Analysis of ATL Transformations

Matthieu Allon

Valerio Cosentino

Outline

- Problem Description
- Goal
- Possible Approaches
- Graph representation and uses cases
- Feedbacks

Problem Description

- ATL transformations can be difficult to understand
  - Rules
    - How they are connected each other
  - Helpers
    - Unused helpers
    - What they do

- The ATL debug does not help a lot

Goal

- Facilitate the understanding of ATL transformations
  - Graph representations
    - Portolan
    - SVG
  - Offer a new solution to debug ATL transformations

Possible Approaches

- Static Analysis
  - ATL transformation files
- Dynamic Analysis
  - ATL engine

Possible Approaches

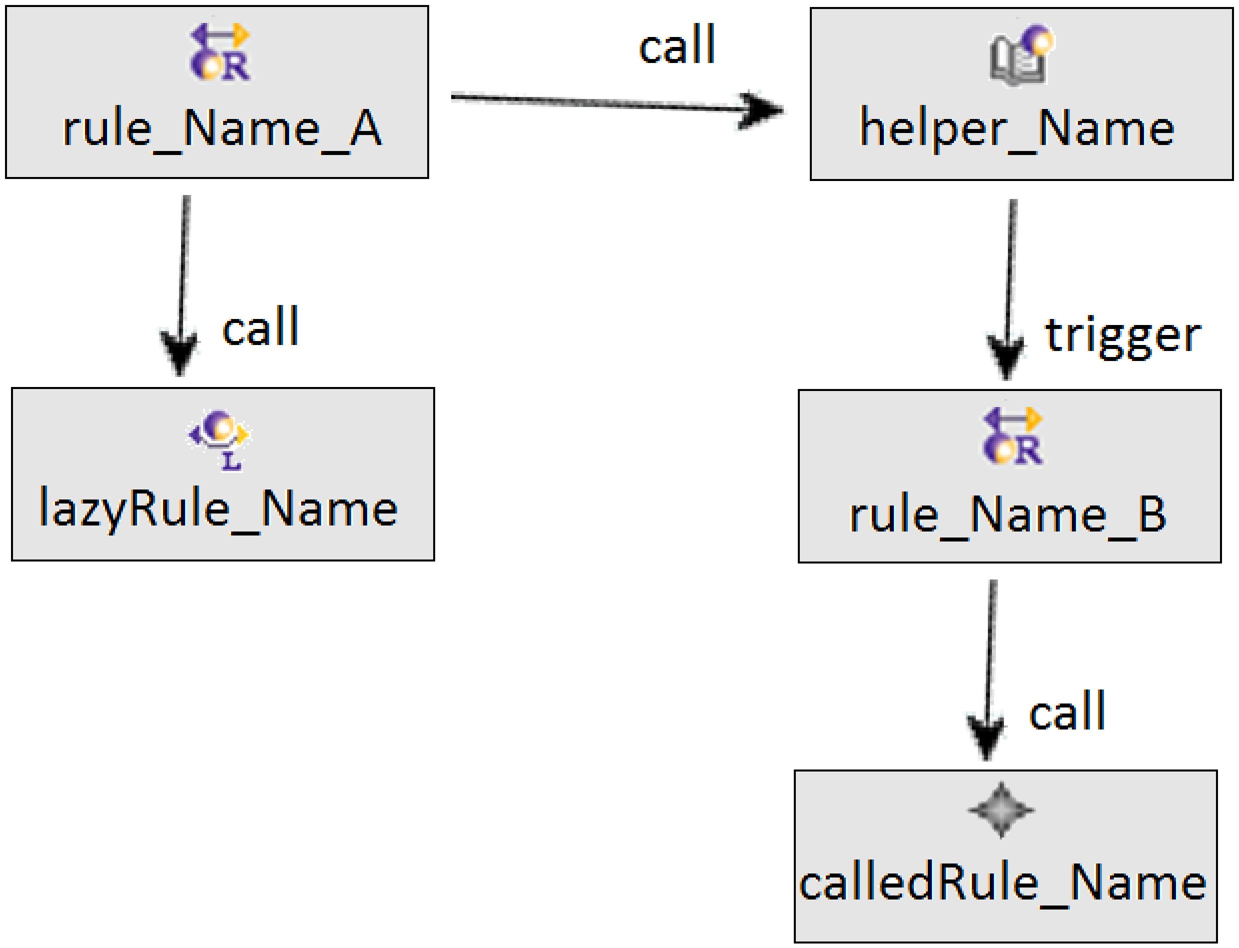
- Static analysis
  - Advantages:
    - Input model independency
  - Drawbacks:
    - Input metamodel analysis
    - Recursion identification
    - Types in ATL transformation (OclAny) ?
    - ATL metamodel dependency ?

Possible Approaches

- Dynamic analysis
  - Advantages:
    - Recursion identification
    - ATL engine stack information
  - Drawbacks:
    - Input model dependency
    - Different ATL engines (EMFTVM)
    - Increasing of memory use/computational time?

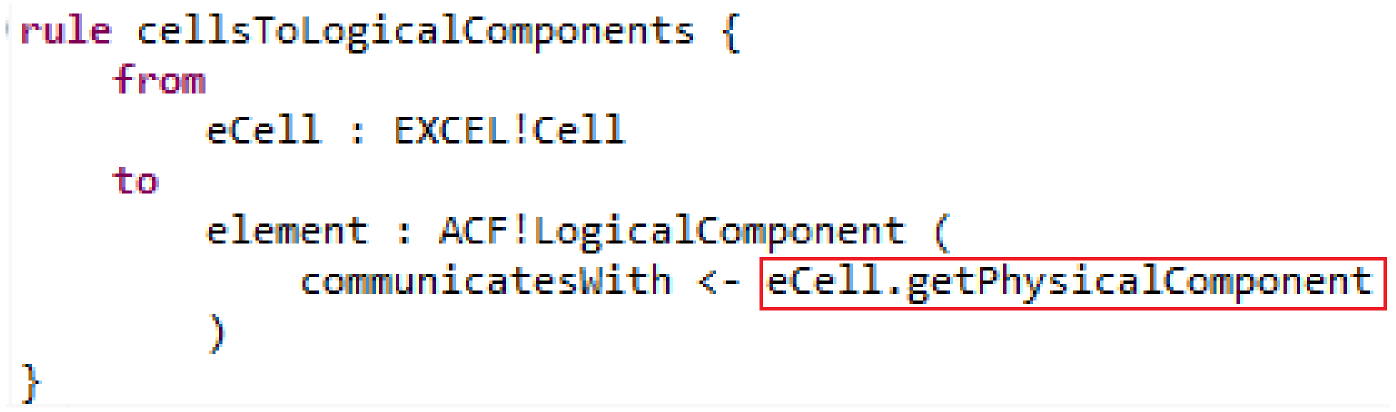
Graph representation

- Elements representation :

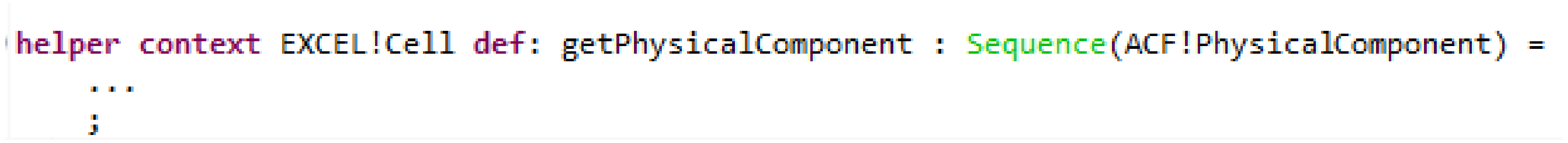


Graph use cases

- Trigger of rules :
  - One rule calling a helper :



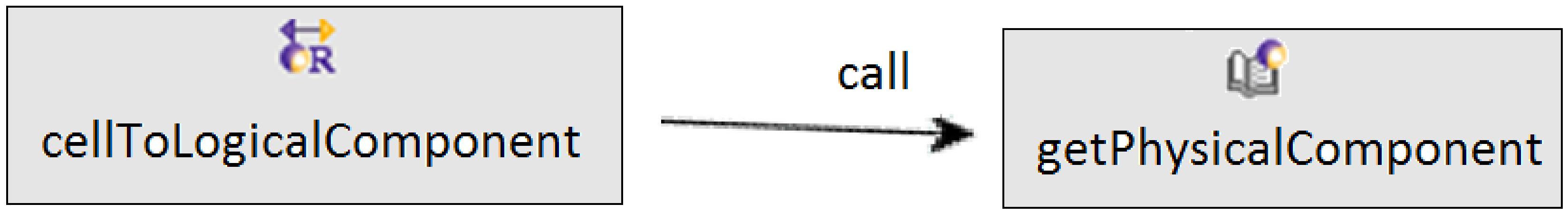
```
rule cellsToLogicalComponents {
    from
        eCell : EXCEL!Cell
    to
        element : ACF!LogicalComponent (
            communicatesWith <- eCell.getPhysicalComponent
        )
}
```



```
helper context EXCEL!Cell def: getPhysicalComponent : Sequence(ACF!PhysicalComponent) =
    ...
    ;
```

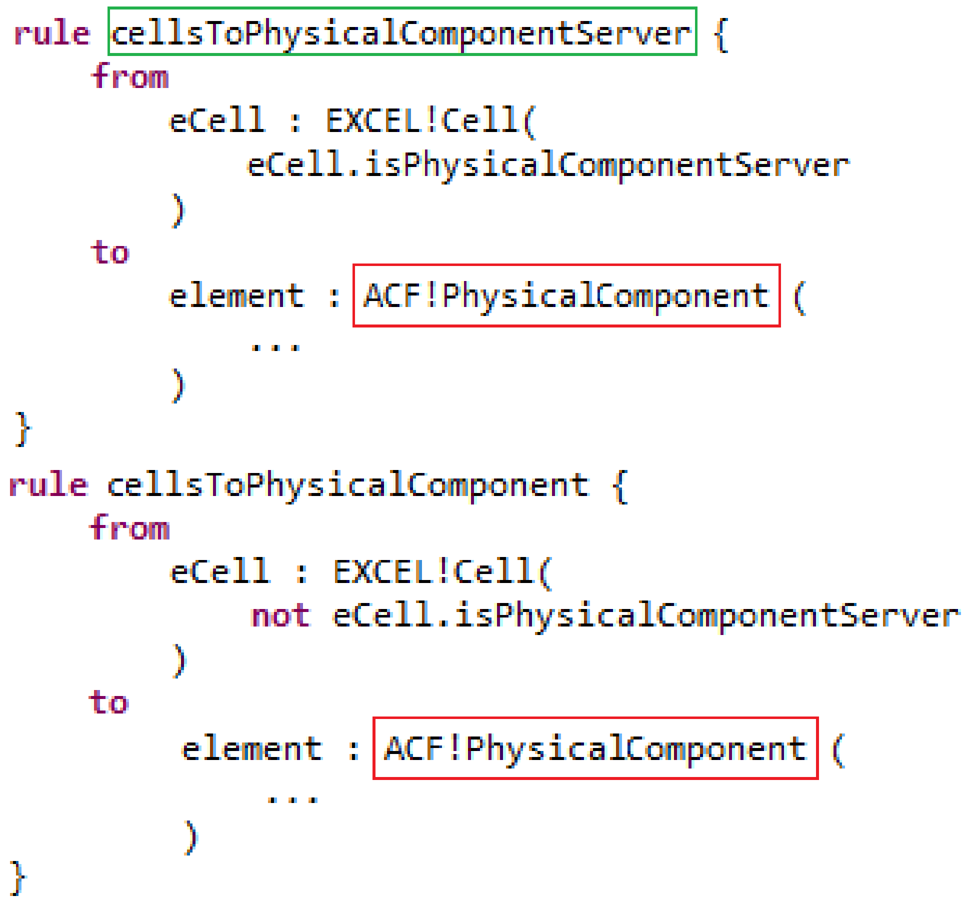
Graph use cases

- Trigger of rules :
  - One rule calling a helper :



Graph use cases

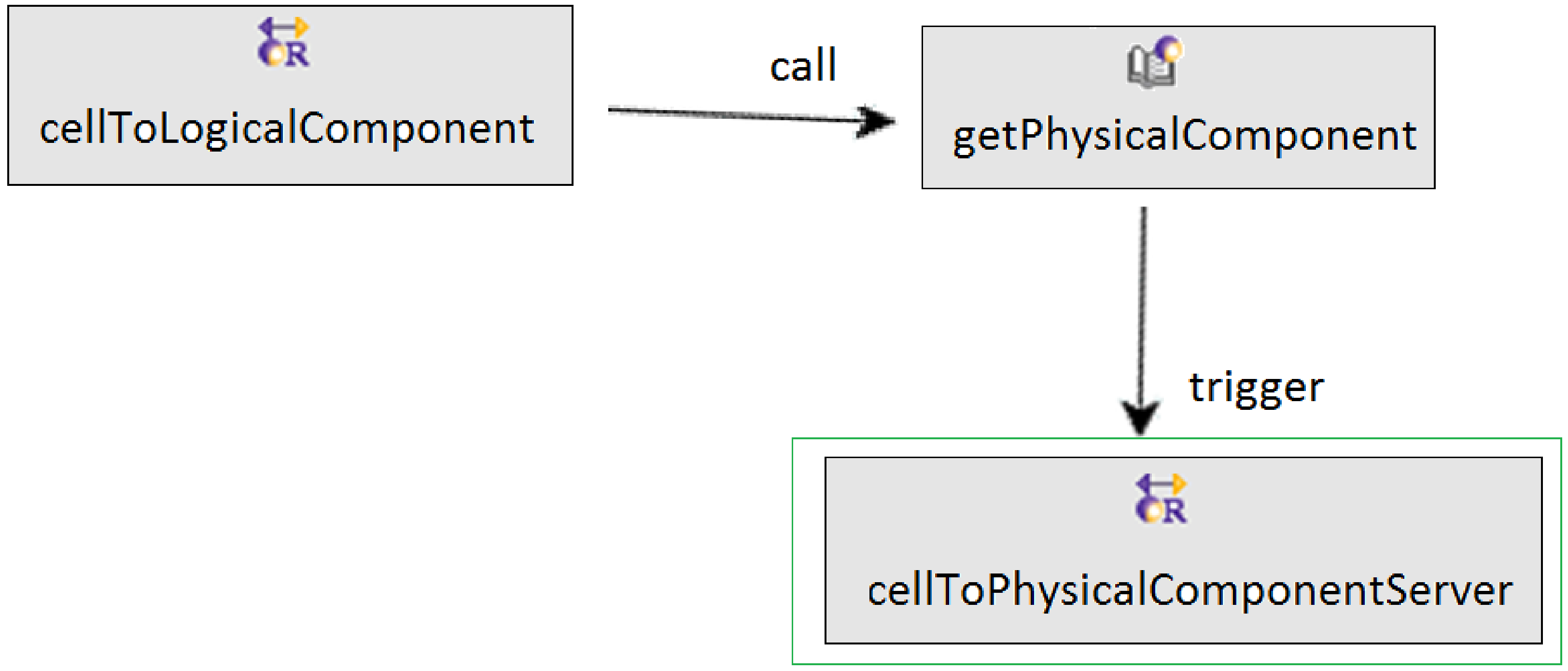
- Trigger of rules :
  - And two others which generate the same element :



```
rule cellsToPhysicalComponentServer {
    from
        eCell : EXCEL!Cell(
            eCell.isPhysicalComponentServer
        )
    to
        element : ACF!PhysicalComponent (
            ...
        )
}
rule cellsToPhysicalComponent {
    from
        eCell : EXCEL!Cell(
            not eCell.isPhysicalComponentServer
        )
    to
         element : ACF!PhysicalComponent (
             ...
         )
}
```

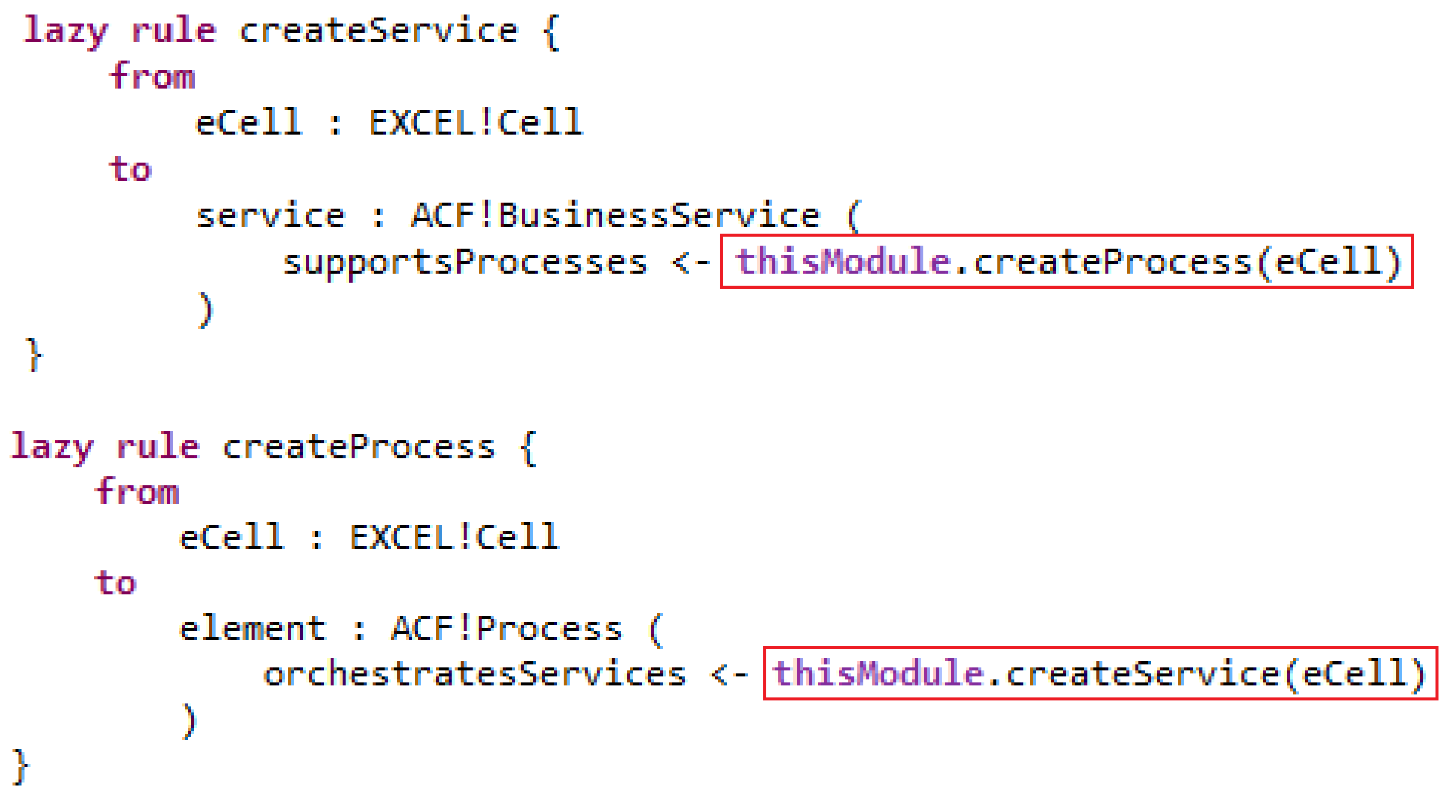
Graph use cases

- Trigger of rules :
  - The graph allow to know which rule is triggered



Graph use cases

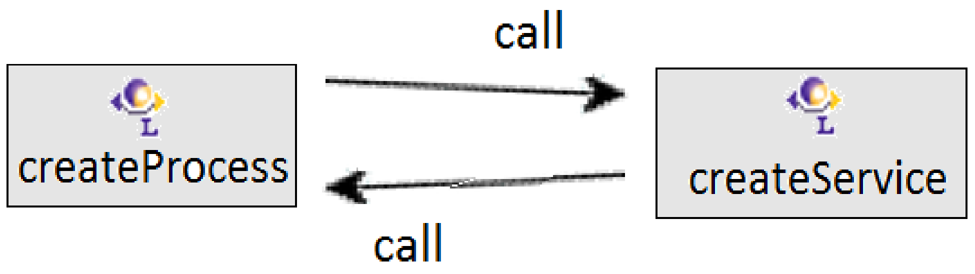
- Recursion with rules / helpers :
  - Recursion with two lazy rules



```
lazy rule createService {
    from
        eCell : EXCEL!Cell
    to
        service : ACF!BusinessService (
            supportsProcesses <- thisModule.createProcess(eCell)
        )
}

lazy rule createProcess {
    from
        eCell : EXCEL!Cell
    to
        element : ACF!Process (
            orchestratesServices <- thisModule.createService(eCell)
        )
}
```

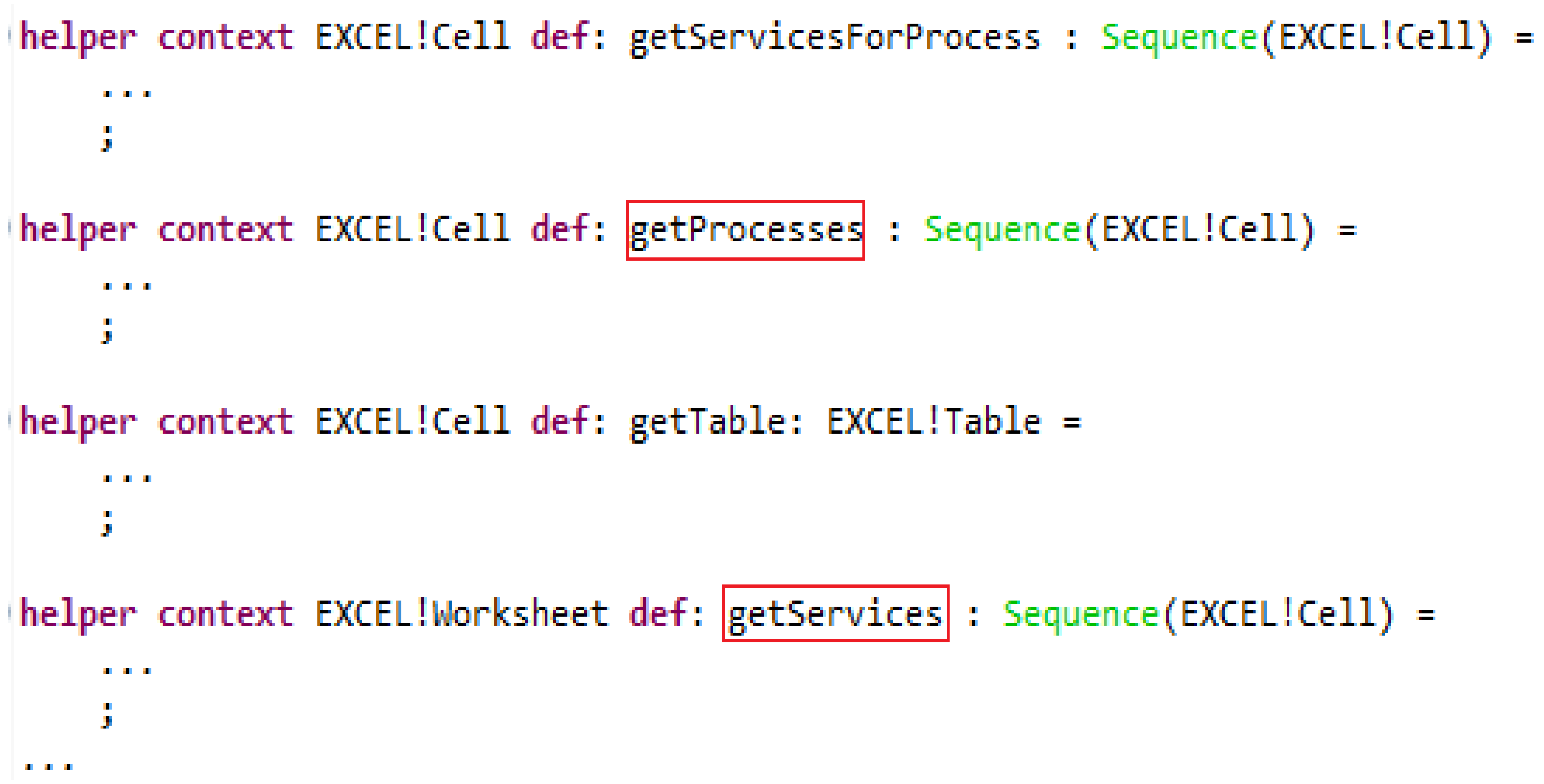
Graph use cases

- Recursion with rules / helpers :
  - The graph allows to know where is the recursion



Graph use cases

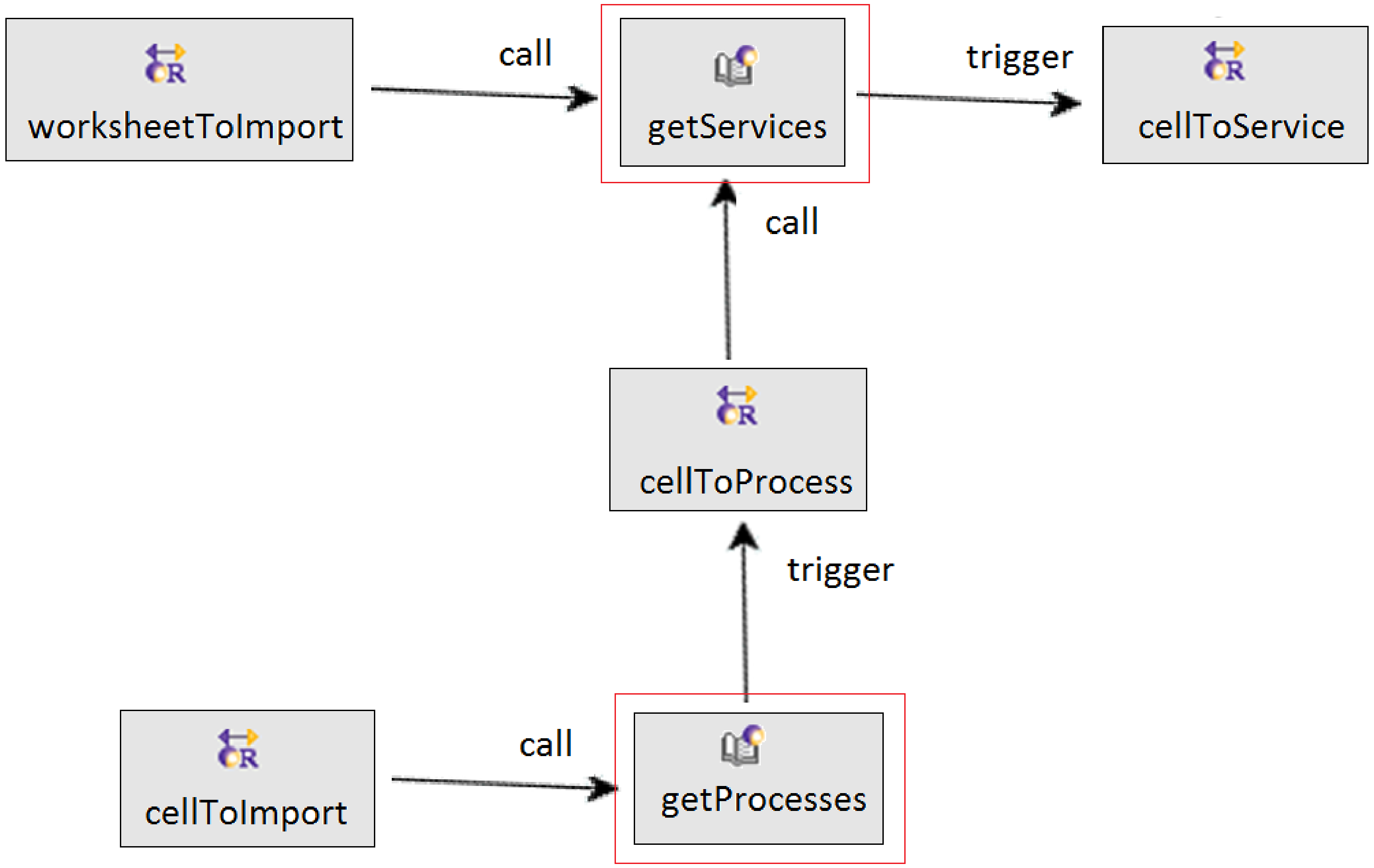
- Helpers / rules used :
  - How to quicly know which helpers are used ?



```
helper context EXCEL!Cell def: getServicesForProcess : Sequence(EXCEL!Cell) =
    ...
    ;

helper context EXCEL!Cell def: getProcesses : Sequence(EXCEL!Cell) =
    ...
    ;

helper context EXCEL!Cell def: getTable: EXCEL!Table =
    ...
    ;

helper context EXCEL!Worksheet def: getServices : Sequence(EXCEL!Cell) =
    ...
    ;
...
```

Graph use cases

- Helpers / rules used :
  - Graph allows to know which helpers are used



Feedbacks