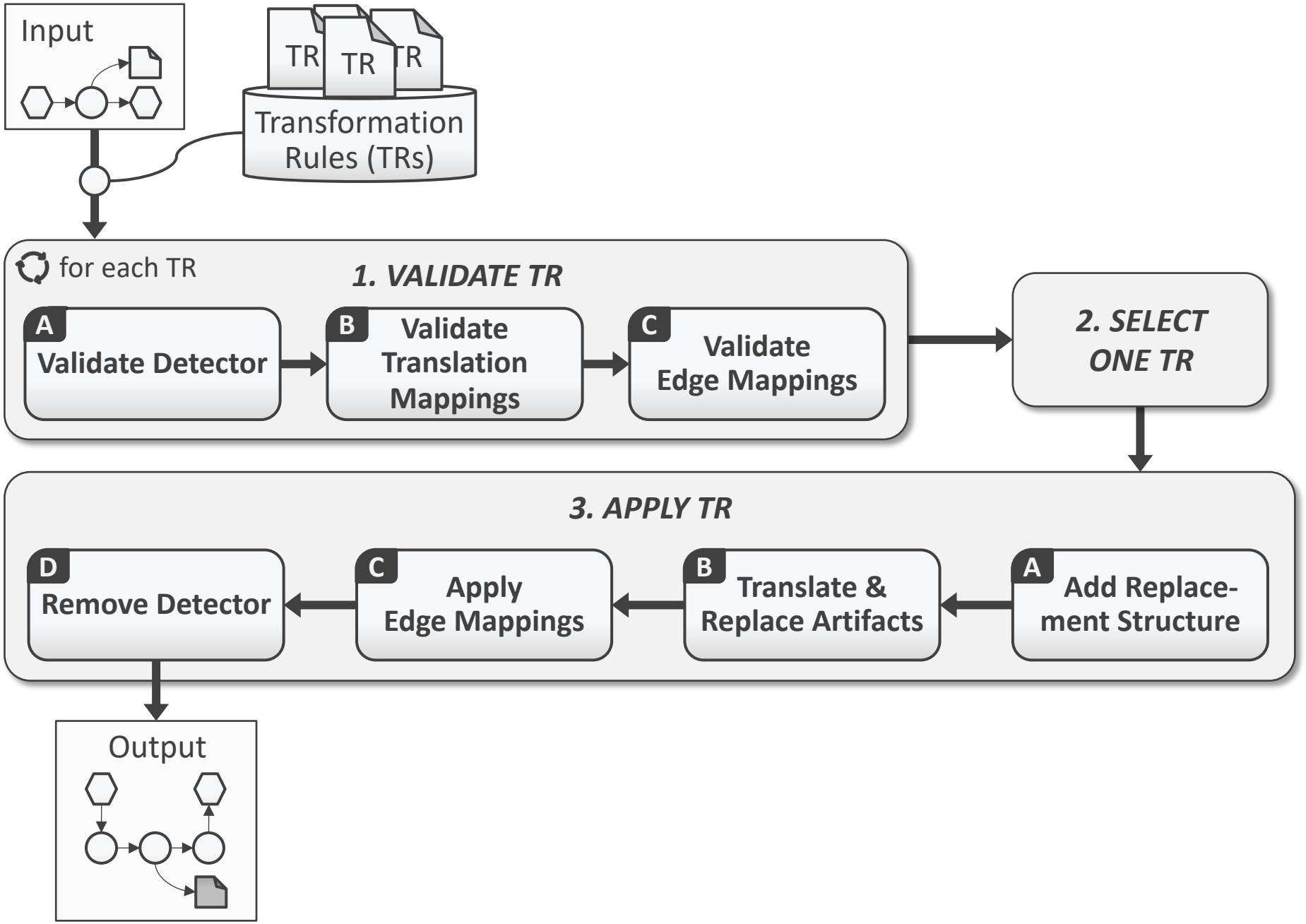
```
from qiskit import QuantumCircuit,
    QuantumRegister,
ClassicalRegister
from qiskit import execute

q = QuantumRegister(3, 'q')
c = ClassicalRegister(3, 'c')
circ = QuantumCircuit(q, c)
circ.
...
```

Translate

```
from pyquil import Program, get_qc
from pyquil.gates import H, CNOT

p = Program()
p += H(0)
...
```

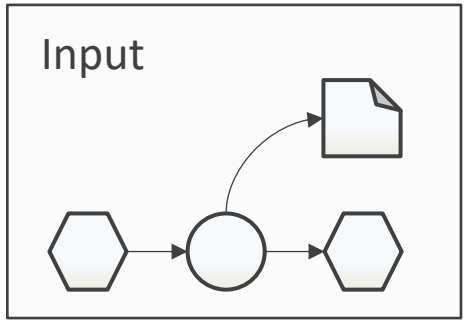
How can existing approaches for translating individual artifacts be integrated into model transformation processes?



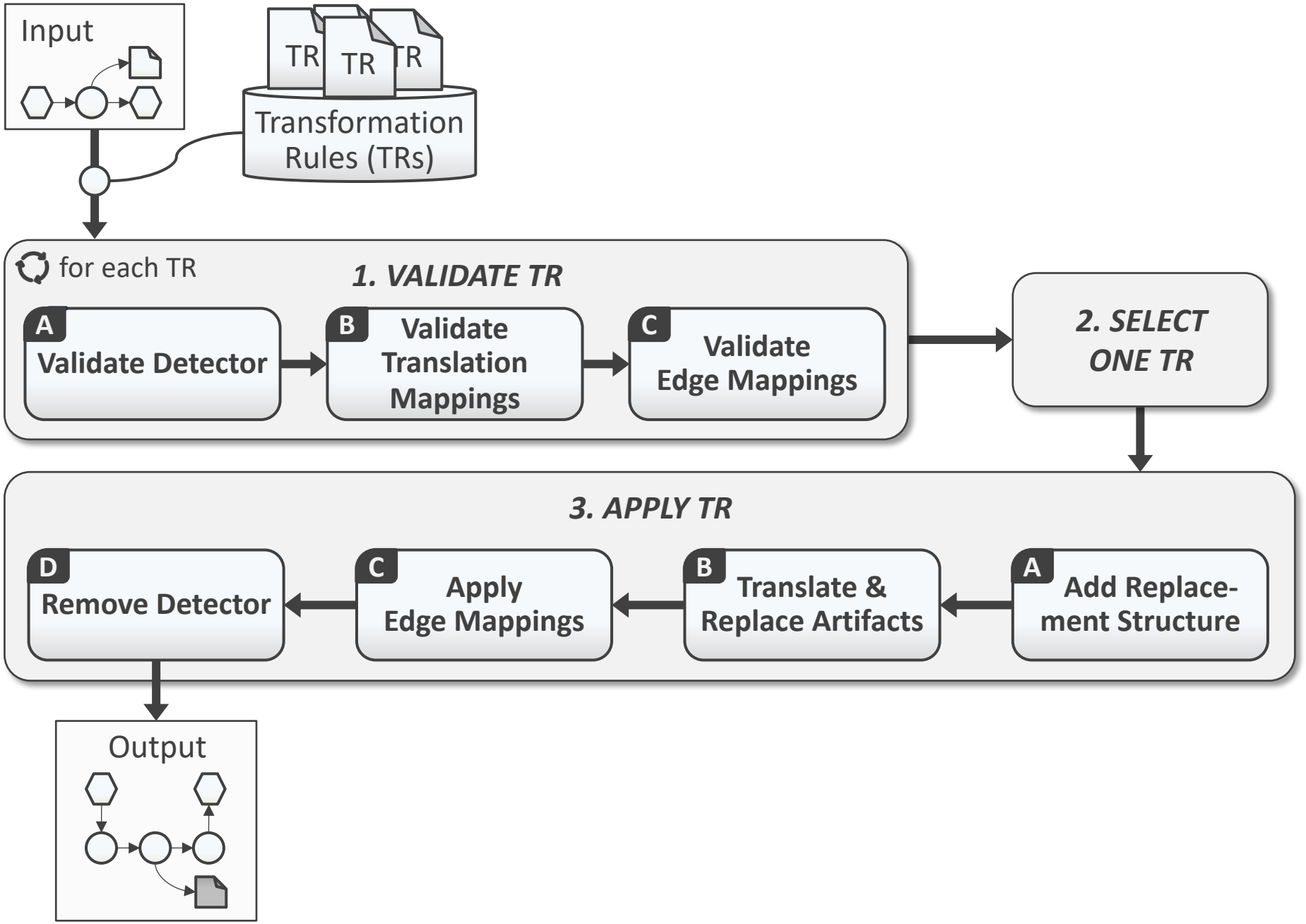

Input
TR
TR
TR
Transformation Rules (TRs)
for each TR
1. VALIDATE TR
A
Validate Detector
B
Validate Translation Mappings
C
Validate Edge Mappings
2. SELECT ONE TR
3. APPLY TR
A
Add Replace-ment Structure
B
Translate & Replace Artifacts
C
Apply Edge Mappings
D
Remove Detector
Output

Metamodel

$$m = (N_m, E_m, NT_m, ET_m, type_m, supertype_m)$$



N_m	nodes
E_m	edges
NT_m	node types
ET_m	edge types
$type_m$	map assigning types
$supertype_m$	map assigning supertypes

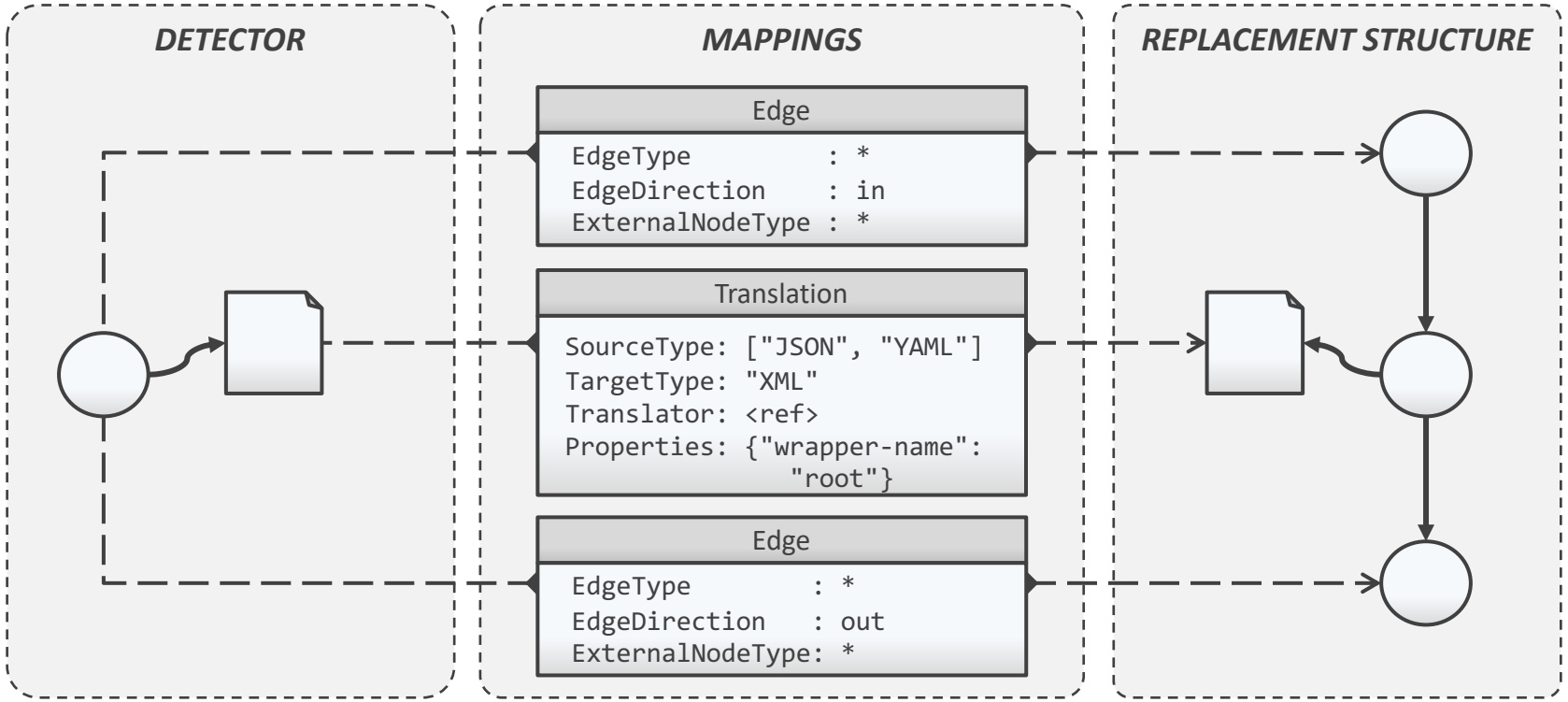

Input
TR
TR
TR
Transformation Rules (TRs)
for each TR
1. VALIDATE TR
A
Validate Detector
B
Validate Translation Mappings
C
Validate Edge Mappings
2. SELECT ONE TR
3. APPLY TR
A
Add Replace-ment Structure
B
Translate & Replace Artifacts
C
Apply Edge Mappings
D
Remove Detector
Output

Transformation Rules

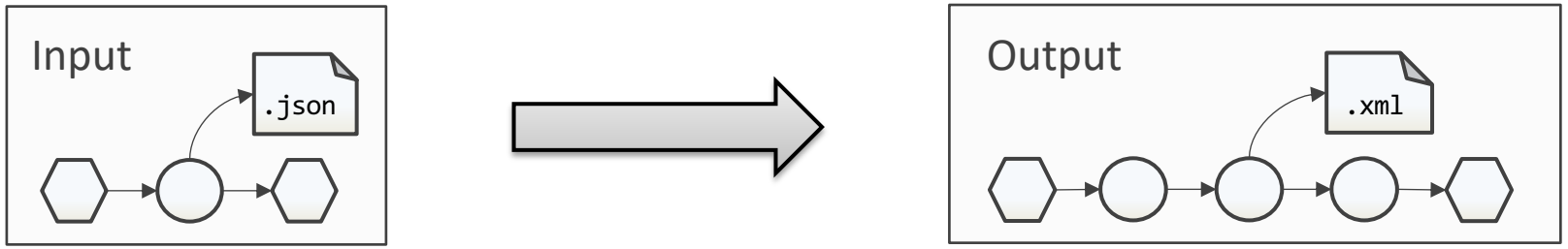
$$tr = (d_{tr}, rs_{tr}, TM_{tr}, EM_{tr})$$

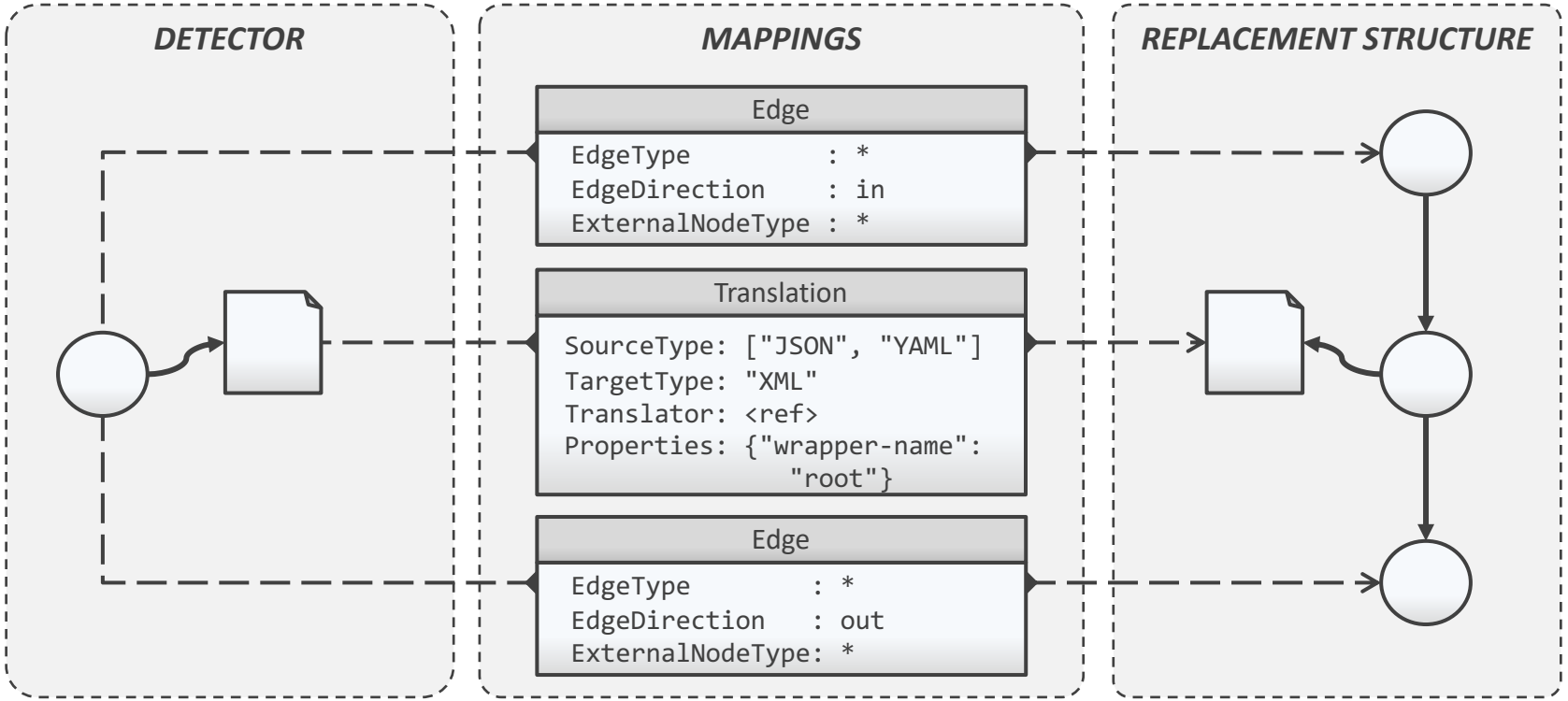
$$tm = (sa_{tm}, ta_{tm}, SAT_{tm}, tat_{tm}, ref_{tm}, P_{tm})$$

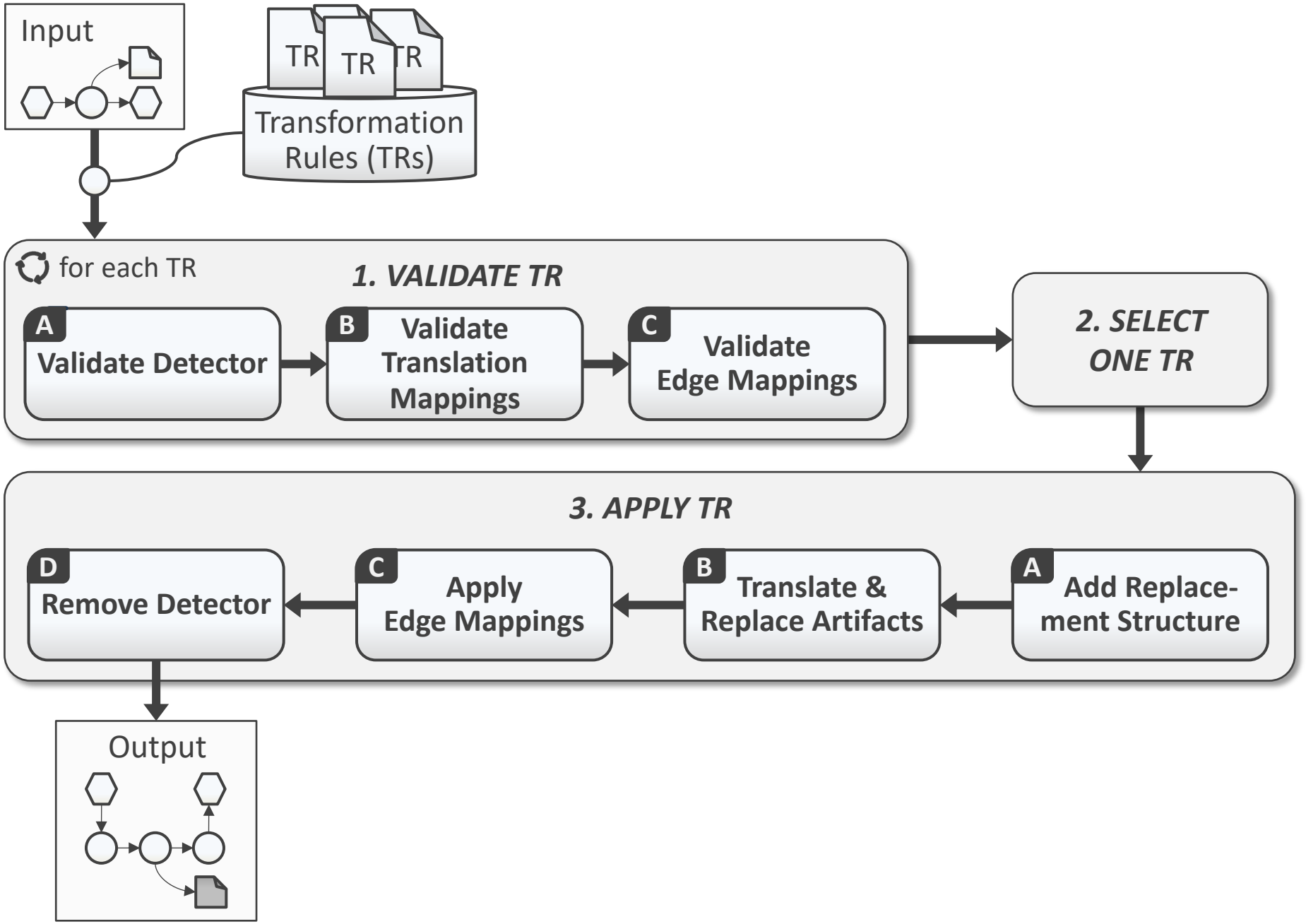
$$em = (sn_{em}, tn_{em}, ET_{em}, direction_{em}, ENT_{em})$$



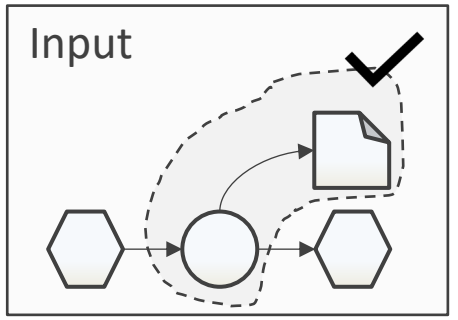
Transformation Rules – Example

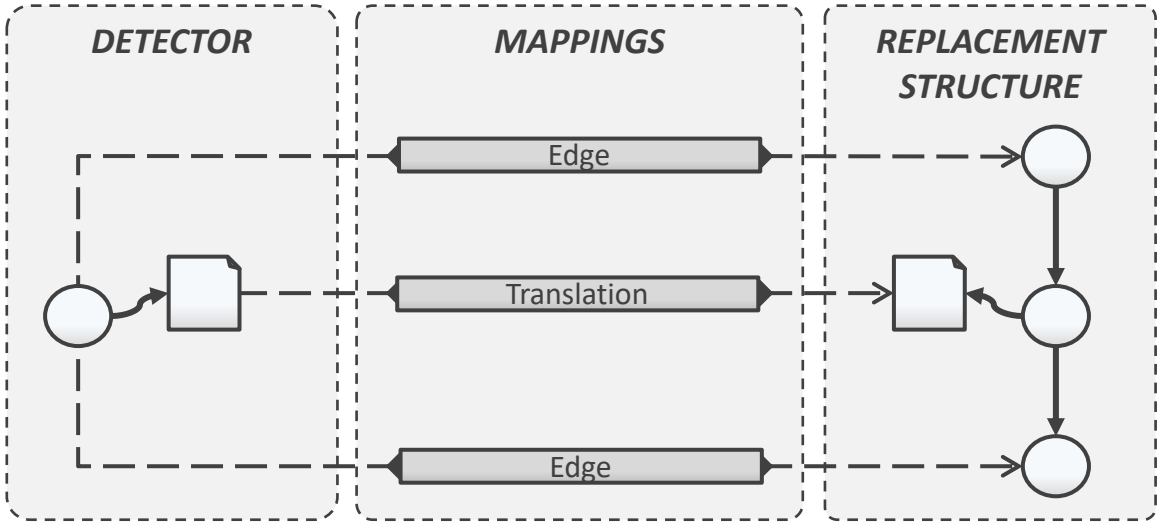


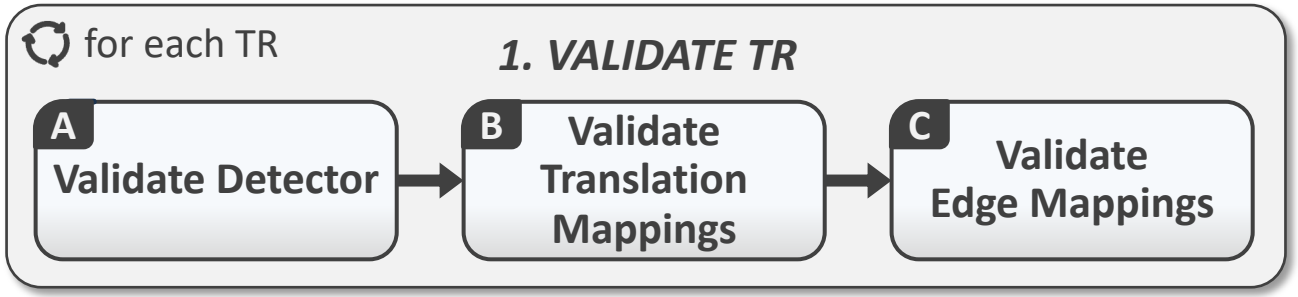



Input
TR
TR
TR
Transformation Rules (TRs)
for each TR
1. VALIDATE TR
A
Validate Detector
B
Validate Translation Mappings
C
Validate Edge Mappings
2. SELECT ONE TR
3. APPLY TR
A
Add Replace-ment Structure
B
Translate & Replace Artifacts
C
Apply Edge Mappings
D
Remove Detector
Output

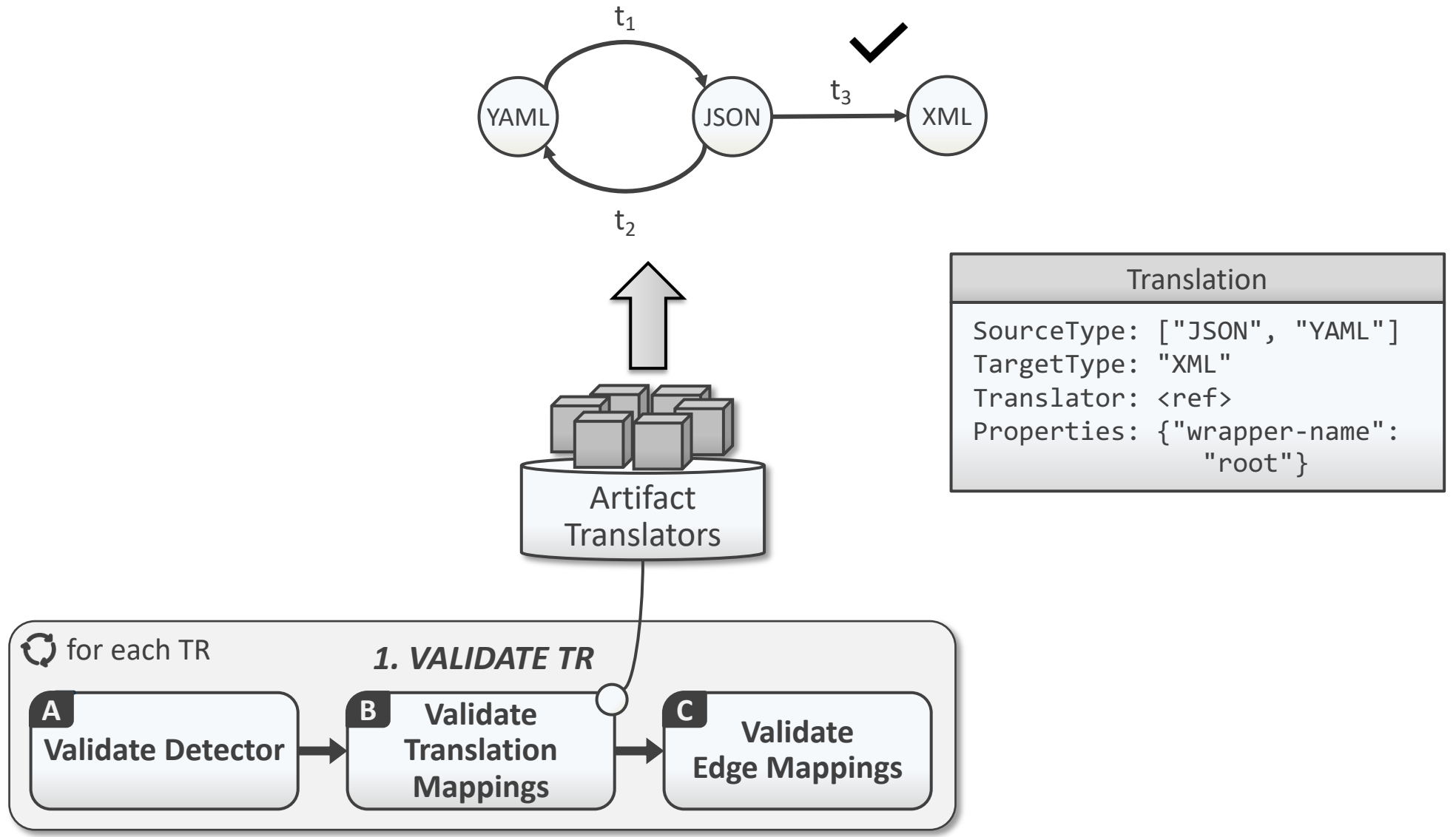
Transformation Rule Validation – Detector







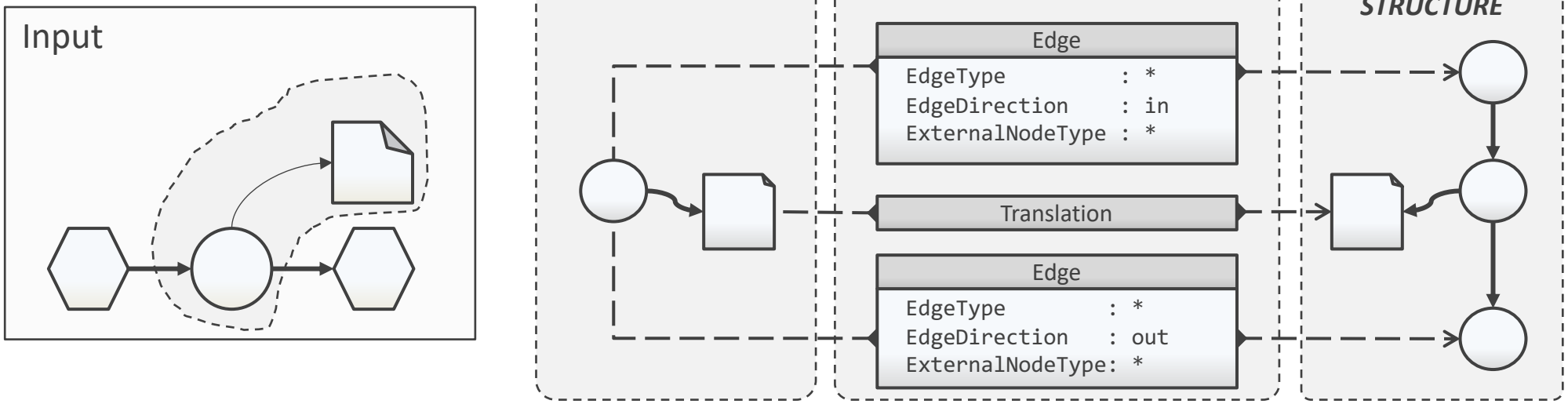
Transformation Rule Validation – Translation Mappings

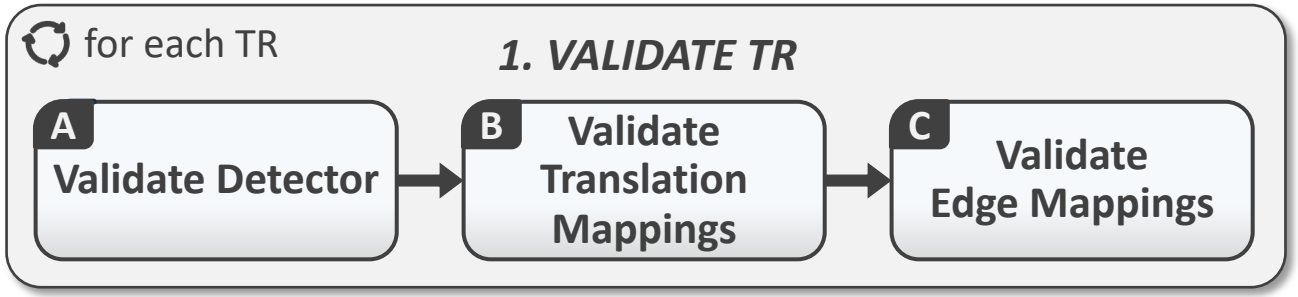


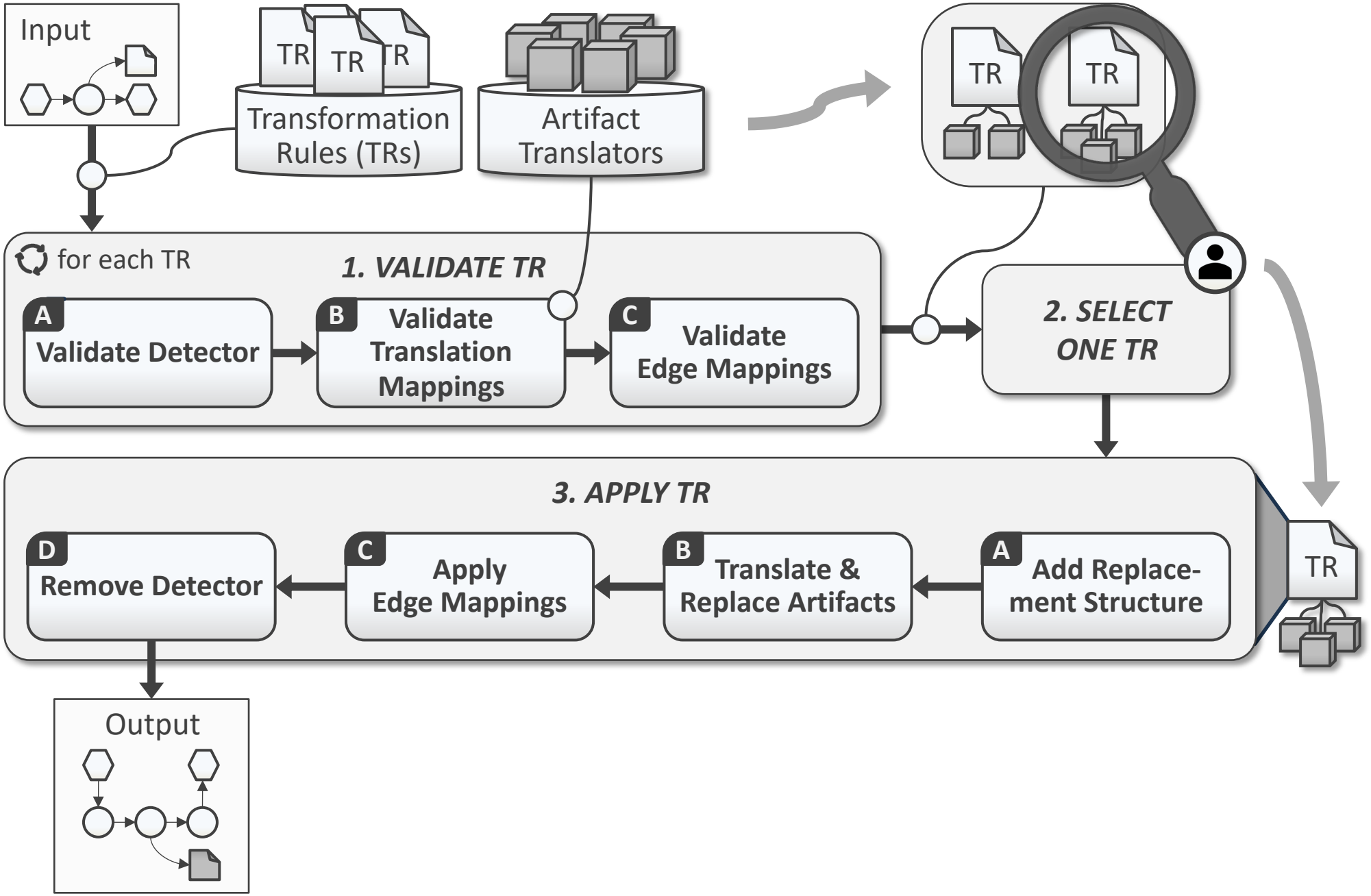
Translation

```
SourceType: ["JSON", "YAML"]
TargetType: "XML"
Translator: <ref>
Properties: {"wrapper-name":
              "root"}
```

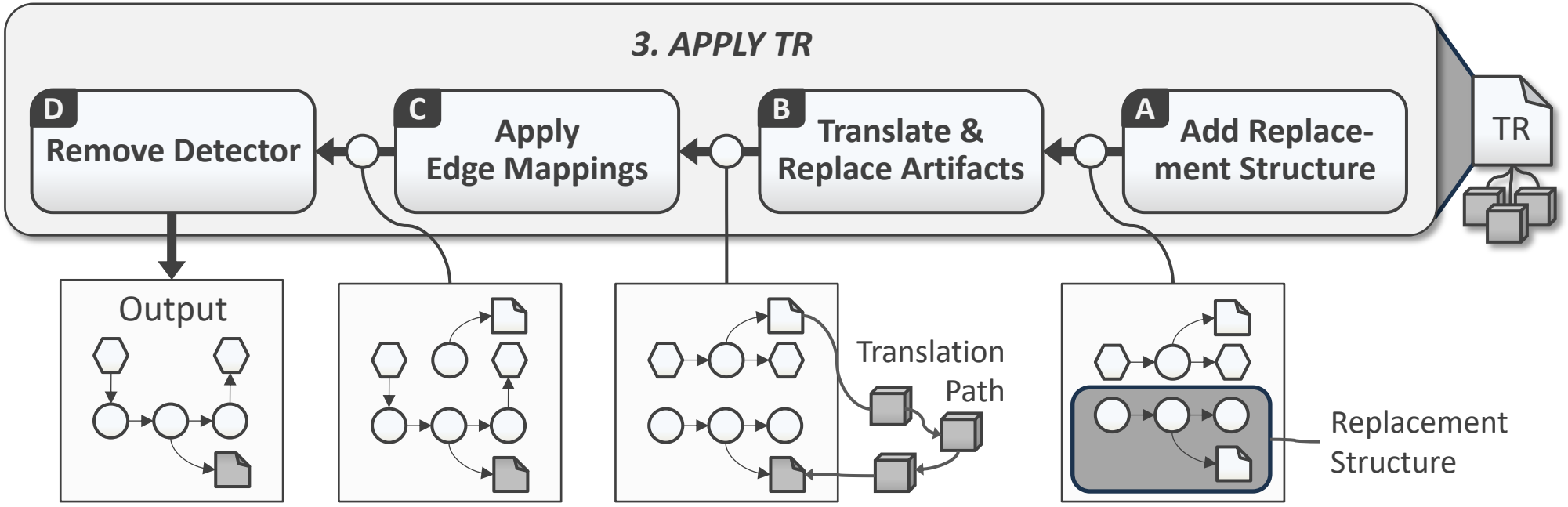
Transformation Rule Validation – Edge Mappings

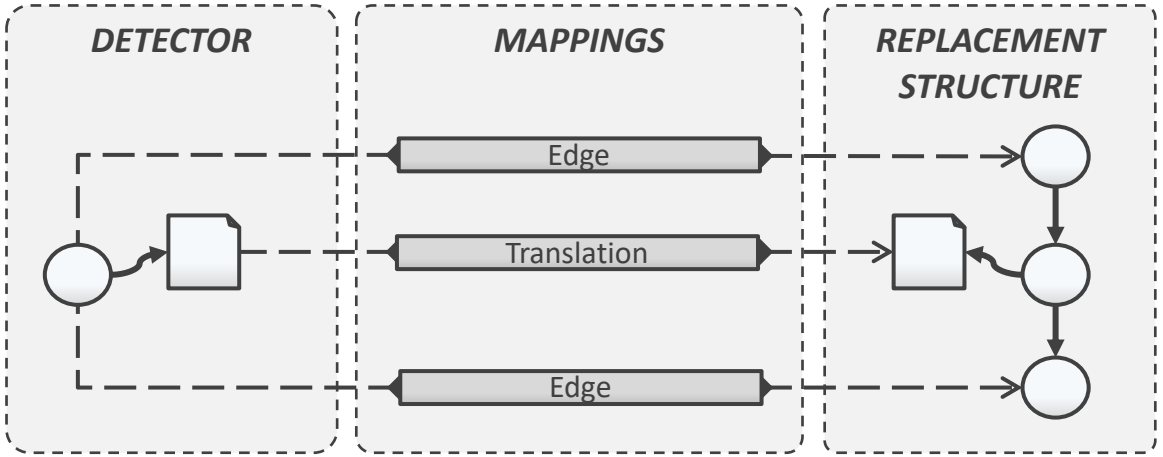





Input
TR
TR
TR
Transformation Rules (TRs)
Artifact Translators
TR
TR
for each TR
1. VALIDATE TR
A
Validate Detector
B
Validate Translation Mappings
C
Validate Edge Mappings
2. SELECT ONE TR
3. APPLY TR
D
Remove Detector
C
Apply Edge Mappings
B
Translate & Replace Artifacts
A
Add Replace-ment Structure
TR
Output

Transformation Rule Application

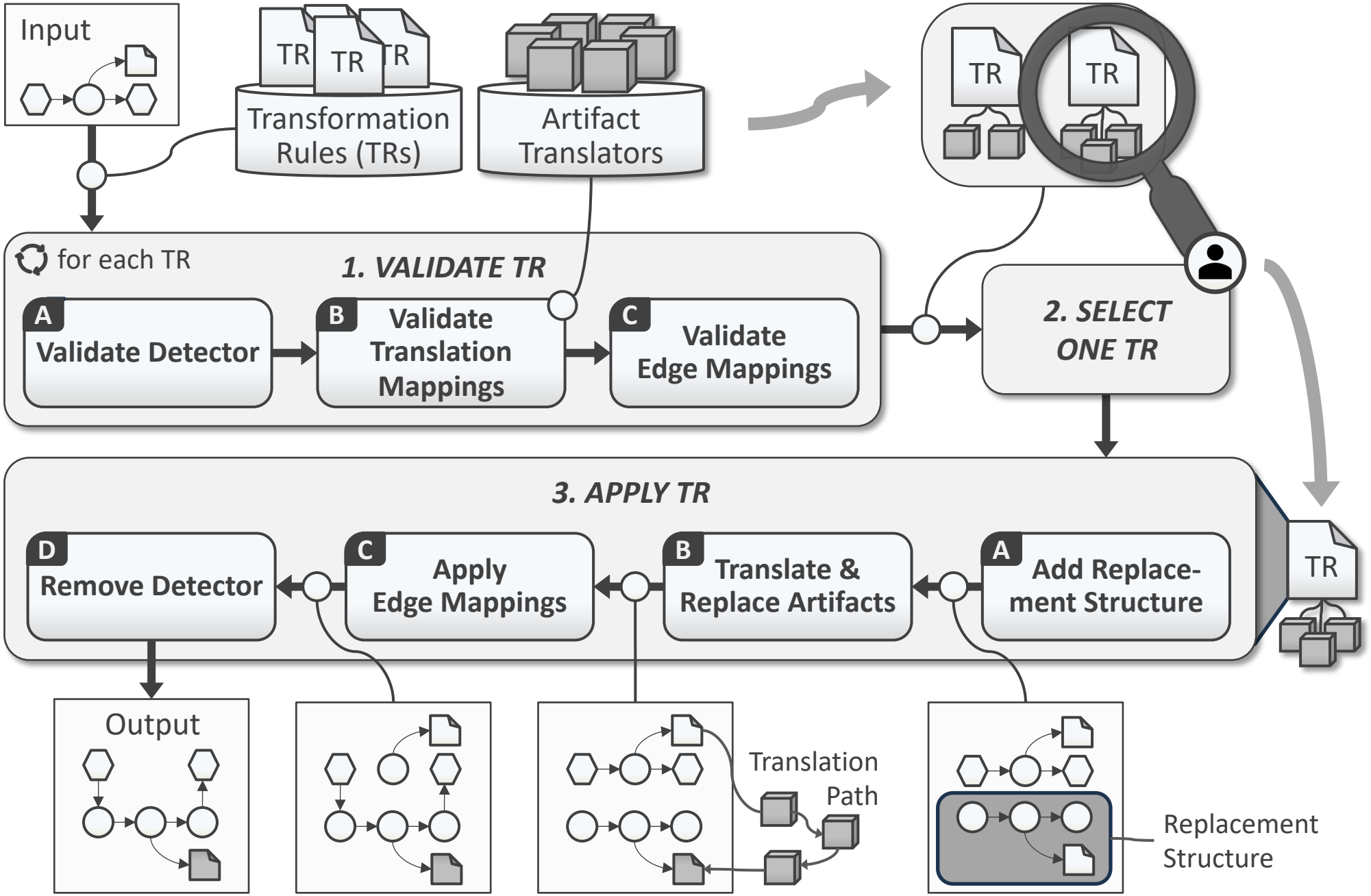




Transformation Rule Application

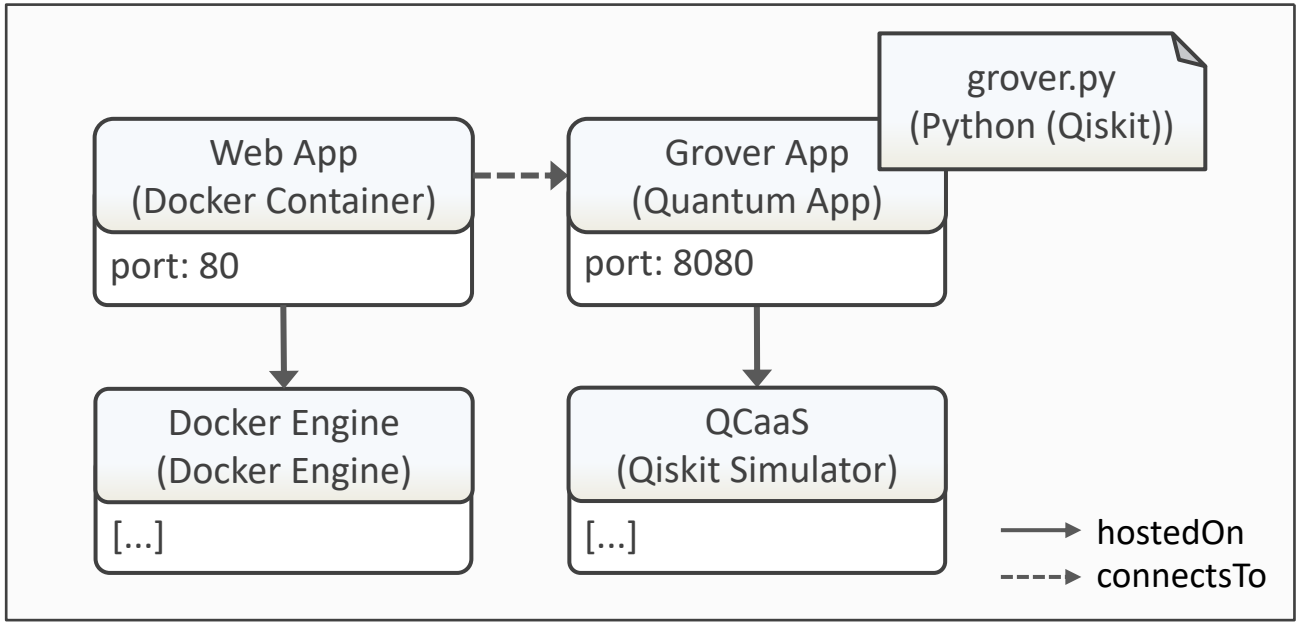
Require: input model $m \in M$, transformation rule $tr \in TR$, set of translators T, set of external edges and corresponding edge mappings EXT

1: $ME_m := ME_m \cup N_{\pi_2(tr)}$
2: **for all** $(e_i, em_i) \in EXT$ **do**
3: **if** $direction(e_i) = incoming$ **then** $\pi_1(e_i) = \pi_2(em_i)$
4: **else** $\pi_2(e_i) = \pi_2(em_i)$ **end if**
5: **end for**
6: **for all** $tm_i \in \pi_3(tr)$ **do**
7: $tp := \text{TRANSLATIONPATH}(\pi_3(tm_i), \pi_4(tm_i))$ ▷ any pathfinding algorithm
8: $a := \pi_1(tm_i)$
9: **for all** $tp_i \in tp$ **do**
10: $a := \text{TRANSLATE}(tp_i, a, \pi_6(tm_i))$
11: **end for**
12: $ME_m := ME_m \cup \{a\}$
13: **for all** $e_j \in \pi_2(m)$ **do**
14: **if** $\pi_2(e_j) = \pi_2(tm_i)$ **then** $\pi_2(e_j) := a$ **else** $\pi_1(e_j) := a$ **end if**
15: **end for**
16: $ME_m := ME_m \setminus \{\pi_2(tm_i)\}$
17: **end for**
18: $ME_m := ME_m \setminus \{me_i \in ME_m : \exists (me_i, me_j) \in sm\}$

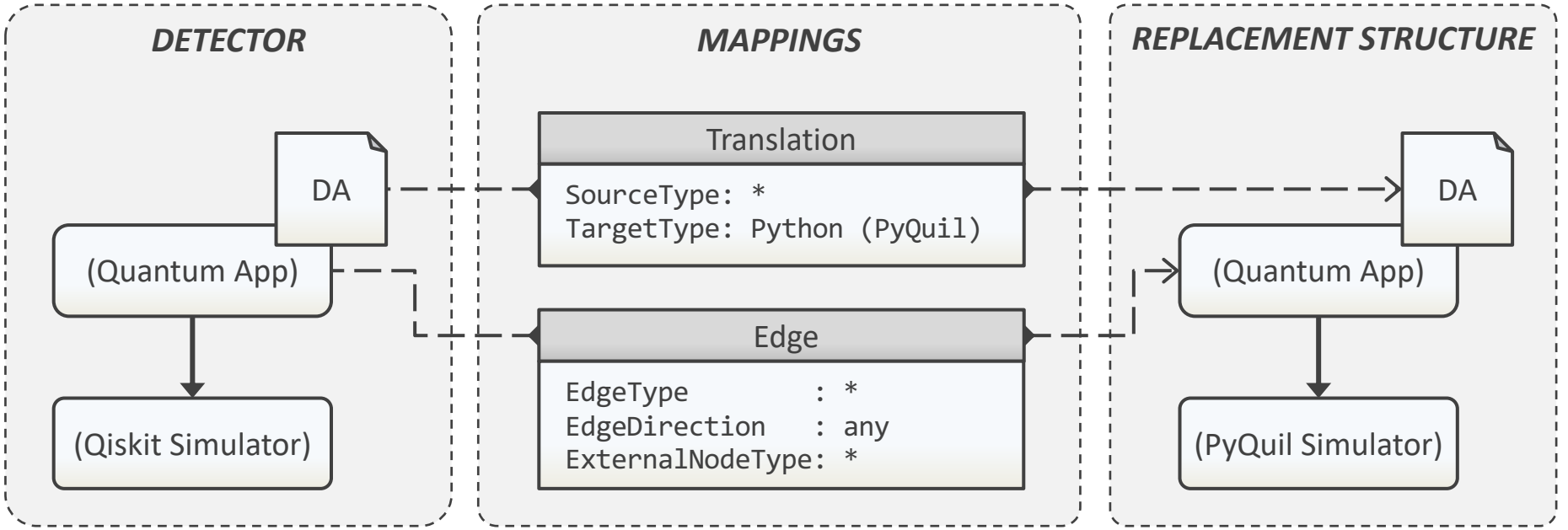

Input
TR
TR
TR
Transformation Rules (TRs)
Artifact Translators
TR
TR
for each TR
1. VALIDATE TR
A
Validate Detector
B
Validate Translation Mappings
C
Validate Edge Mappings
2. SELECT ONE TR
3. APPLY TR
D
Remove Detector
C
Apply Edge Mappings
B
Translate & Replace Artifacts
A
Add Replace-ment Structure
TR
Output
Translation Path
Replacement Structure

Exemplary Use Cases

Exemplary Deployment Model

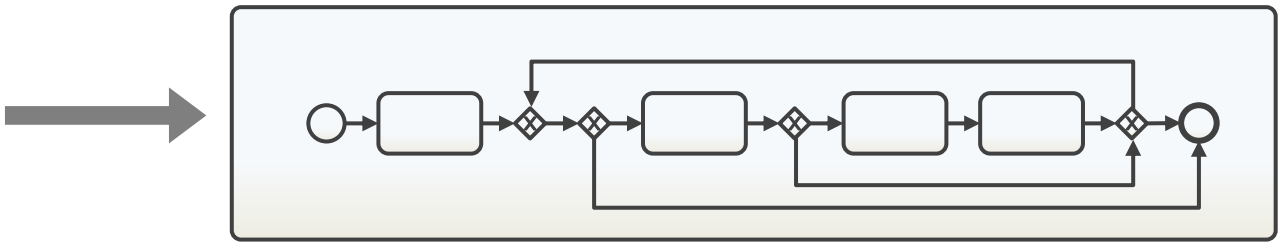


Deployment Model Transformation Rule Example



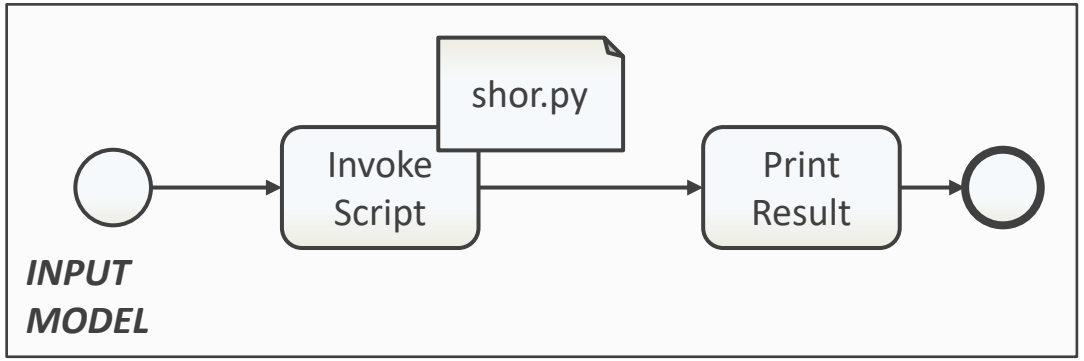
Script Splitter Use Case

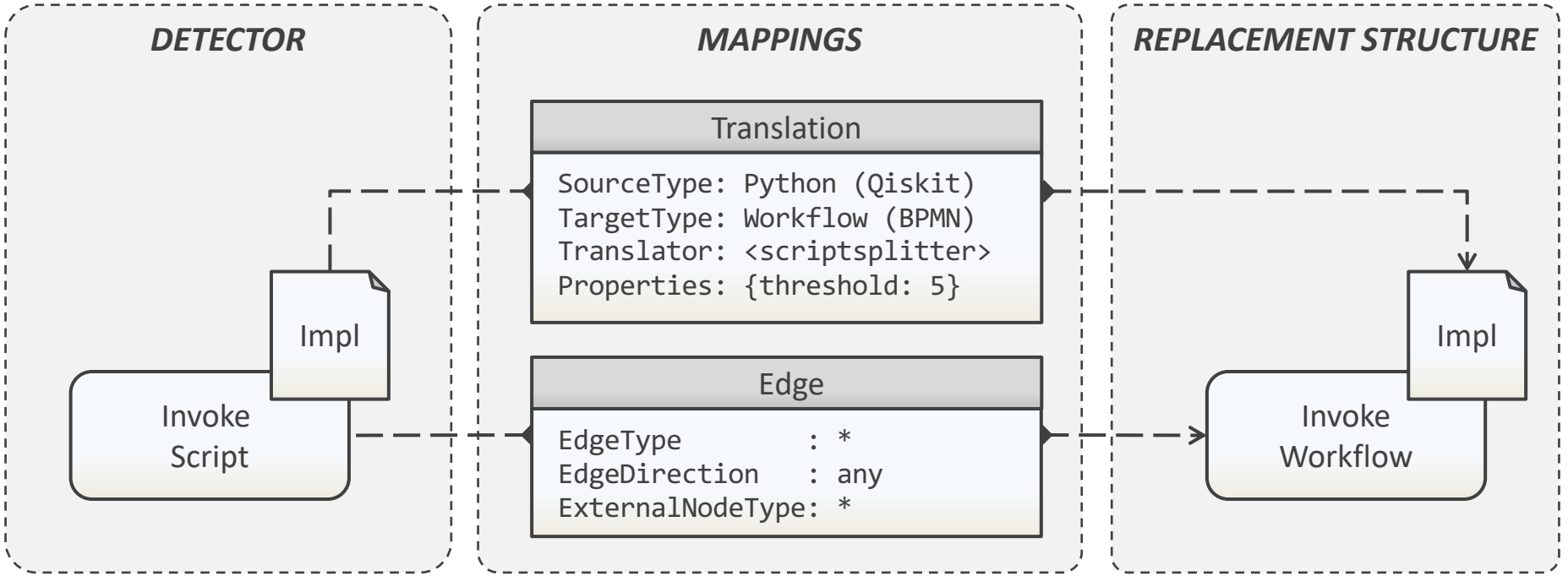
```
from qiskit import QuantumCircuit,
     QuantumRegister,
ClassicalRegister
from qiskit import execute

q = QuantumRegister(3, ‘q’)
c = ClassicalRegister(3, ‘c’)
circ = QuantumCircuit(q, c)
circ.h(q)
...
```



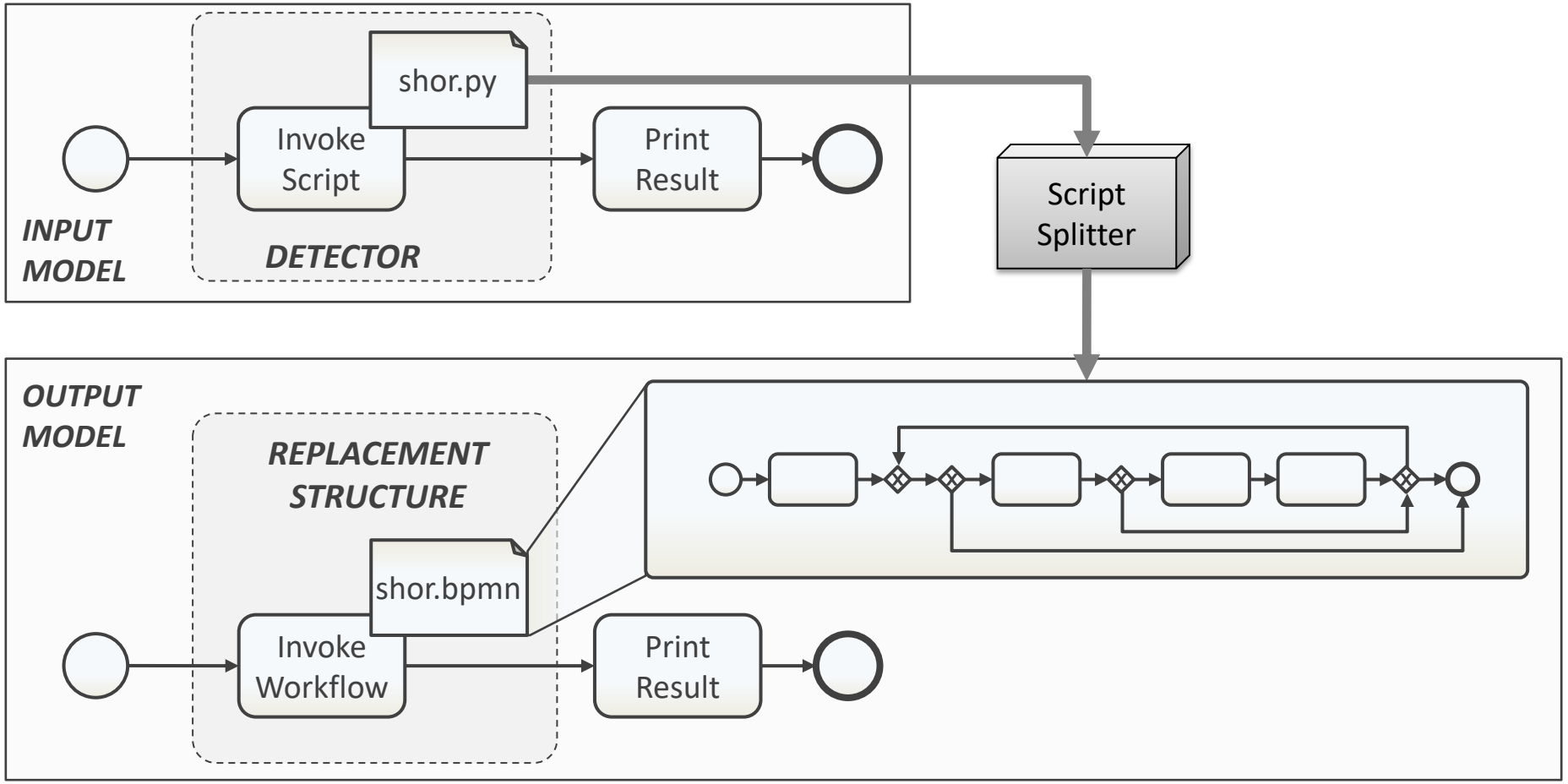
Vietz, Daniel; Barzen, Johanna; Leymann, Frank; Weder, Benjamin: **Splitting Quantum-Classical Scripts for the Generation of Quantum Workflows.** In: Proceedings of the 26th Conference on Enterprise Design, Operations, and Computing (EDOC 2022), Springer International Publishing, 2022

BPM Transformation Rule Example





BPM Input & Output Model



Discussion

Discussion

- Approach works on different abstraction levels
- Translators act as black boxes
  - Recursive model transformations
  - Hide confidential parts
- Sequence of transformation rules is not commutative
- Translators must exist

Summary and Future Work

- Integration of artifact translators into model transformation
- Validate & apply transformation rules
- Practical use cases

Future Work

- Automatically migrate running instances of the transformed models

Thank you ☺