Model-Based Software Engineering

Lecture 10 – Transformation

Prof. Dr. Joel Greenyer

SOFTWARE ENGINEERING

June 28, 2016

Leibniz Universität Hannover

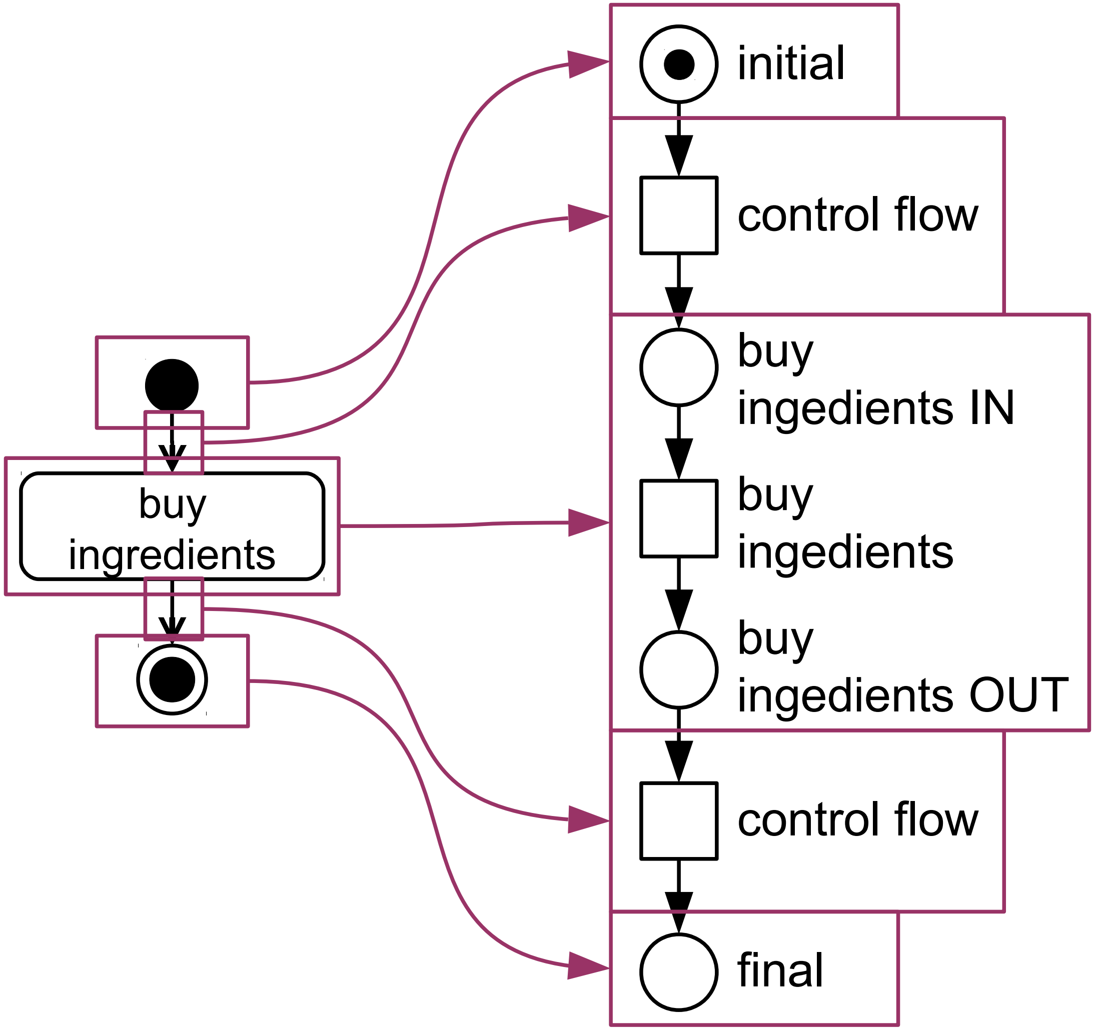
in the last lecture...

5.4. Model-to-model transformation – Triple Graph Grammars

Example:Transform Activity Diagrams to Petri nets

in the last lecture...

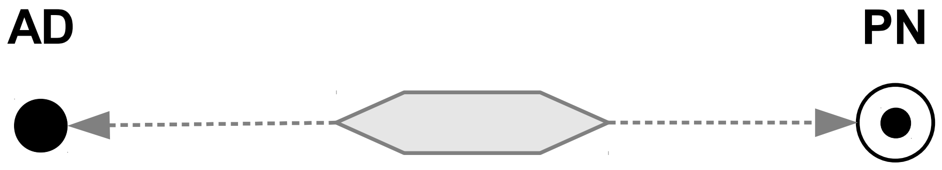
- Let's start simple: How to transform
  - Initial nodes?
  - Final nodes?
  - Action nodes?
  - Control flow edges?



Relations Between Model Patterns Example: Activity to Petri net

in the last lecture...

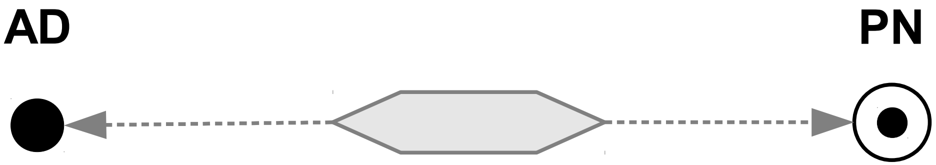
- Intial node ↔ Place with intial marking



Relations Between Model Patterns Example: Activity to Petri net

in the last lecture...

- Intial node ↔ Place with intial marking

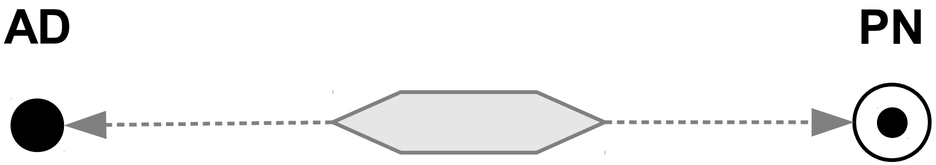


- Final node ↔ Empty Place

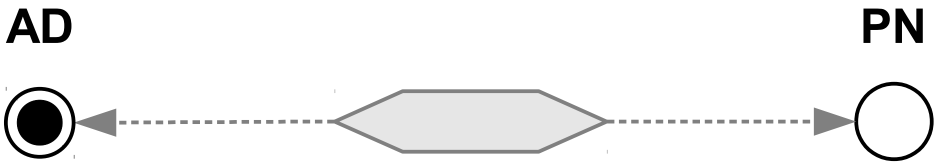
Relations Between Model Patterns Example: Activity to Petri net

in the last lecture...

- Intial node ↔ Place with intial marking



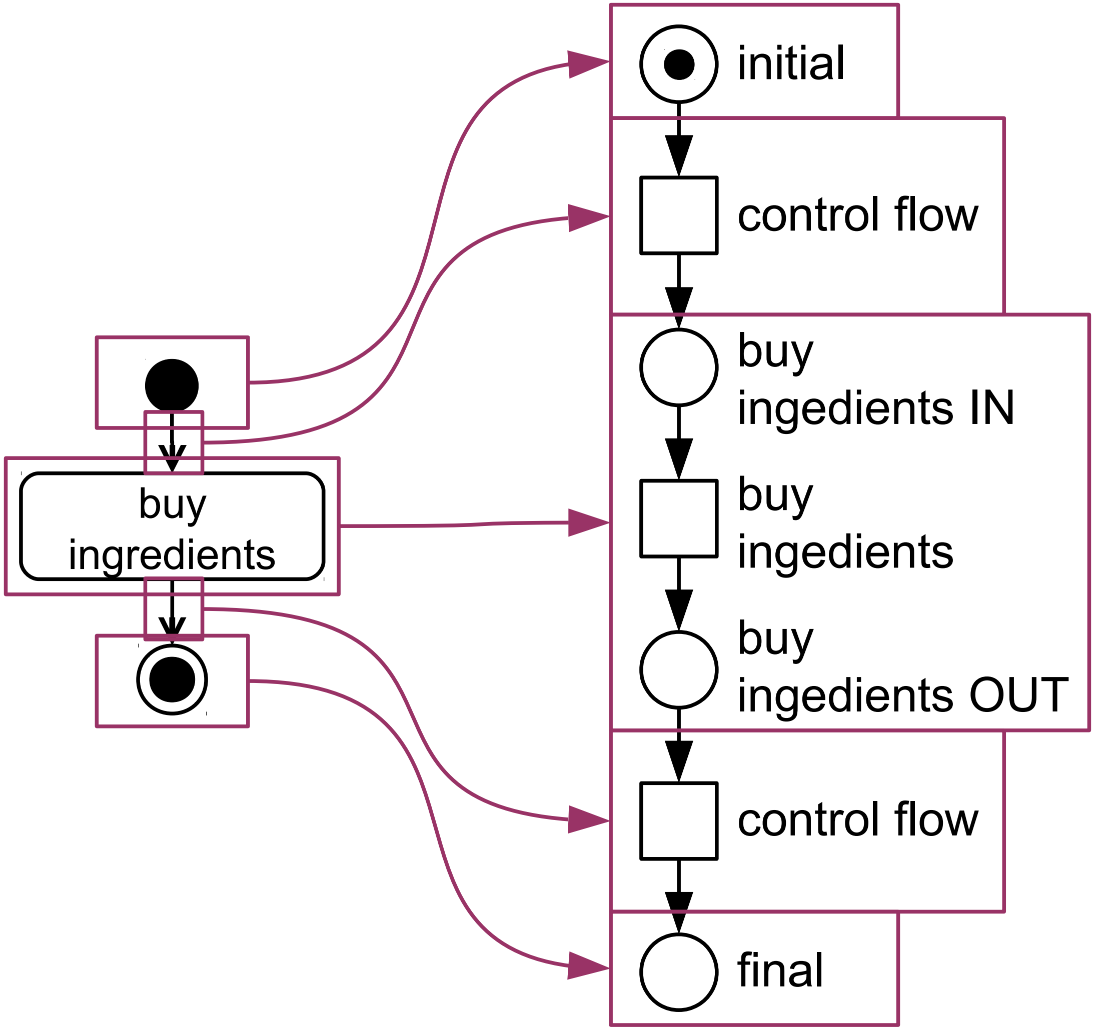
- Final node ↔ Empty Place



Example:Transform Activity Diagrams to Petri nets

in the last lecture...

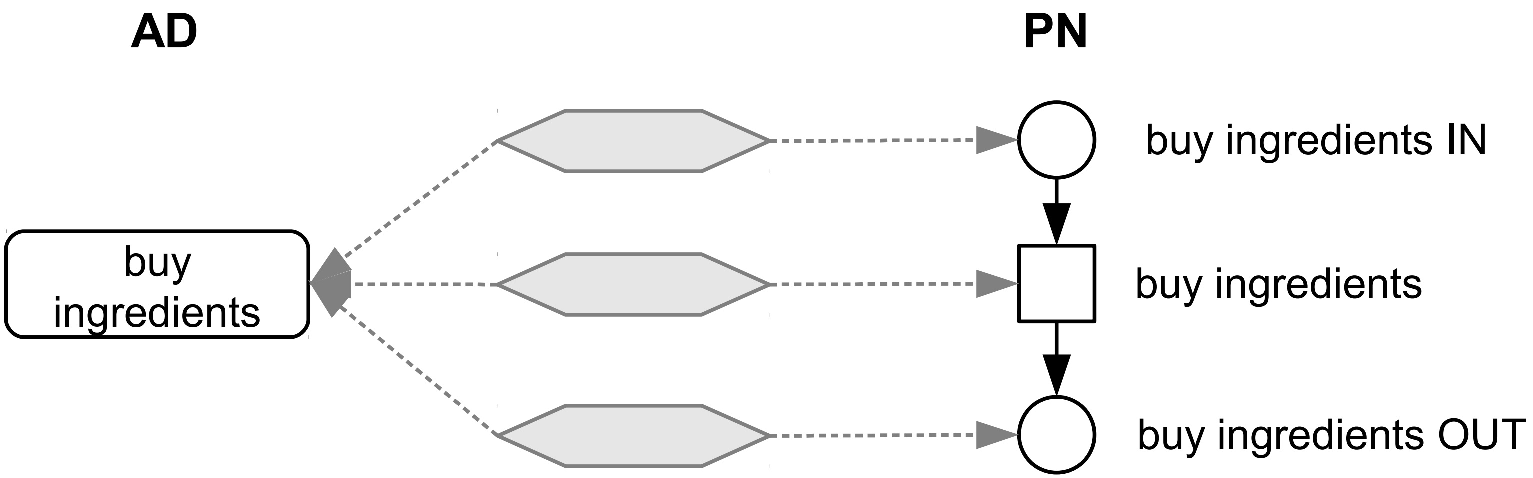
- Let's start simple: How to transform
  - Initial nodes?
  - Final nodes?
  - Action nodes?
  - Control flow edges?



Relations Between Model Patterns Example: Activity to Petri net

in the last lecture...

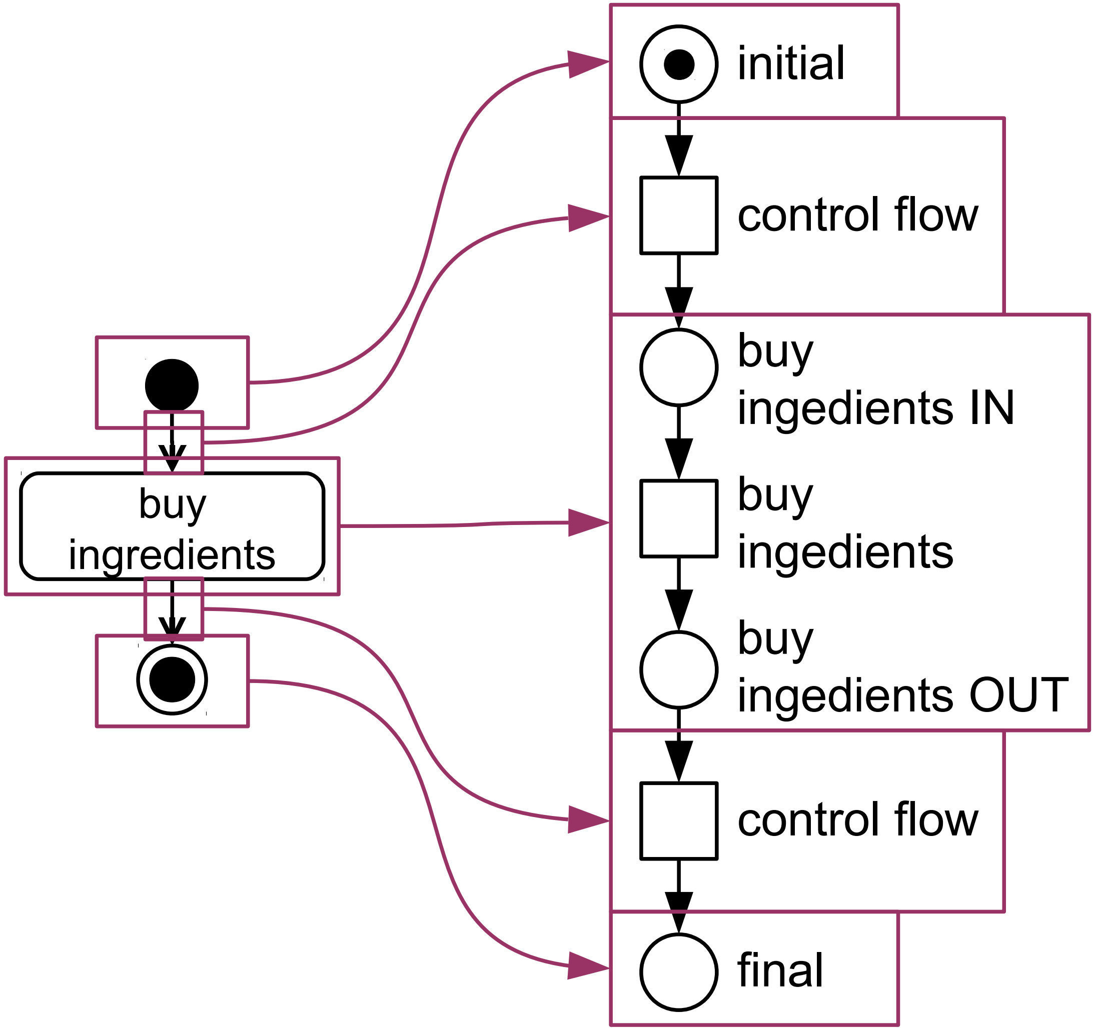
- Action node ↔ Transition with input and output place



Example:Transform Activity Diagrams to Petri nets

in the last lecture...

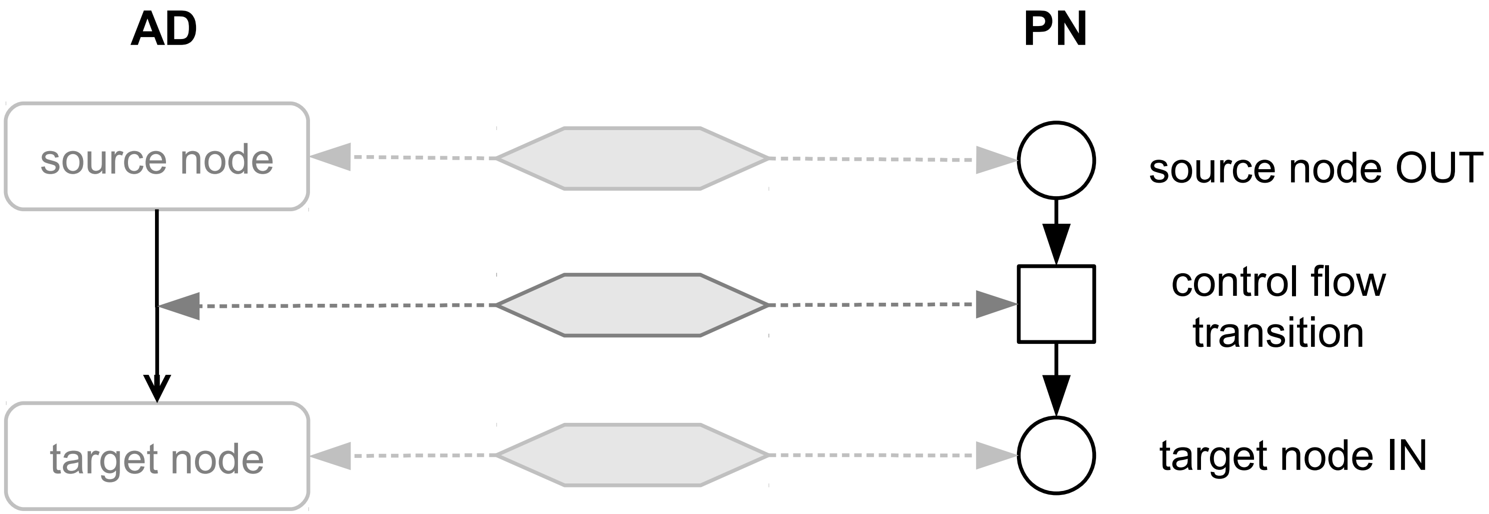
- Let's start simple: How to transform
  - Initial nodes?
  - Final nodes?
  - Action nodes?
  - Control flow edges?



Relations Between Model Patterns Example: Activity to Petri net

in the last lecture...

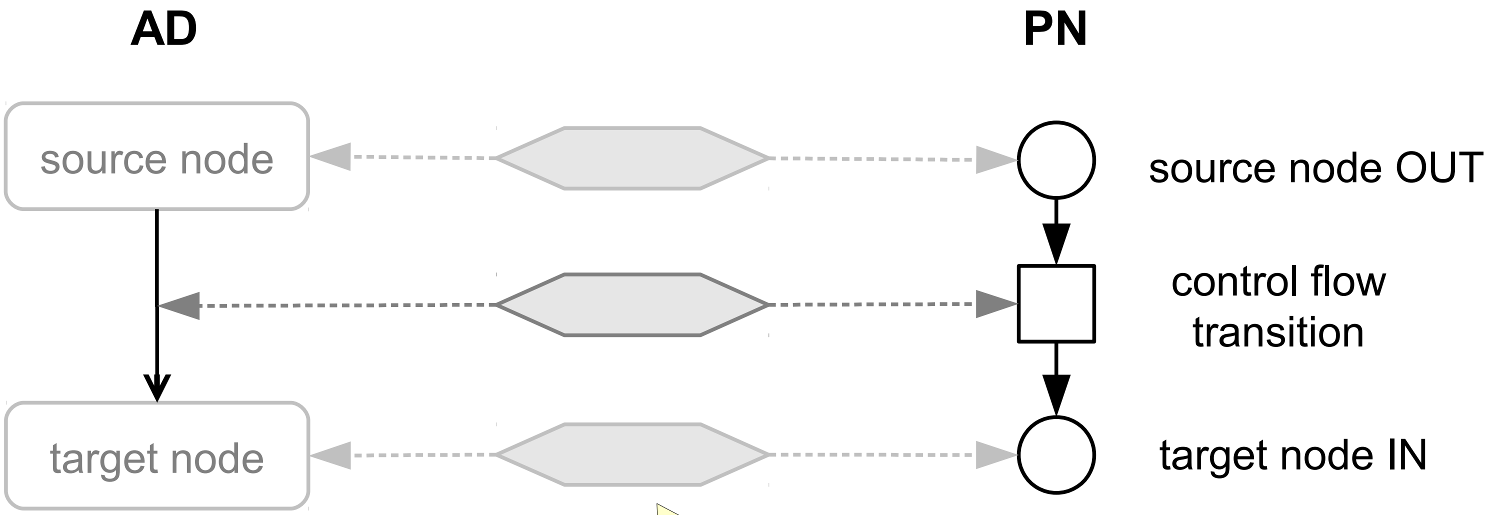
- Control flow edge ↔ Transition connecting in- and out places that correspond to the edge's source and target nodes



Relations Between Model Patterns Example: Activity to Petri net

in the last lecture...

- Control flow edge ↔ Transition connecting in- and out places that correspond to the edge's source and target nodes

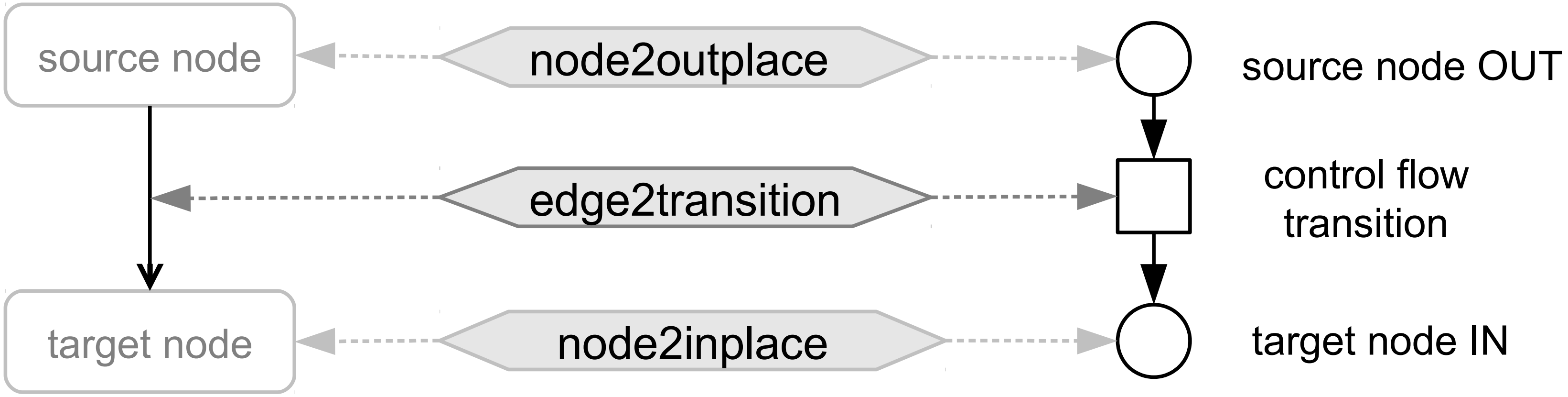


context: previously activity nodes and their mapping to petri net elements

Triple Graphs

in the last lecture...

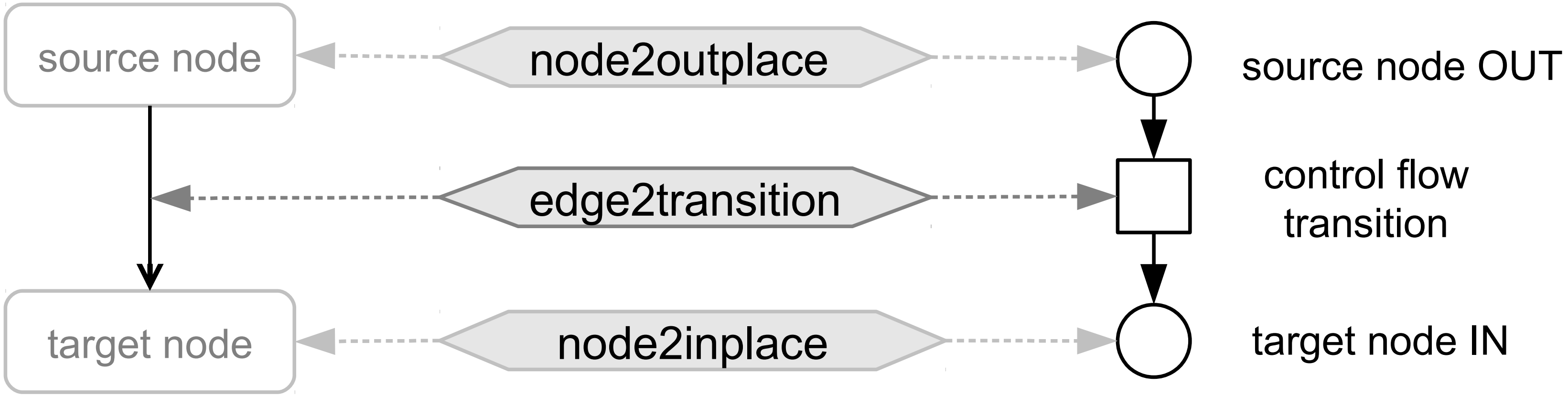
- **Idea 1**: describe the mapping of models as a **triple graph**



Triple Graphs

in the last lecture...

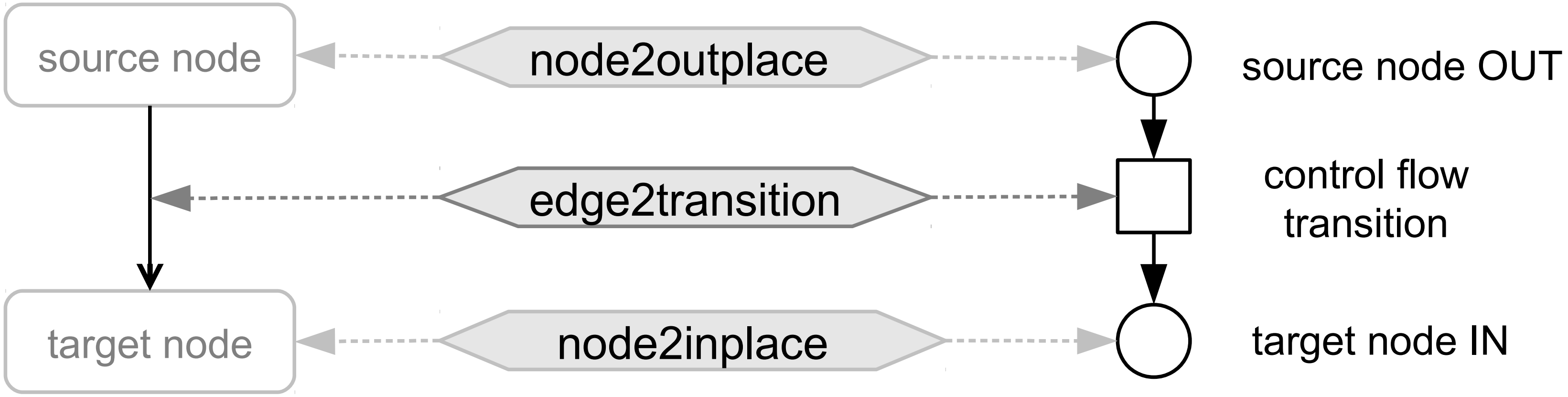
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?



Triple Graphs

in the last lecture...

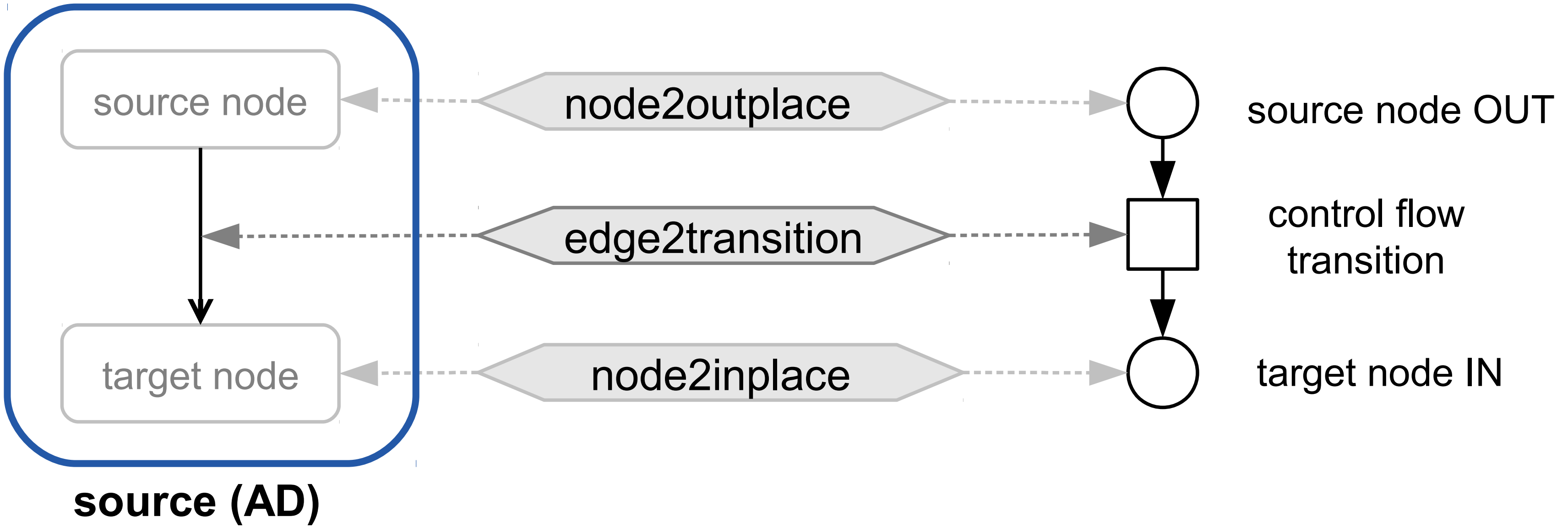
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?
  - **source** graph (model)



Triple Graphs

in the last lecture...

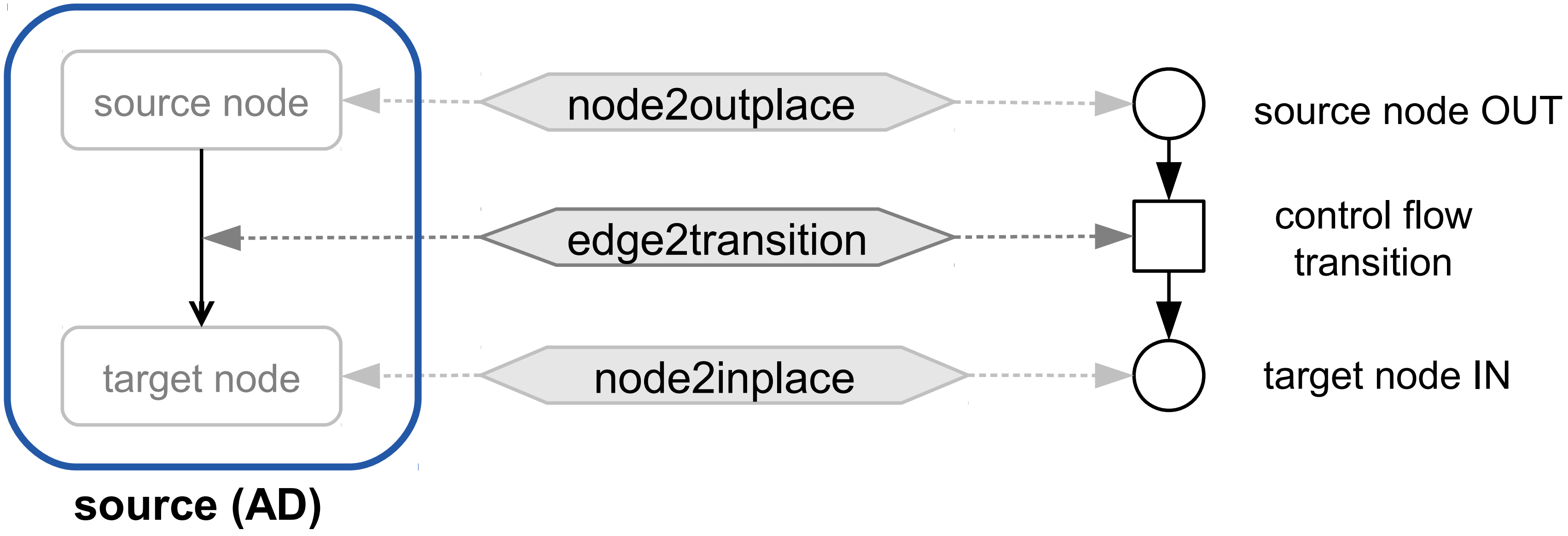
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?
  - **source** graph (model)



Triple Graphs

in the last lecture...

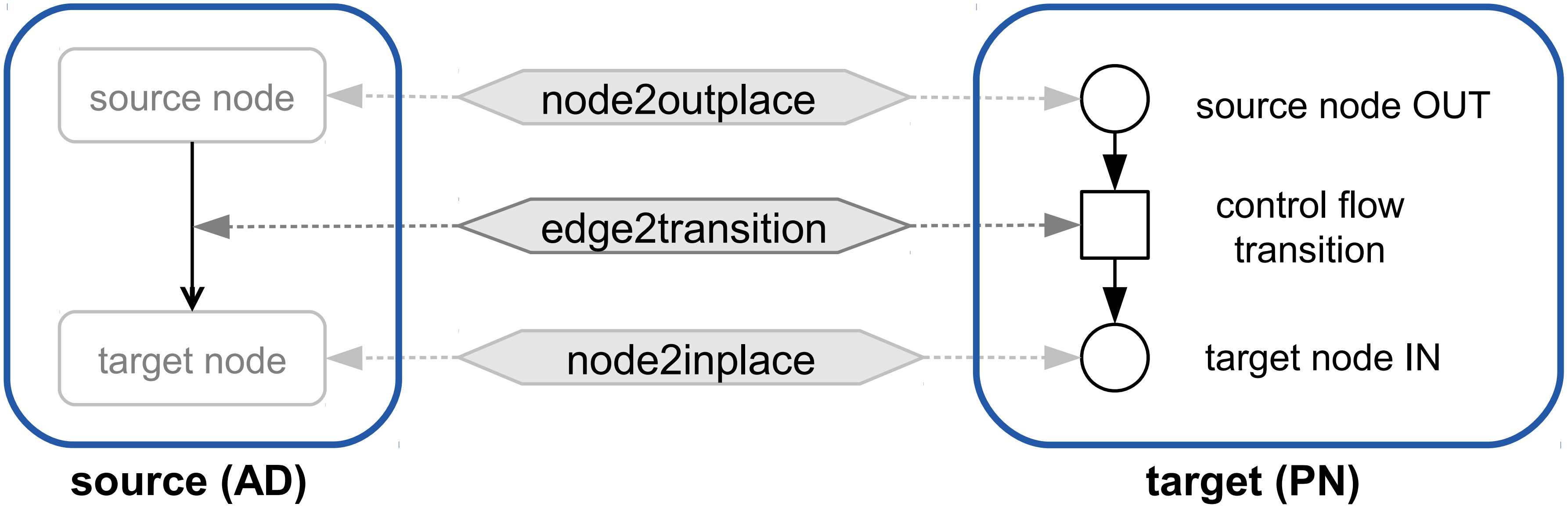
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?
  - **source** graph (model)
  - **target** graph (model)



Triple Graphs

in the last lecture...

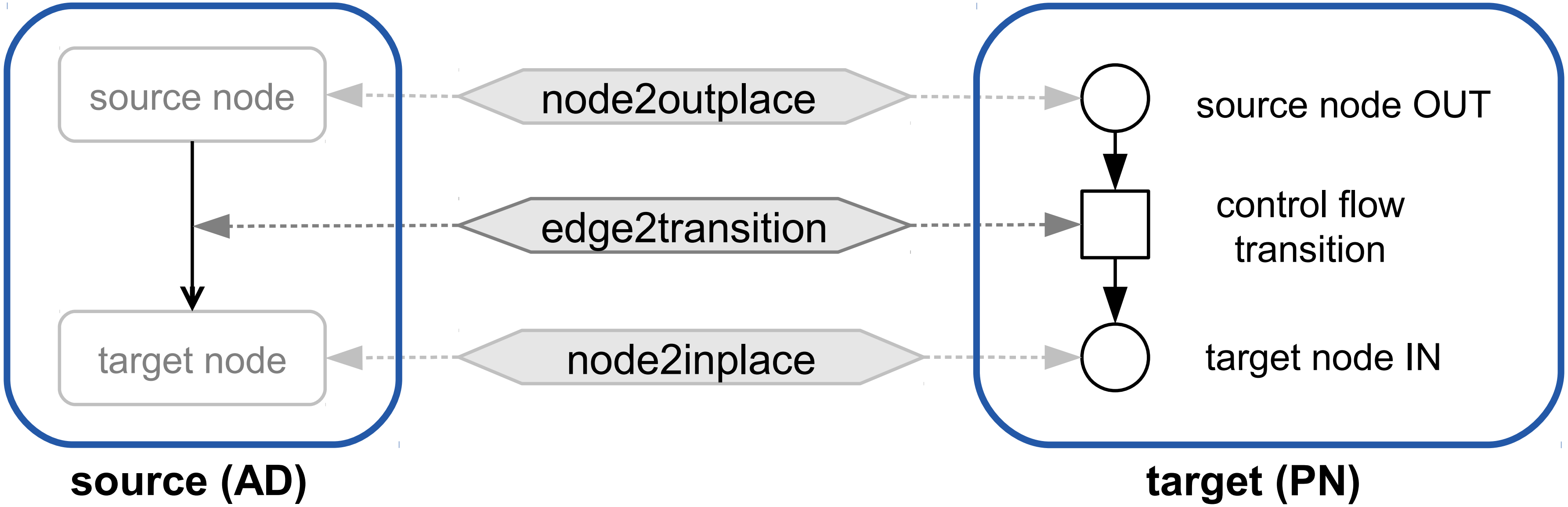
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?
  - **source** graph (model)
  - **target** graph (model)



Triple Graphs

in the last lecture...

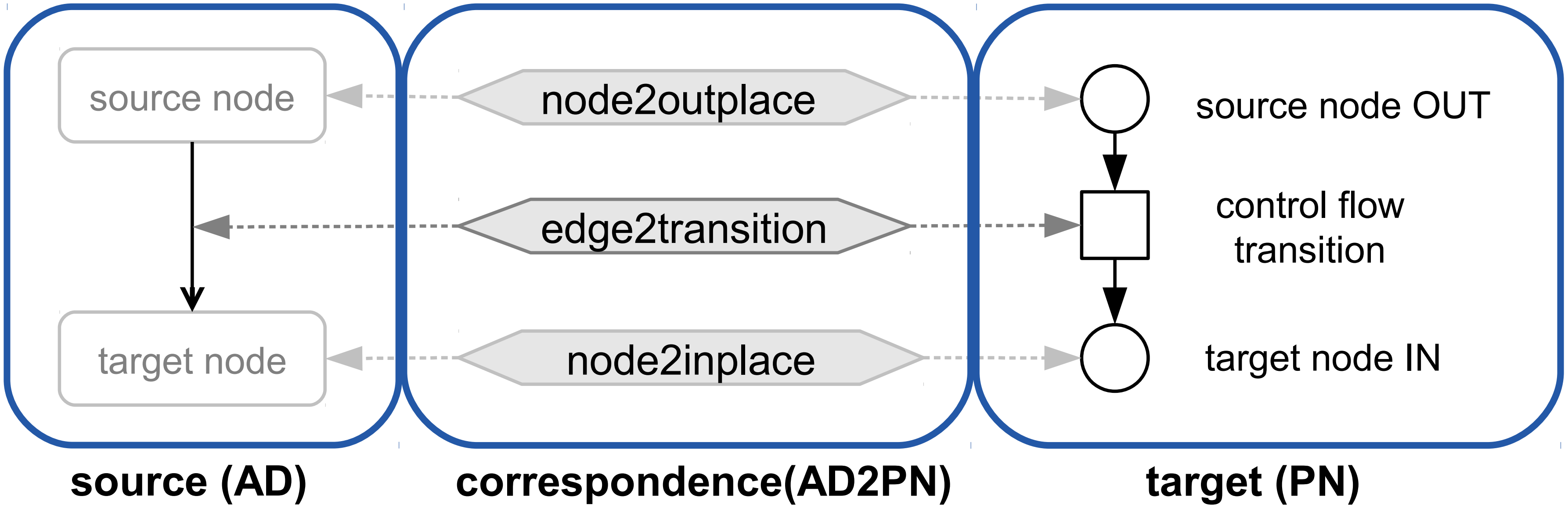
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?
  - **source** graph (model)
  - **target** graph (model)
  - **correspondence** graph (model) that connects the source and target graphs (models)



Triple Graphs

in the last lecture...

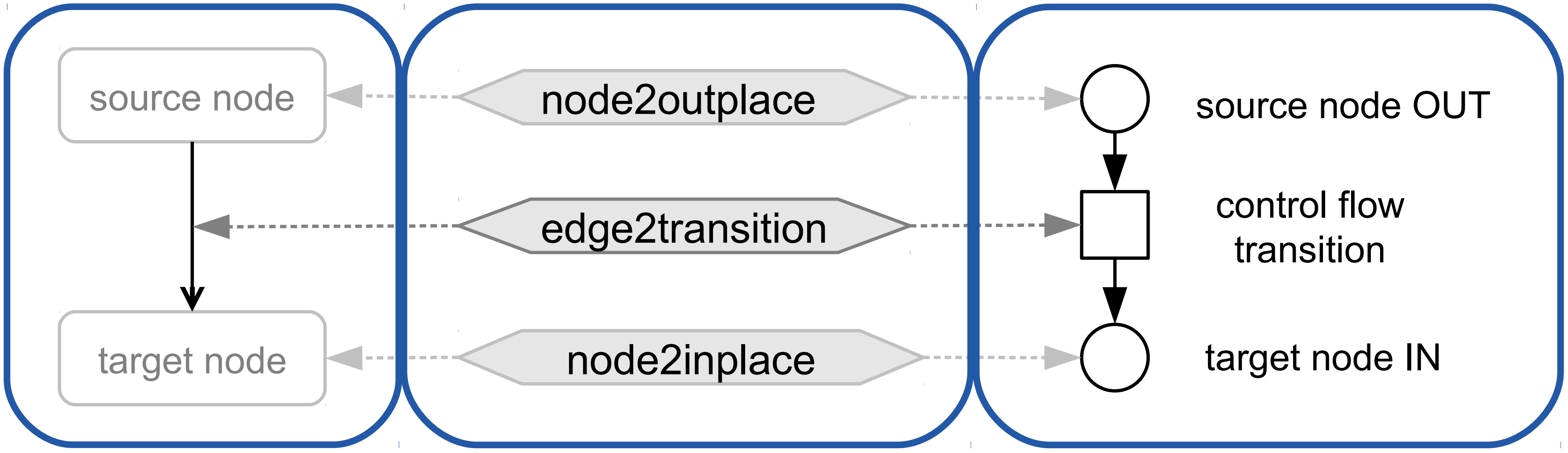
- **Idea 1**: describe the mapping of models as a **triple graph**
- What does a **triple graph** consist of?
  - **source** graph (model)
  - **target** graph (model)
  - **correspondence** graph (model) that connects the source and target graphs (models)



Triple Graphs

in the last lecture...

- The three different graphs (source, target, correspondence) are typed over (usually different) type graphs (metamodels)

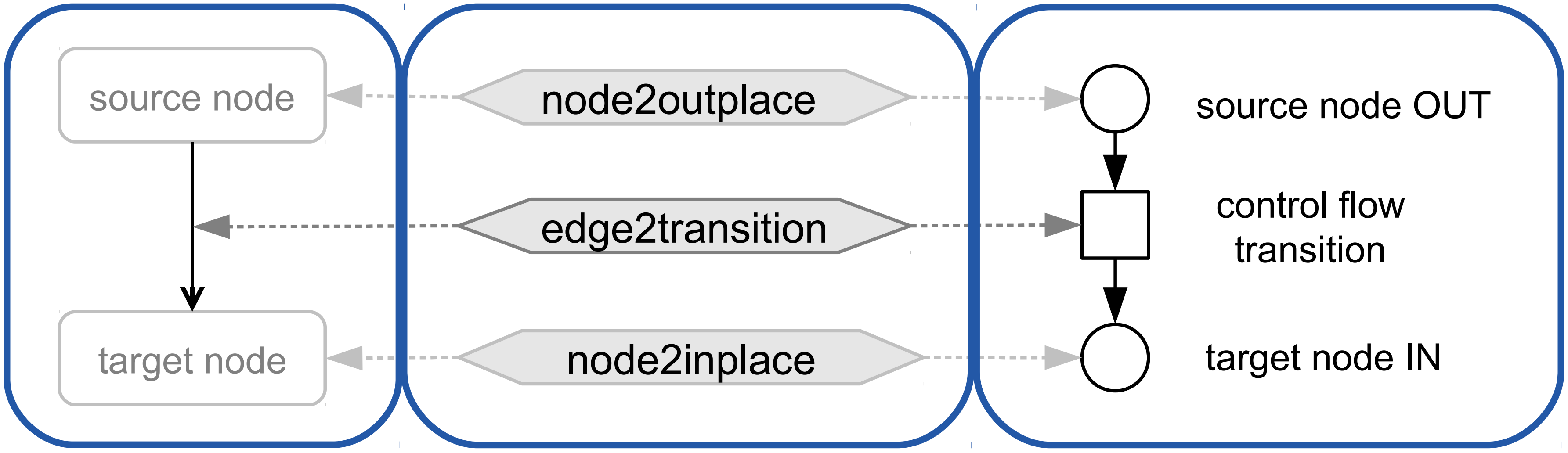


source (AD) **correspondence(AD2PN)** **target (PN)**

Triple Graphs

in the last lecture...

- The three different graphs (source, target, correspondence) are typed over (usually different) type graphs (metamodels)
- Also called source-, target-, and correspondence- **domain**

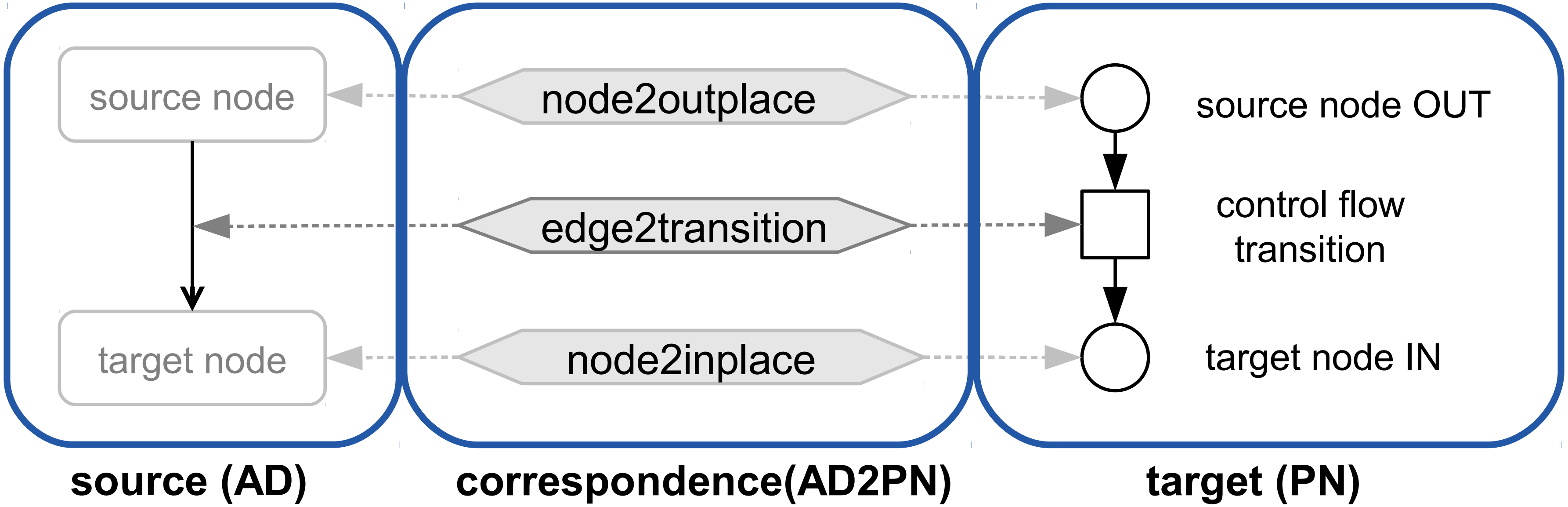


source (AD) **correspondence(AD2PN)** **target (PN)**

in the last lecture...

Triple Graphs

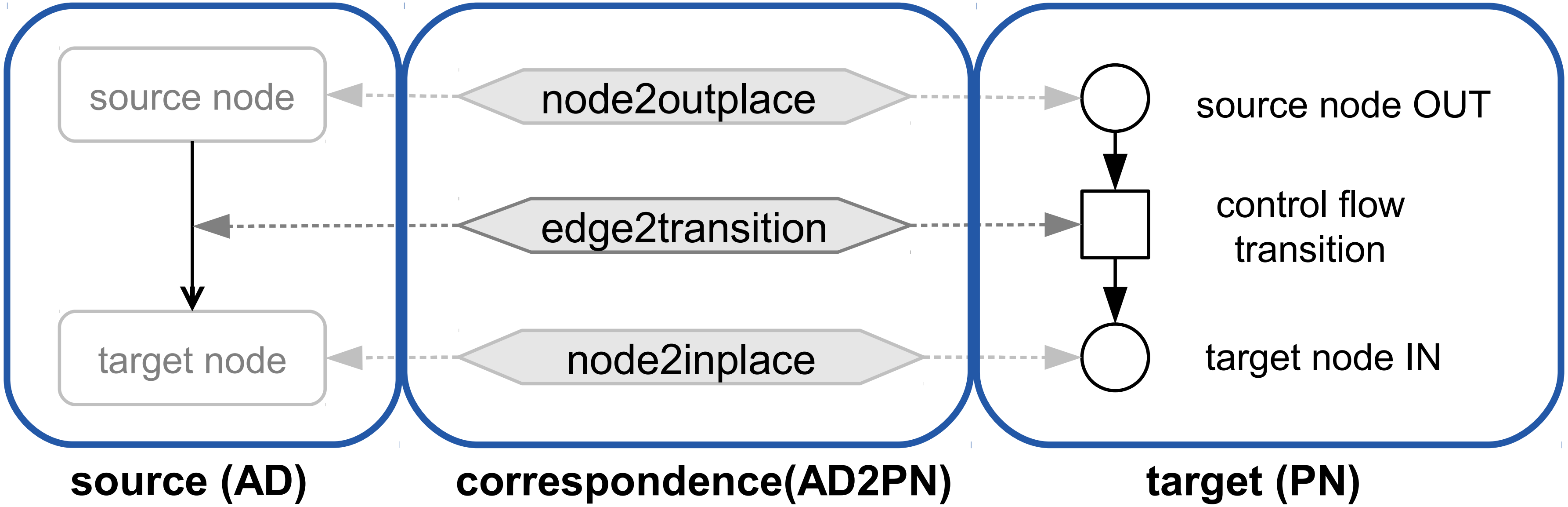
- The three different graphs (source, target, correspondence) are typed over (usually different) type graphs (metamodels)
- Also called source-, target-, and correspondence- **domain**
  - **source domain:** Activity Diagrams



Triple Graphs

in the last lecture...

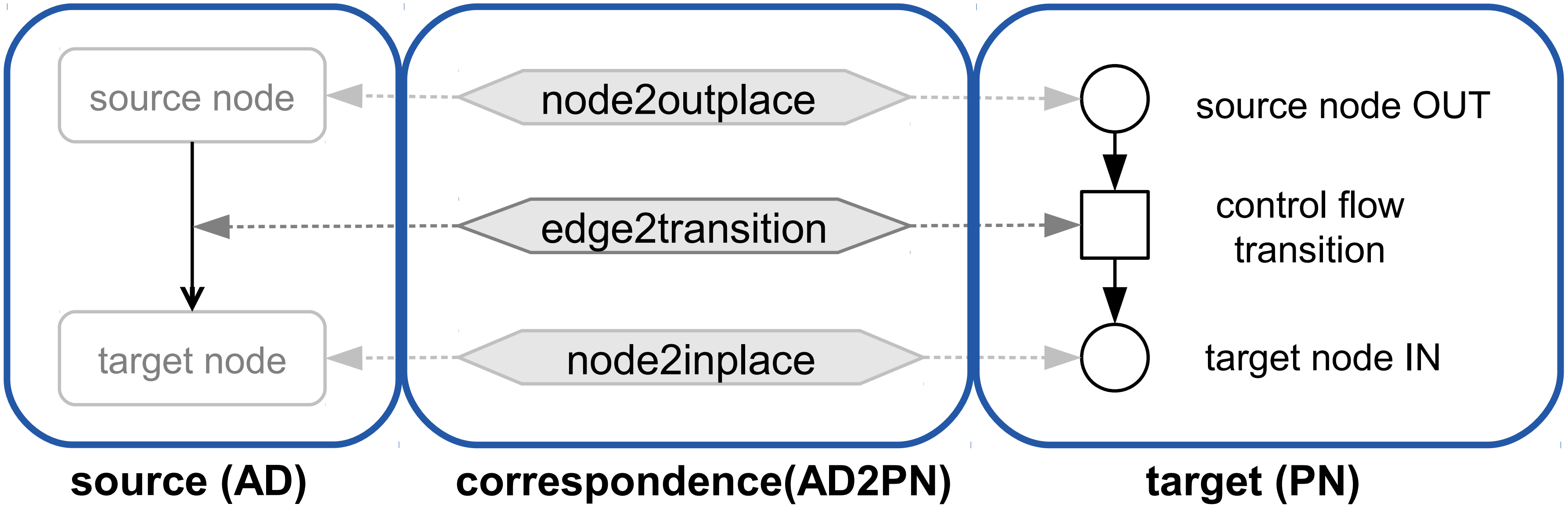
- The three different graphs (source, target, correspondence) are typed over (usually different) type graphs (metamodels)
- Also called source-, target-, and correspondence- **domain**
  - **source domain:** Activity Diagrams
  - **target domain:** Petri net



Triple Graphs

in the last lecture...

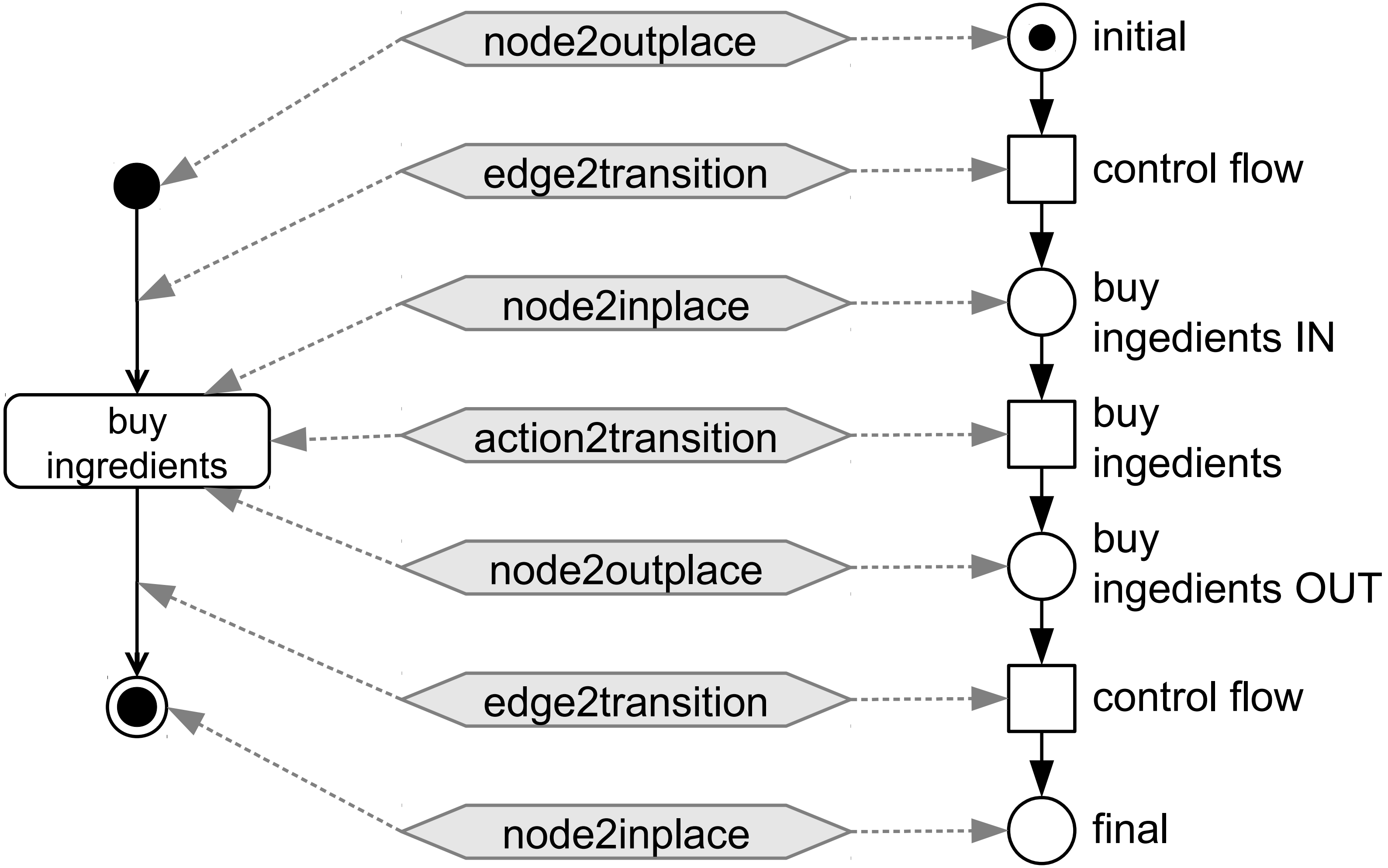
- The three different graphs (source, target, correspondence) are typed over (usually different) type graphs (metamodels)
- Also called source-, target-, and correspondence- **domain**
  - **source domain:** Activity Diagrams
  - **target domain:** Petri net
  - **correspondence domain:** AD2PN



Triple Graphs

in the last lecture...

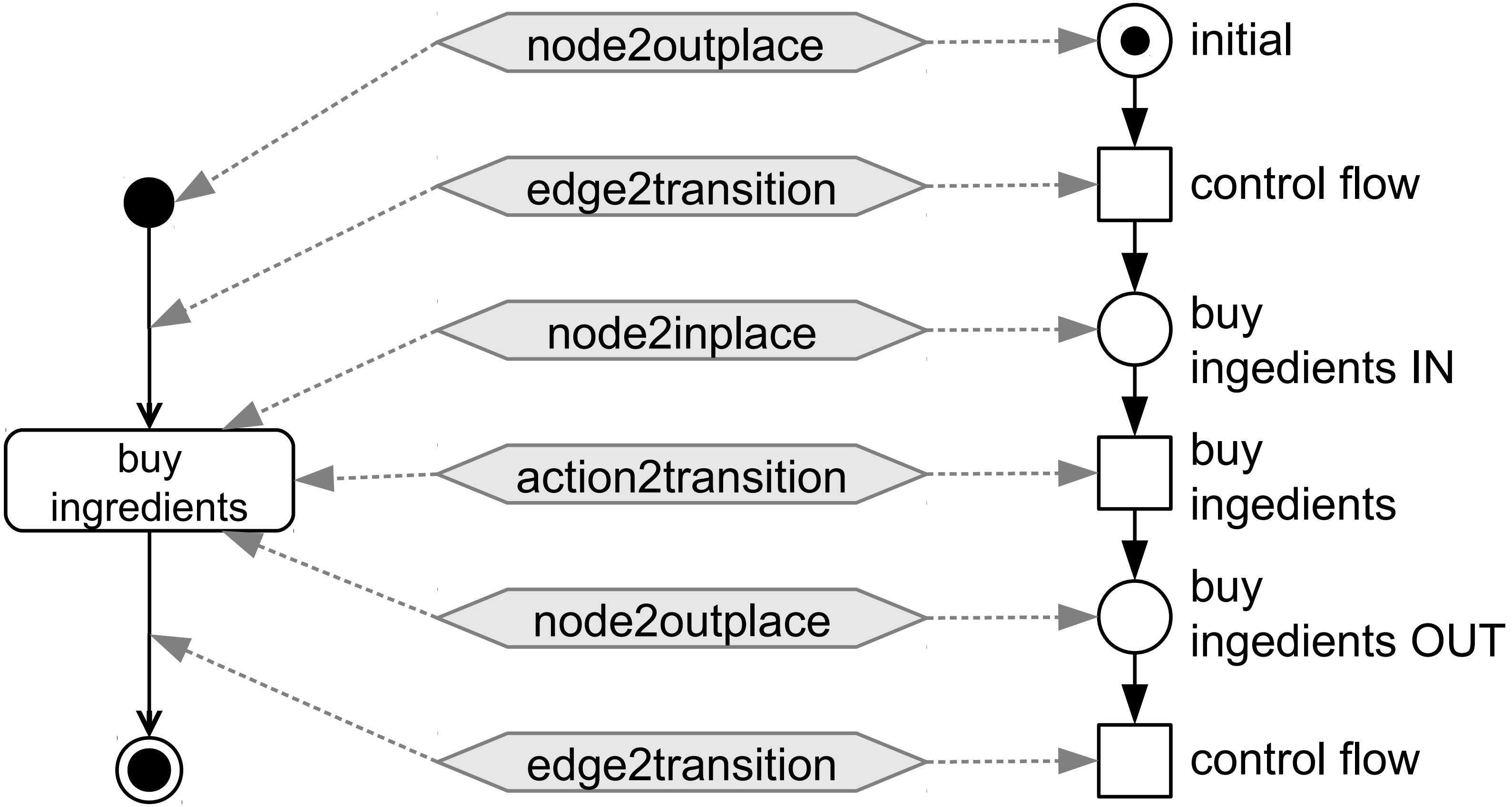
- Example of a bigger triple graph



Triple Graphs

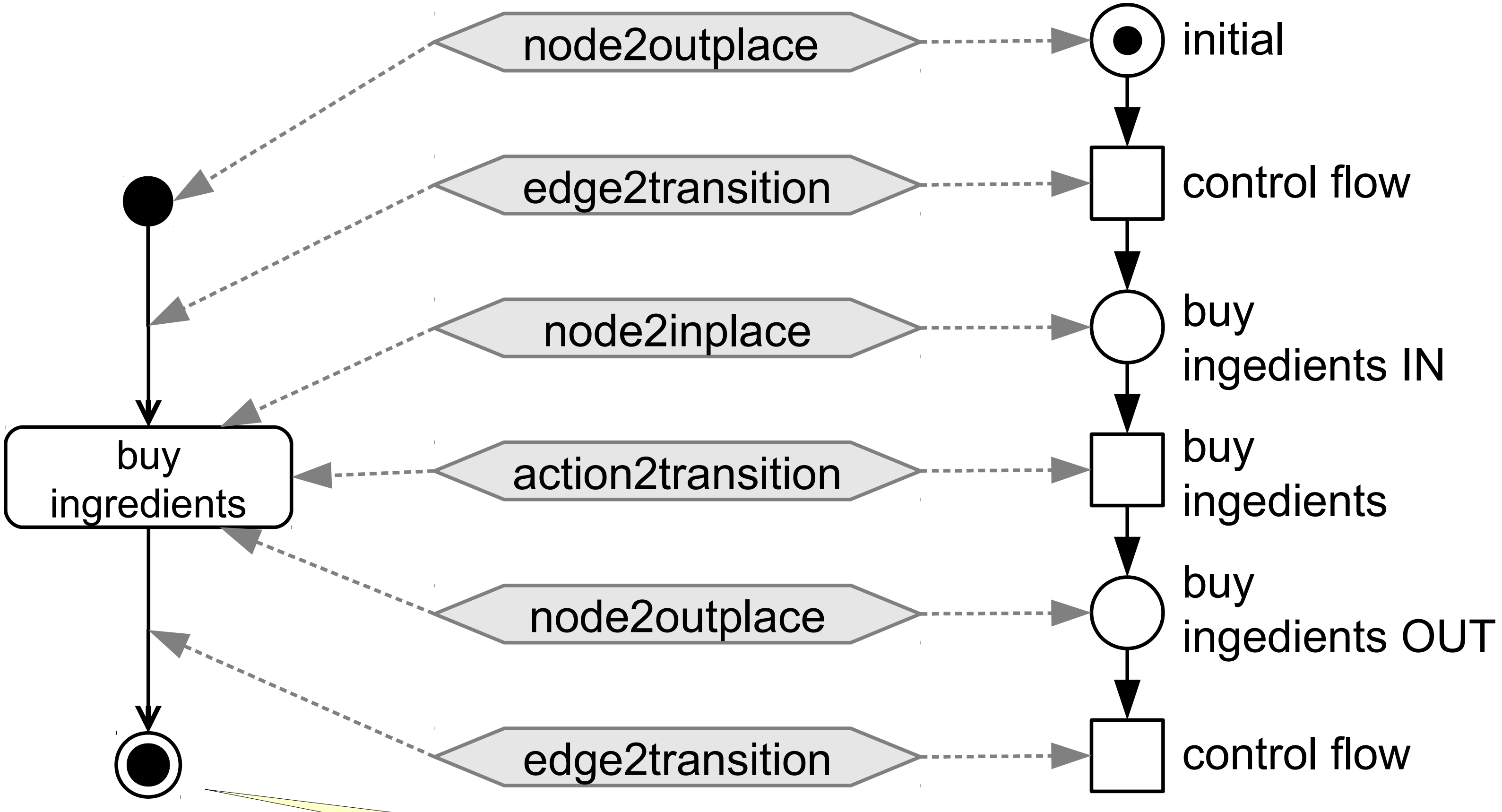
in the last lecture...

- An “invalid” triple graph



Triple Graphs

- An “invalid” triple graph



invalid: mapping of final node is missing

Triple Graph Grammar (TGG)

in the last lecture...

- How to describe which triple graphs are valid in which ones are not?
  - i.e., express which mappings are valid and which ones are not

Triple Graph Grammar (TGG)

in the last lecture...

- How to describe which triple graphs are valid in which ones are not?
  - i.e., express which mappings are valid and which ones are not

- **Idea 2**: Use a **graph grammar** that describes the production of valid triple graphs

Triple Graph Grammar (TGG)

in the last lecture...

- How to describe which triple graphs are valid in which ones are not?
  - i.e., express which mappings are valid and which ones are not

- **Idea 2**: Use a **graph grammar** that describes the production of valid triple graphs
  - → Triple Graph Grammar (TGG)

Graph Grammars

in lecture 8...

Graph Grammars

in lecture 8...

- A **graph grammar** consists of

Graph Grammars

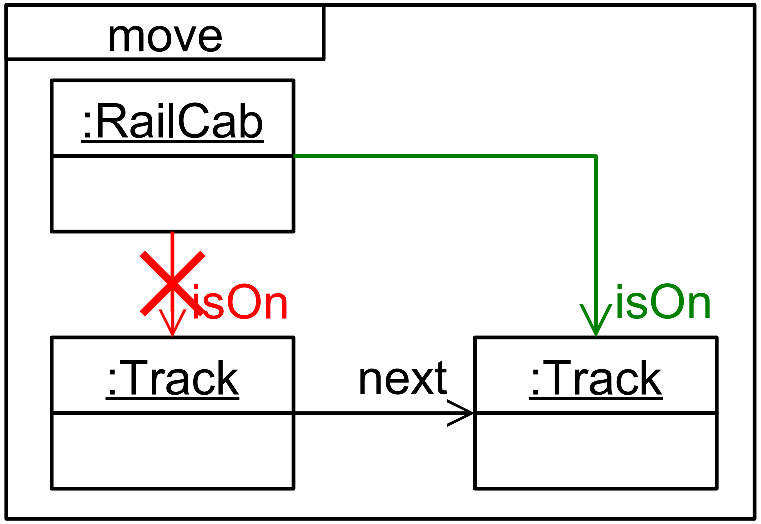
in lecture 8...

- A **graph grammar** consists of
  - a set of **graph grammar rules**

Graph Grammars

in lecture 8...

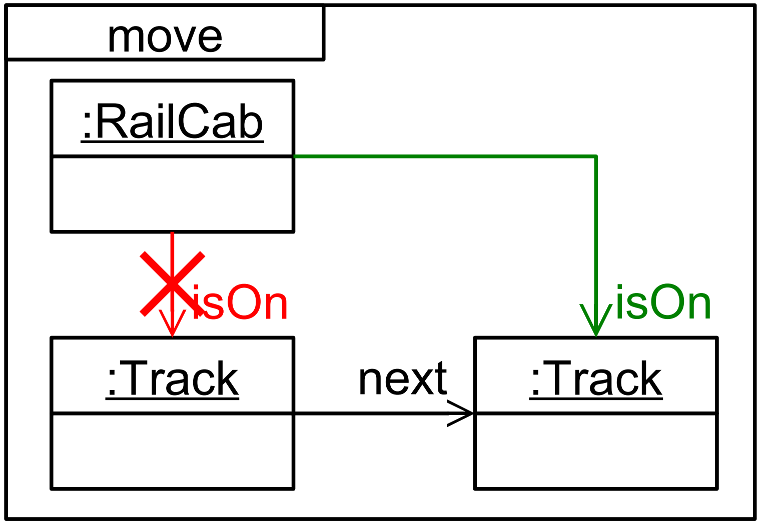
- A **graph grammar** consists of
  - a set of **graph grammar rules**



Graph Grammars

in lecture 8...

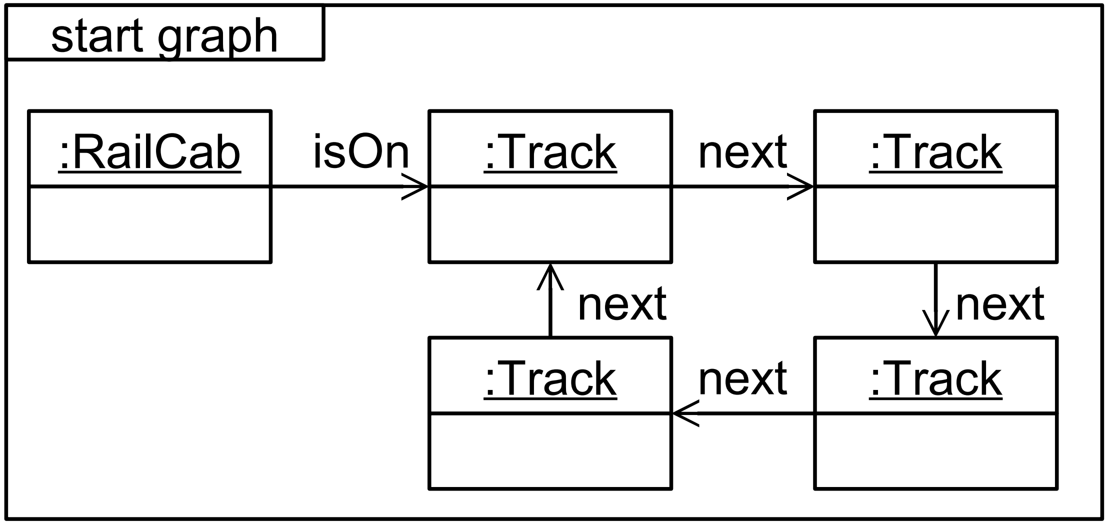
- A **graph grammar** consists of
  - a set of **graph grammar rules**
  - a **start graph** (also called **host graph**)

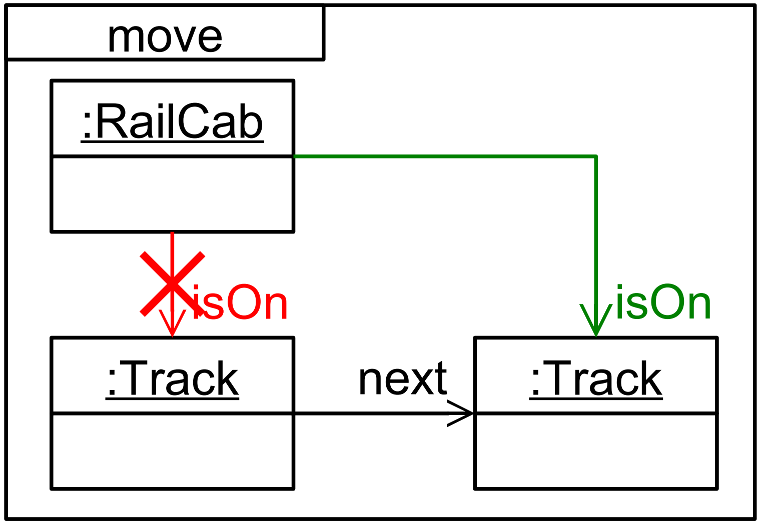


Graph Grammars

in lecture 8...

- A **graph grammar** consists of
  - a set of **graph grammar rules**
  - a **start graph** (also called **host graph**)

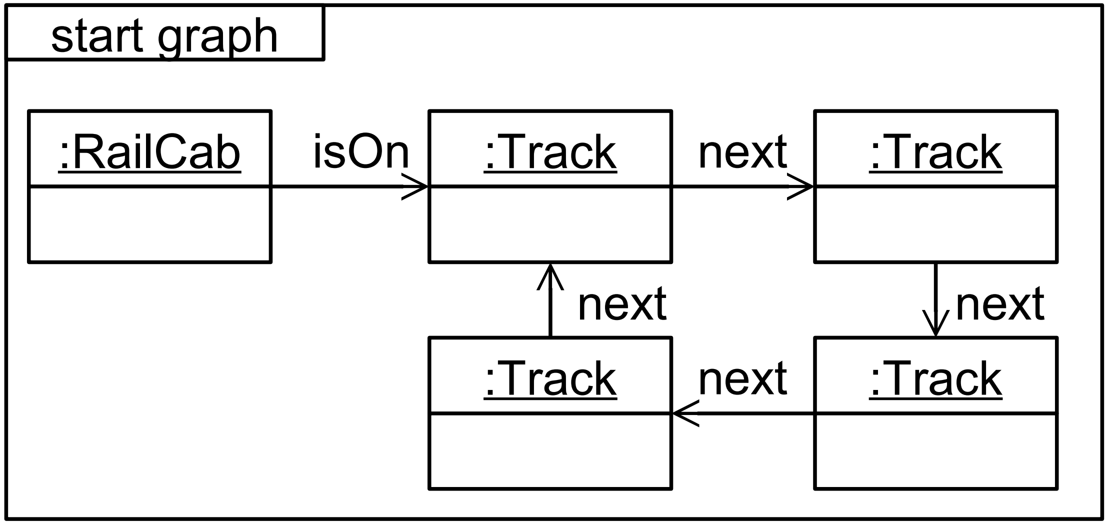


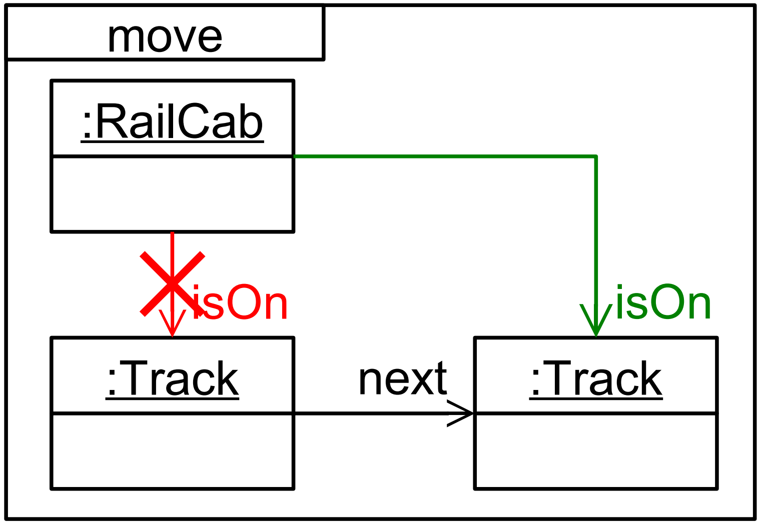


Graph Grammars

in lecture 8...

- A **graph grammar** consists of
  - a set of **graph grammar rules**
  - a **start graph** (also called **host graph**)
  - a **type graph**

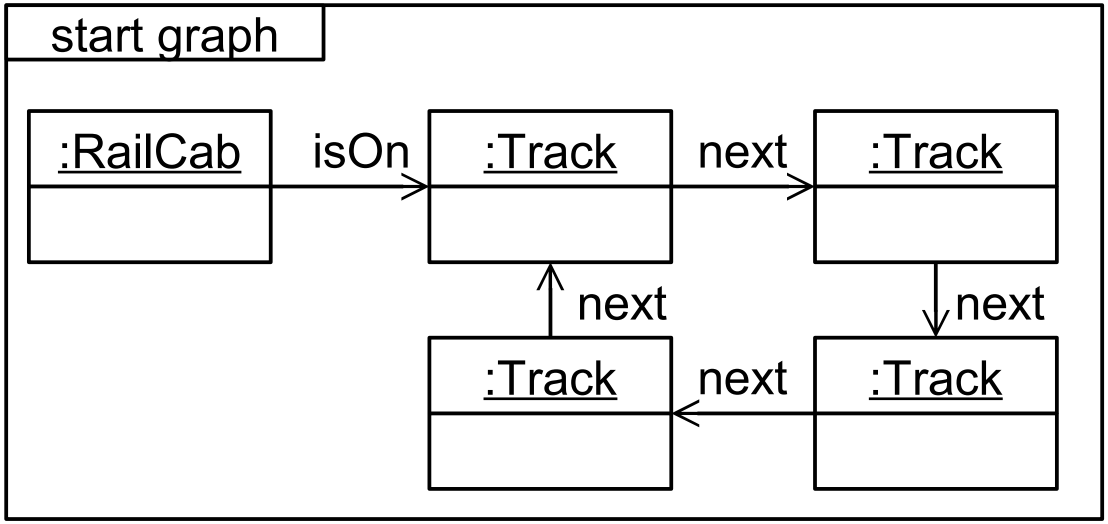


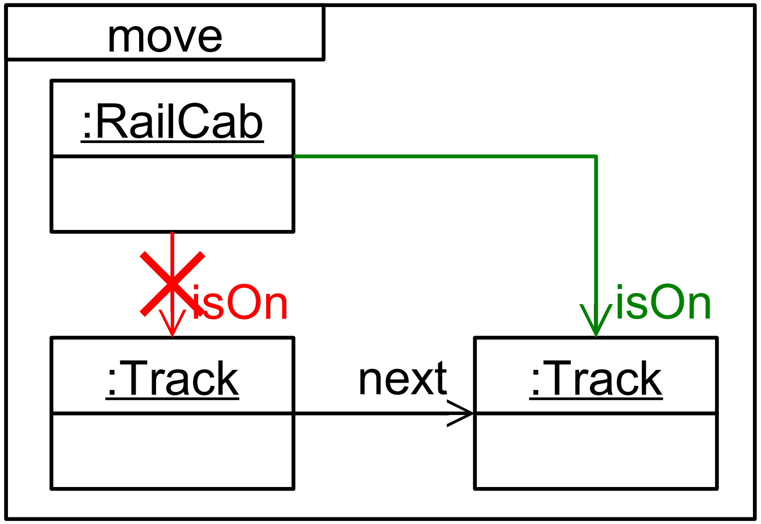


Graph Grammars

in lecture 8...

- A **graph grammar** consists of
  - a set of **graph grammar rules**
  - a **start graph** (also called **host graph**)
  - a **type graph**
- A graph grammar describes a (possibly infinite) set of graphs

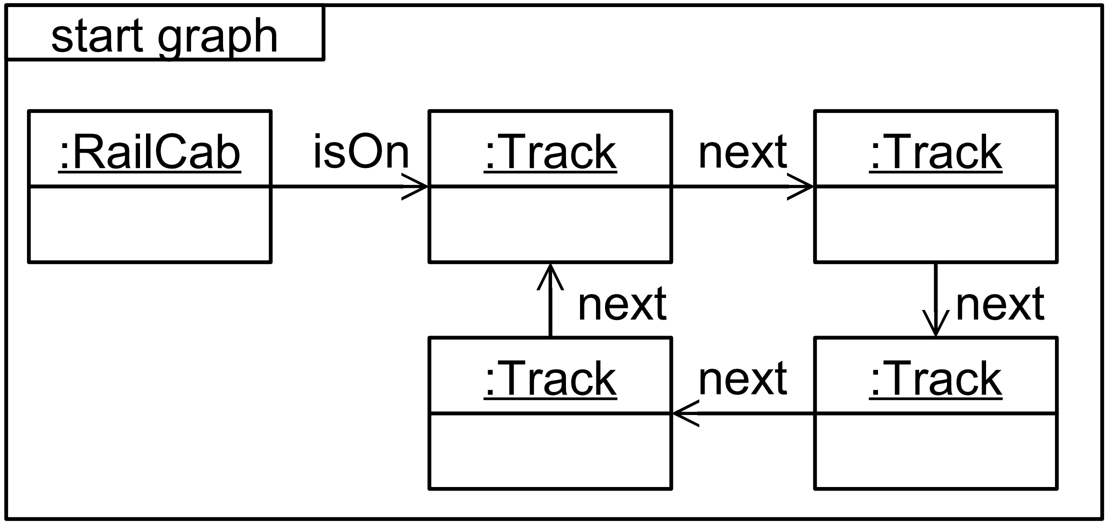


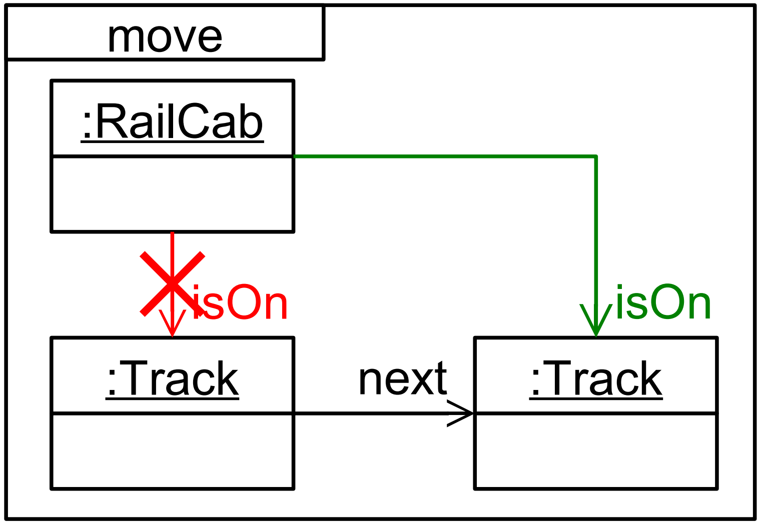


Graph Grammars

in lecture 8...

- A **graph grammar** consists of
  - a set of **graph grammar rules**
  - a **start graph** (also called **host graph**)
  - a **type graph**
- A graph grammar describes a (possibly infinite) set of graphs
  - those that can be constructed from the start graph by applying the graph grammar rules in all possible orders

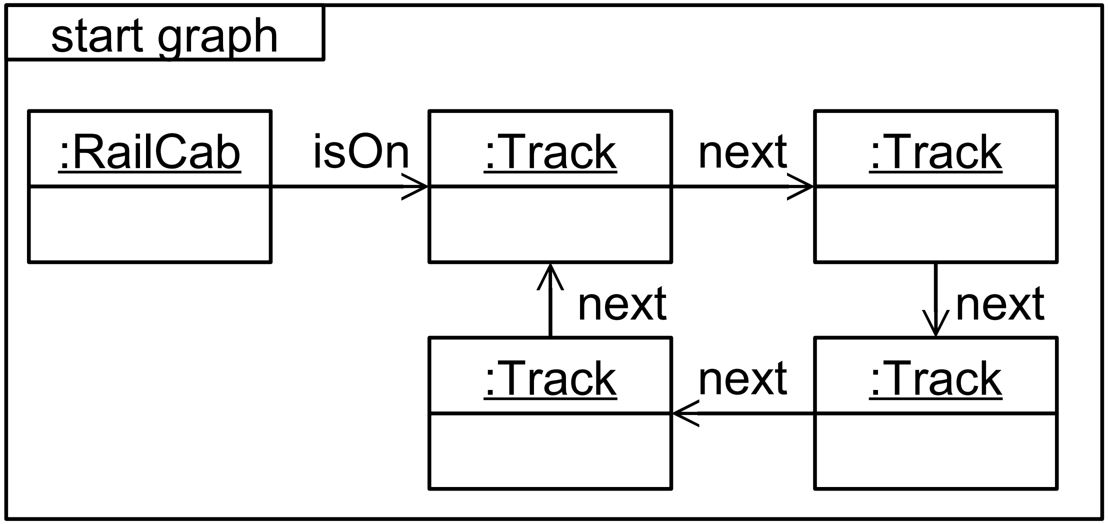


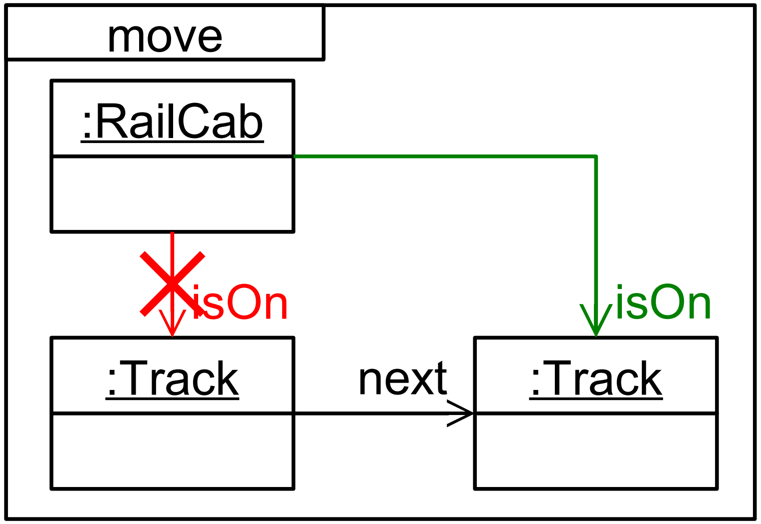


Graph Grammars

in lecture 8...

- A **graph grammar** consists of
  - a set of **graph grammar rules**
  - a **start graph** (also called **host graph**)
  - a **type graph**
- A graph grammar describes a (possibly infinite) set of graphs
  - those that can be constructed from the start graph by applying the graph grammar rules in all possible orders
- Graph grammars are also called **Graph Transformation Systems**





Triple Graph Grammar (TGG) Example

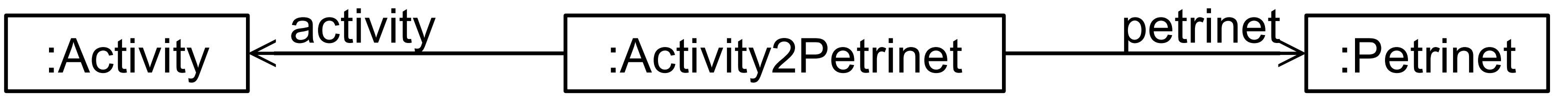
- **TGGs are also regular graph grammars**

Triple Graph Grammar (TGG) Example

- **TGGs are also regular graph grammars**
  - hence, they also define a **start graph** or **axiom** (often a mapping of two model's root nodes)

Triple Graph Grammar (TGG) Example

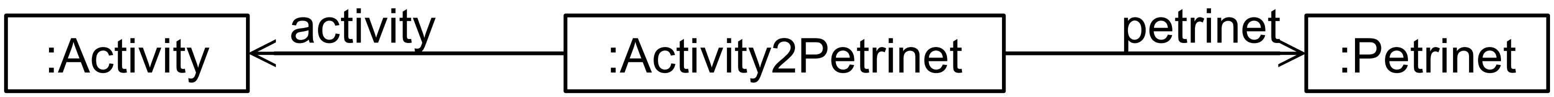
- **TGGs are also regular graph grammars**
  - hence, they also define a **start graph** or **axiom** (often a mapping of two model's root nodes)



Triple Graph Grammar (TGG) Example

- **TGGs are also regular graph grammars**
  - hence, they also define a **start graph** or **axiom**

    (often a mapping of two model's root nodes)



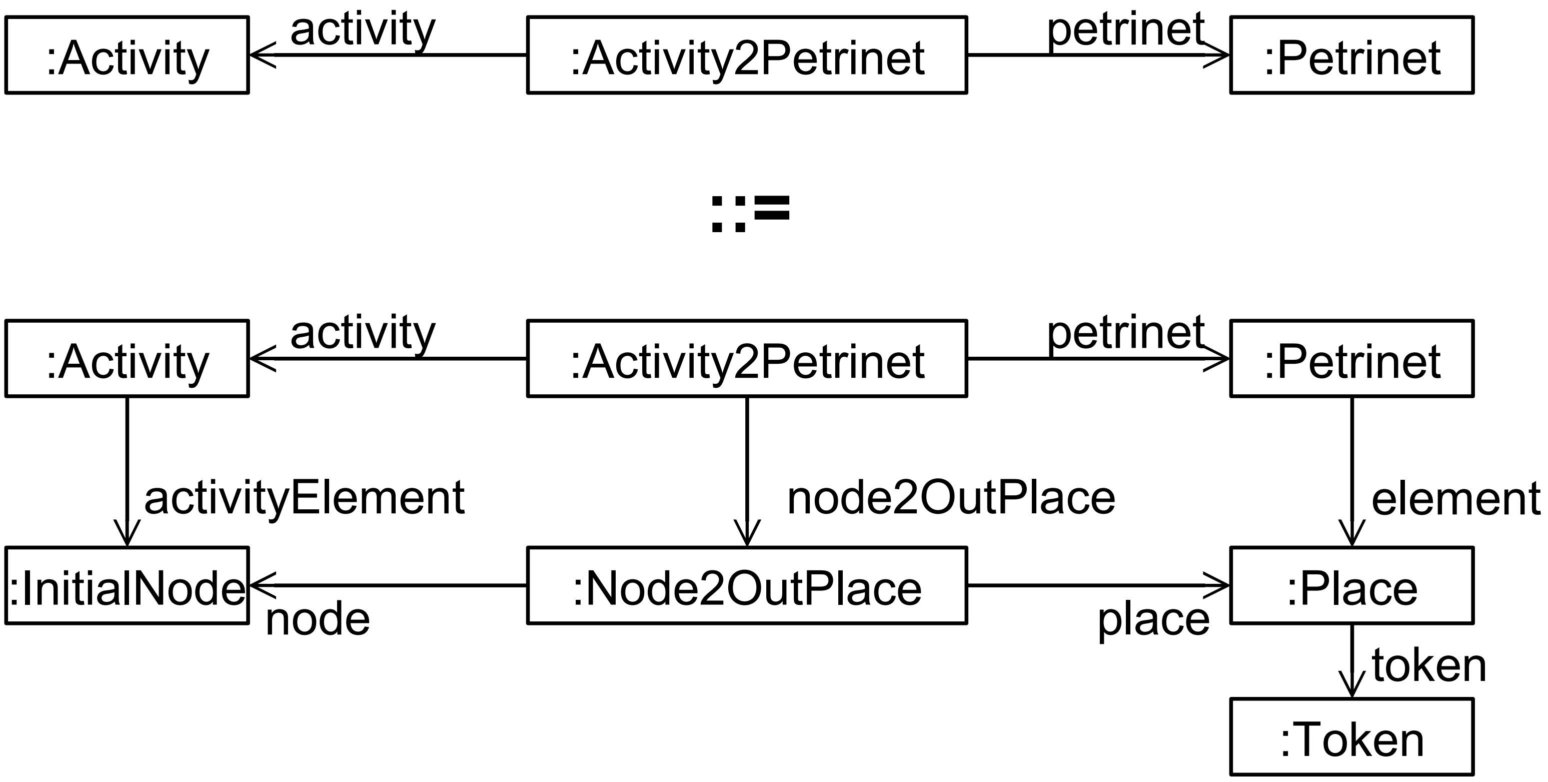
- The axiom is the smallest valid triple graph
  - as we will see, TGG rules are non-deleting

Triple Graph Grammar (TGG) Example

- A TGG rule for extending a valid triple graph

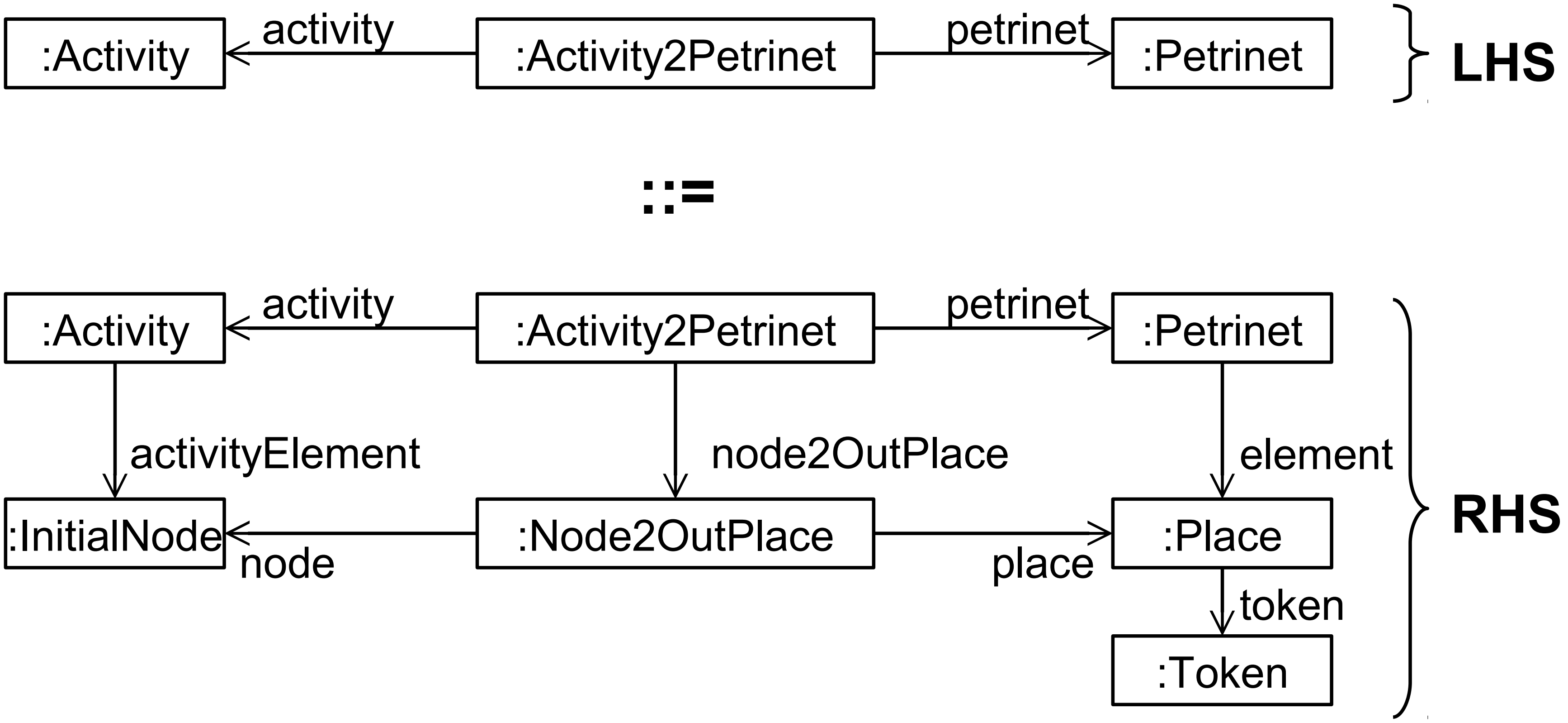
Triple Graph Grammar (TGG) Example

- A TGG rule for extending a valid triple graph



Triple Graph Grammar (TGG) Example

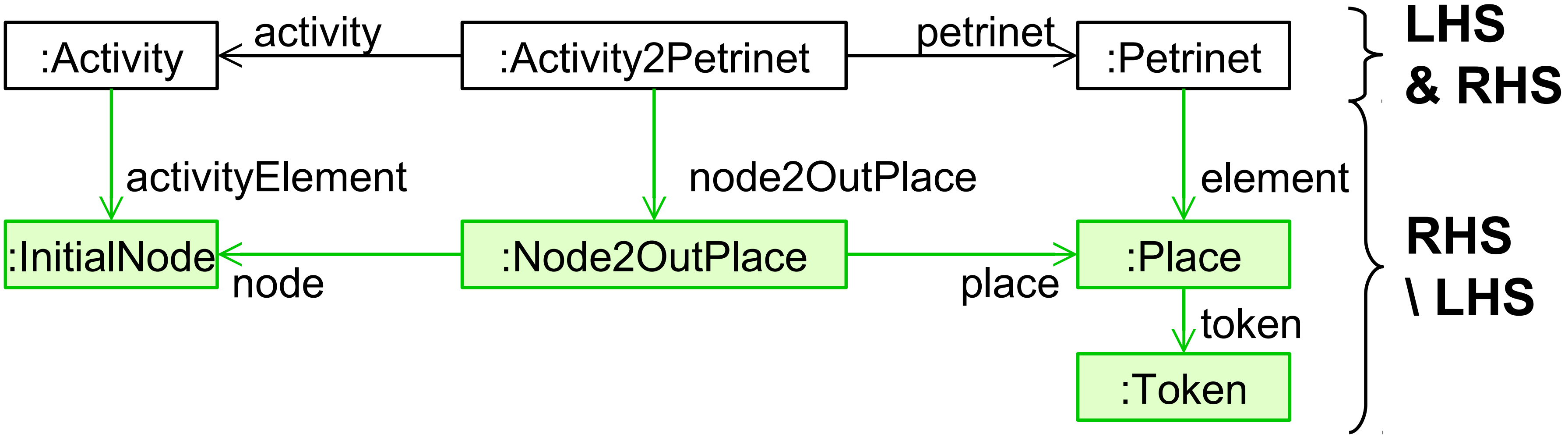
- A TGG rule for extending a valid triple graph



Triple Graph Grammar (TGG) Example

- A TGG rule for extending a valid triple graph

- in shorthand notation:

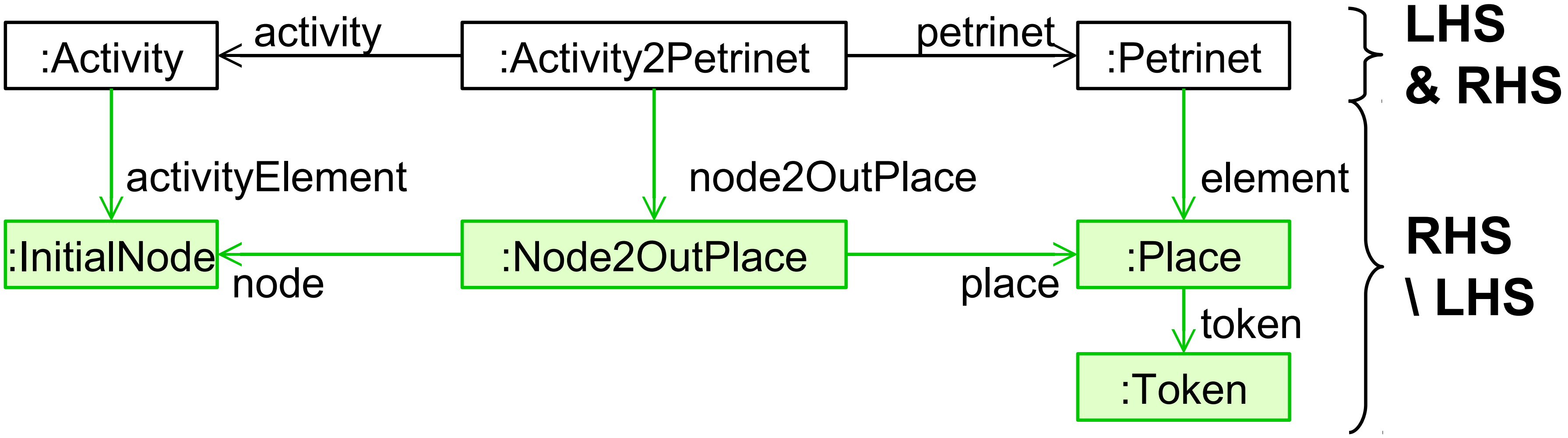


Triple Graph Grammar (TGG) Example

- A TGG rule for extending a valid triple graph

- in shorthand notation:

Intersection of LHS and RHS graph is also called the rule's **context graph**

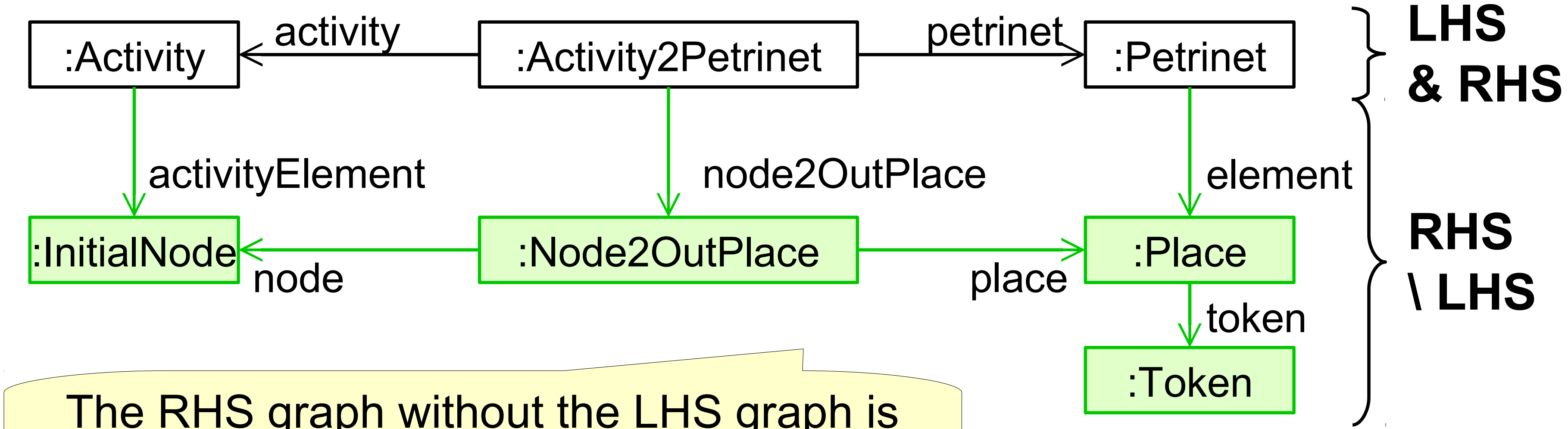


Triple Graph Grammar (TGG) Example

- A TGG rule for extending a valid triple graph

- in shorthand notation:

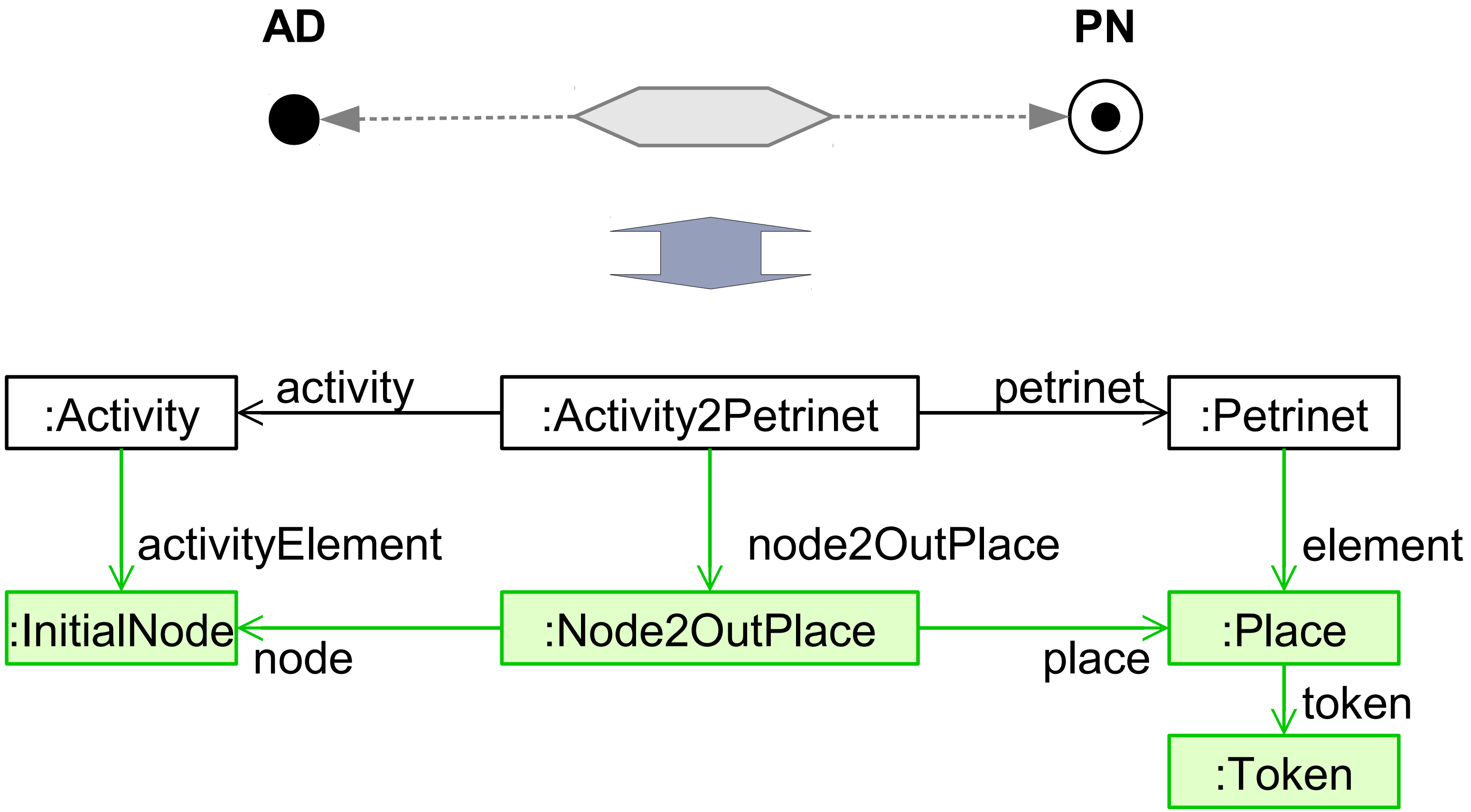
Intersection of LHS and RHS graph is also called the rule's **context graph**



The RHS graph without the LHS graph is also called the rule's **produced graph**

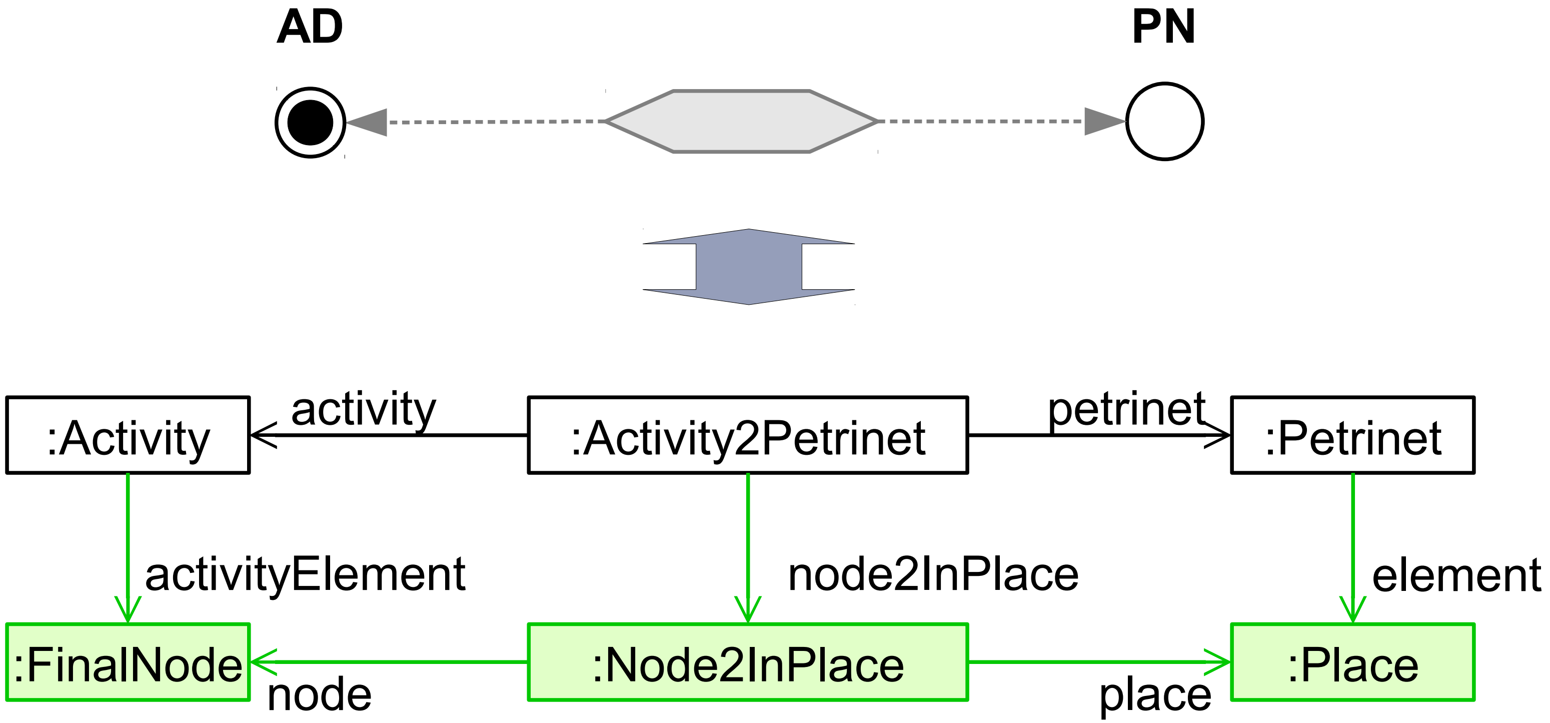
Intuitive Relation vs TGG Rule

- TGG rules are very close to the intuitive relation we described before



Intuitive Relation vs TGG Rule

- Final node ↔ Empty Place (similar to the rule before)



AD

PN

:Activity

activity

:Activity2Petrinet

petrinet

:Petrinet

activityElement

node2InPlace

element

:FinalNode

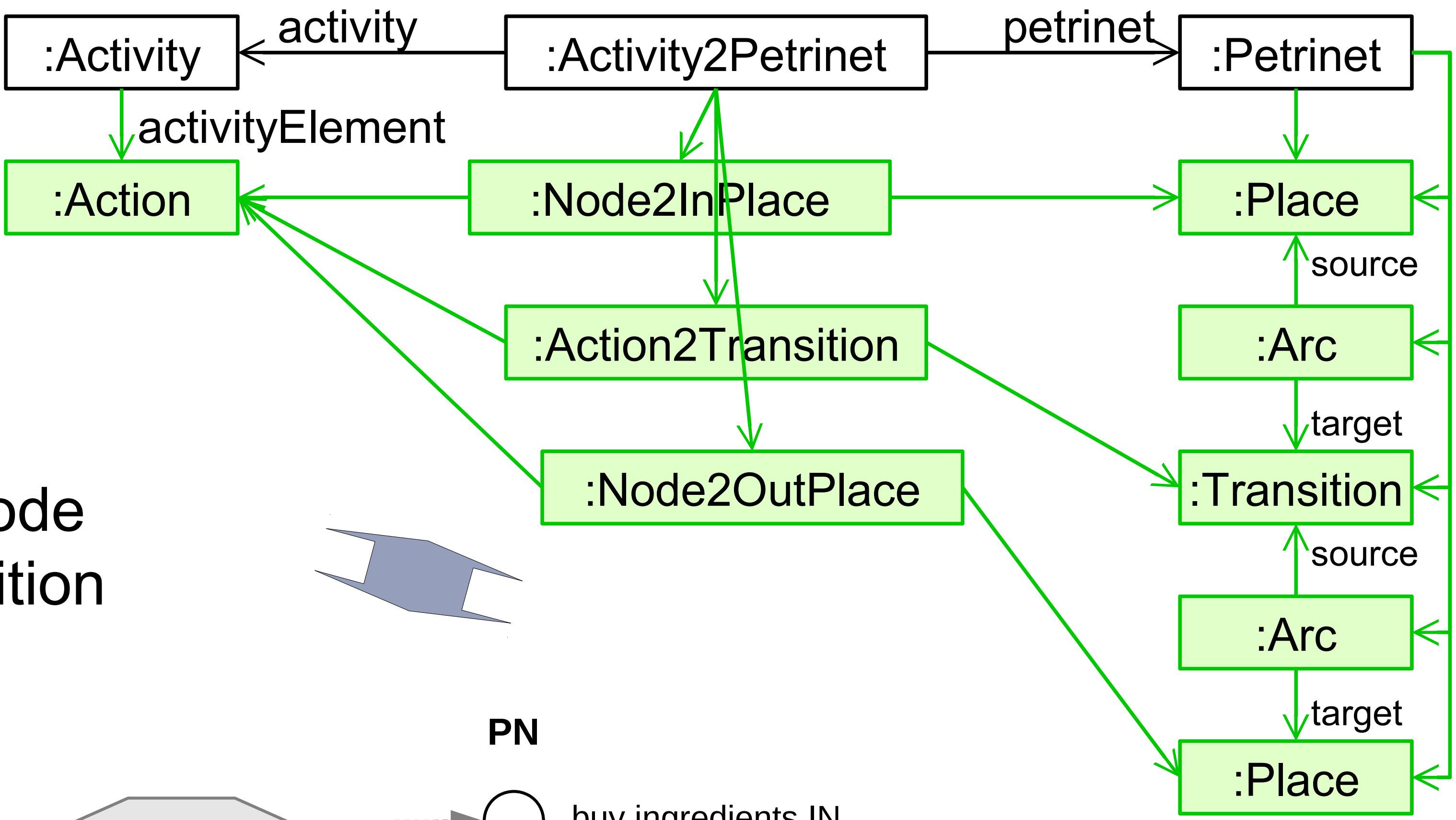
node

:Node2InPlace

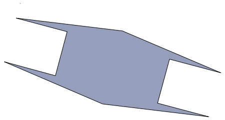
place

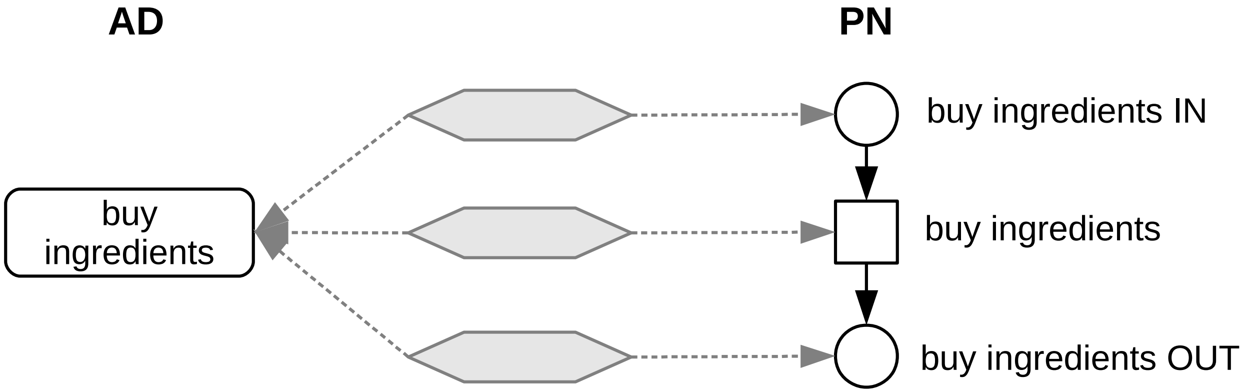
:Place

Intuitive Relation vs TGG Rule



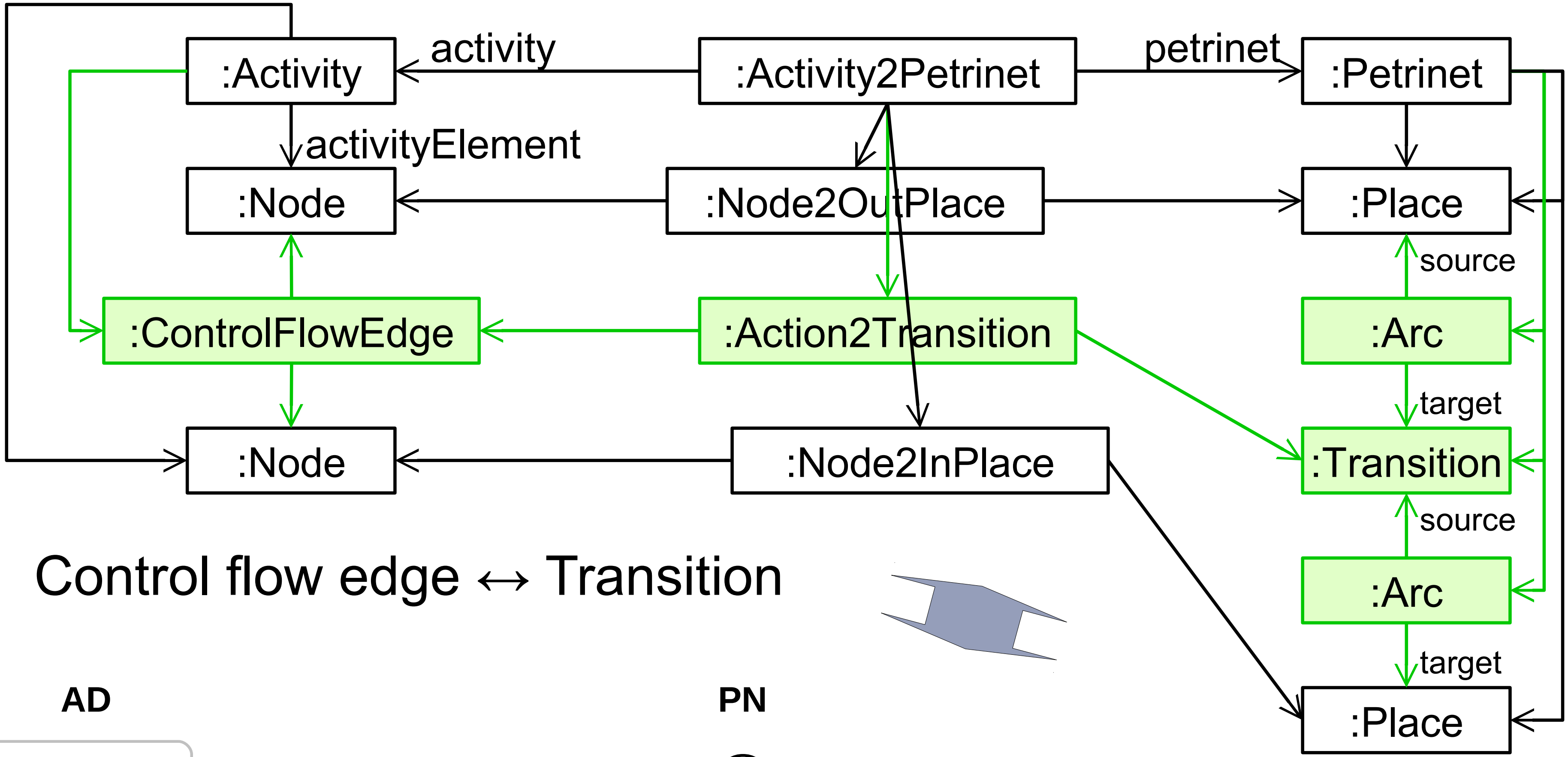
- Action node ↔ Transition



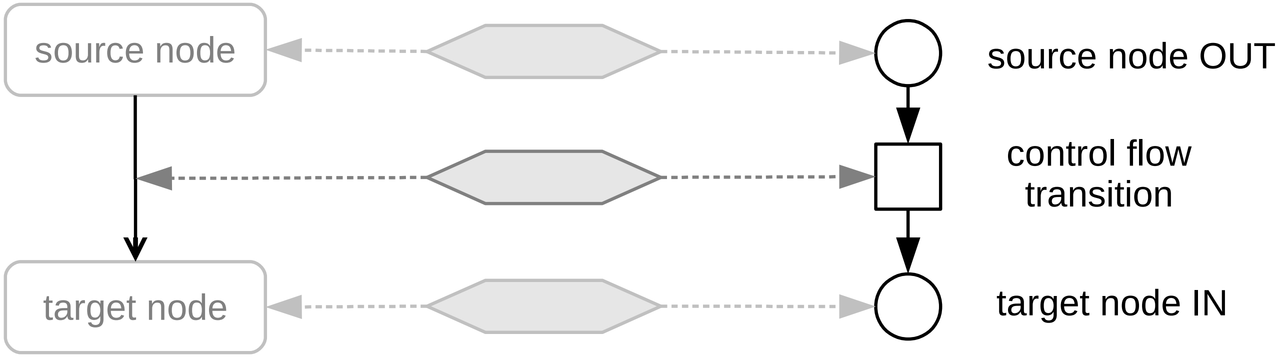


(ommitting some edge labels for space reasons)

Intuitive Relation vs TGG Rule



- Control flow edge ↔ Transition



(ommitting some edge labels for space reasons)

Application Scenarios

- TGGs can be used to produce valid triple graphs

Application Scenarios

- TGGs can be used to produce valid triple graphs
  - but that alone is not very useful...

Application Scenarios

- TGGs can be used to produce valid triple graphs
  - but that alone is not very useful...
- But, we can **operationalize** them for different useful purposes (**application scenarios**)

Application Scenarios

- TGGs can be used to produce valid triple graphs
  - but that alone is not very useful...
- But, we can **operationalize** them for different useful purposes (**application scenarios**)
  - transforming a given source model into a target model

Application Scenarios

- TGGs can be used to produce valid triple graphs
  - but that alone is not very useful...
- But, we can **operationalize** them for different useful purposes (**application scenarios**)
  - transforming a given source model into a target model
  - transforming a given target model back into a source model

Application Scenarios

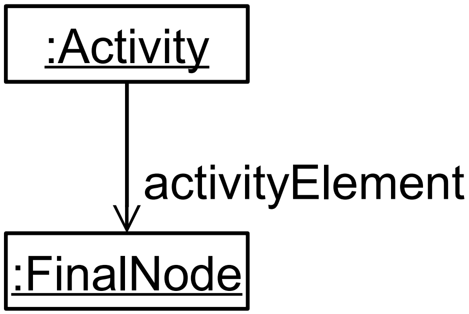
- TGGs can be used to produce valid triple graphs
  - but that alone is not very useful...
- But, we can **operationalize** them for different useful purposes (**application scenarios**)
  - transforming a given source model into a target model
  - transforming a given target model back into a source model
  - given a source and a target model, create the correspondence model to check whether they are valid corresponding models

Application Scenarios

- TGGs can be used to produce valid triple graphs
  - but that alone is not very useful...
- But, we can **operationalize** them for different useful purposes (**application scenarios**)
  - transforming a given source model into a target model
  - transforming a given target model back into a source model
  - given a source and a target model, create the correspondence model to check whether they are valid corresponding models
  - synchronize given source and target models as changes happen in the source or target model

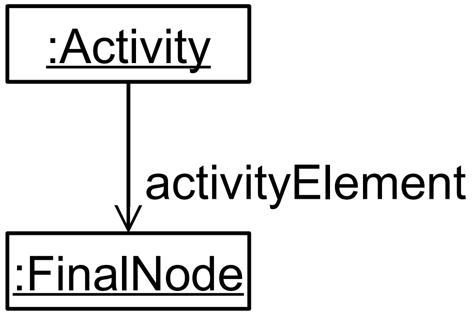
Forward Transformation Scenario

- Given is a source model



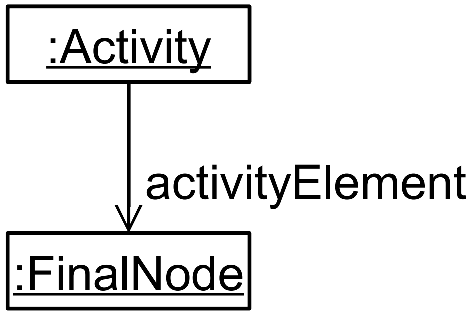
Forward Transformation Scenario

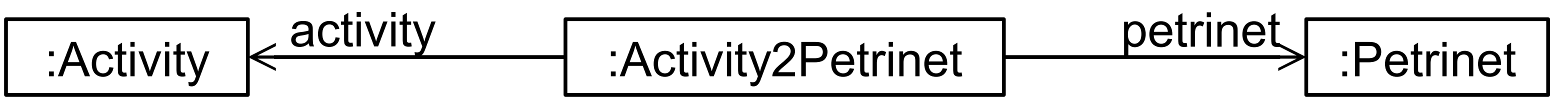
- Given is a source model
- First, we *apply* the axiom:



Forward Transformation Scenario

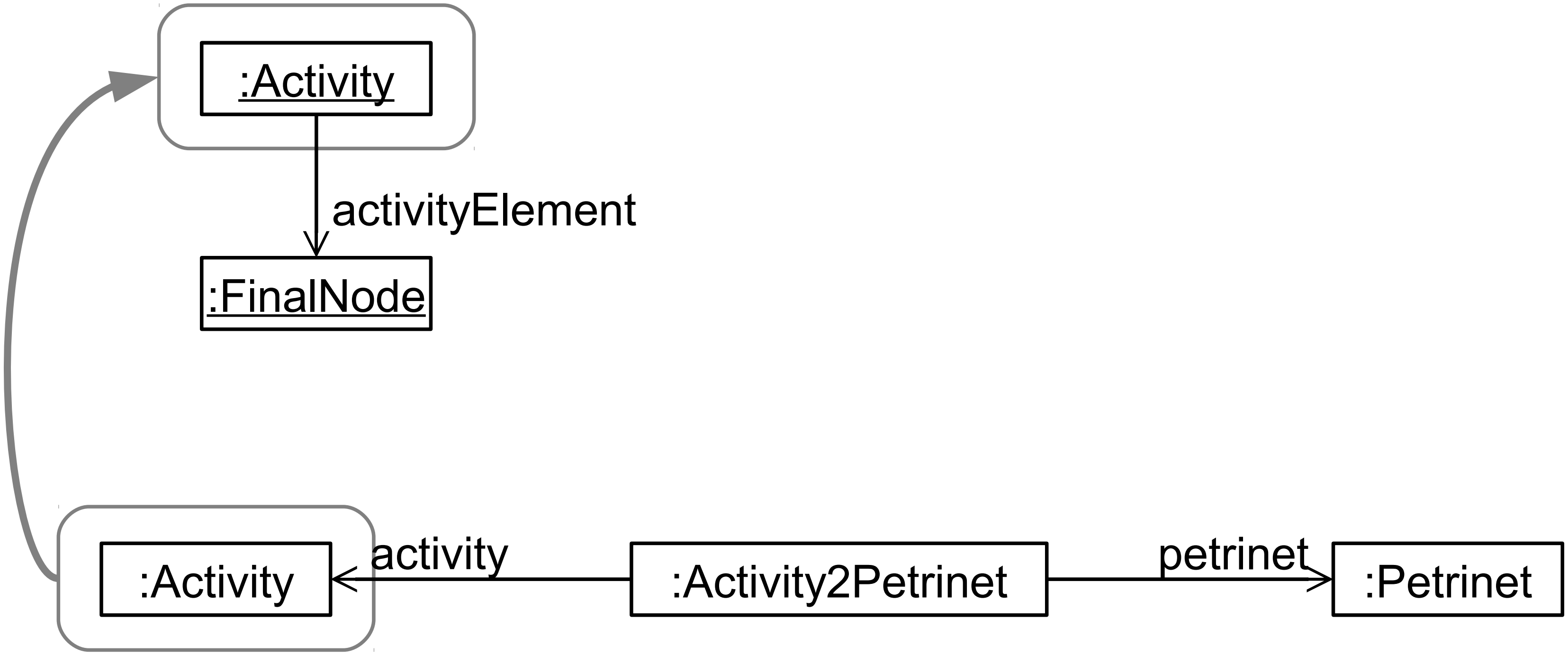
- Given is a source model
- First, we *apply* the axiom:





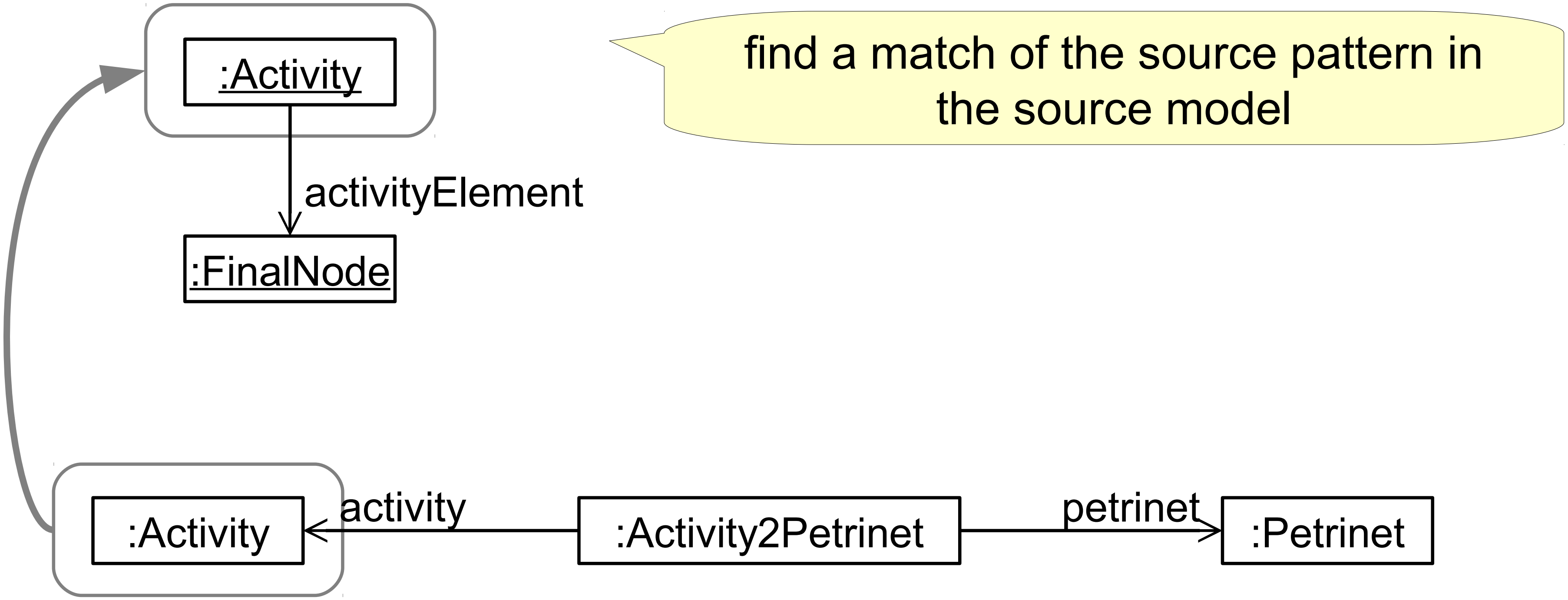
Forward Transformation Scenario

- Given is a source model
- First, we *apply* the axiom:



Forward Transformation Scenario

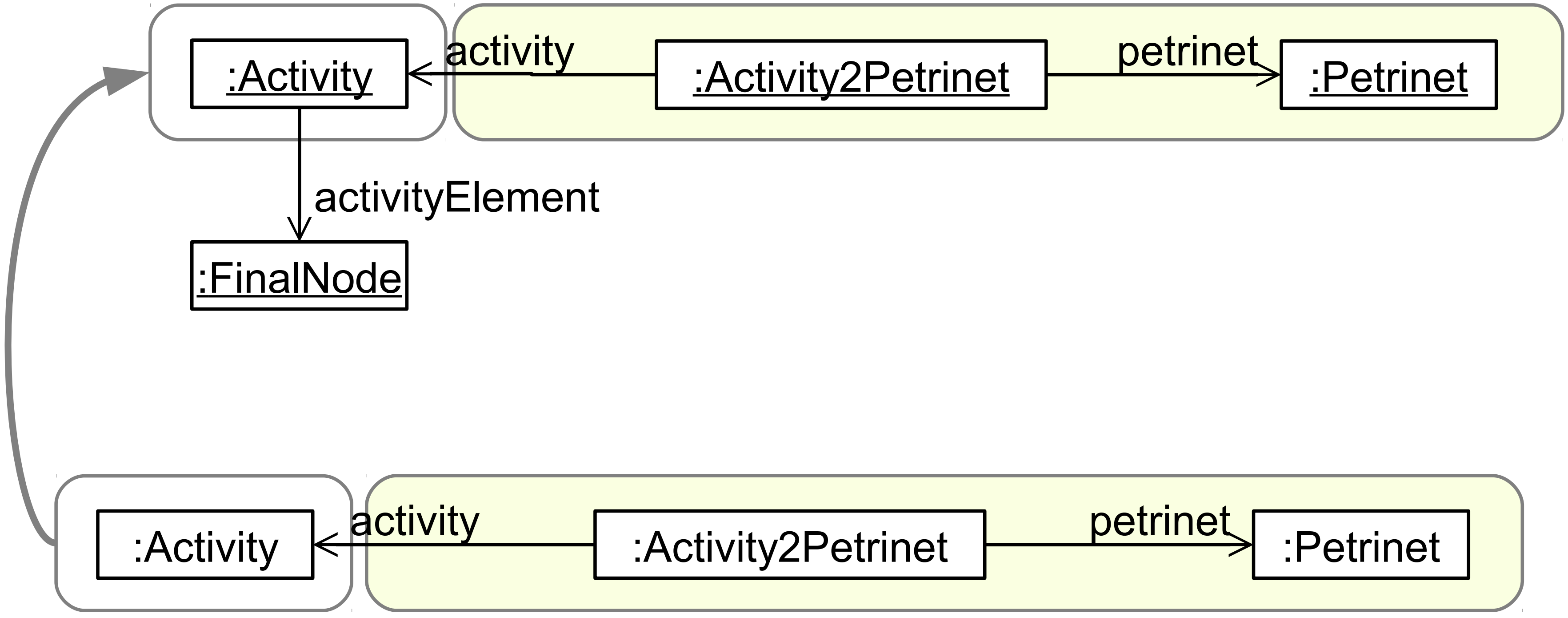
- Given is a source model
- First, we *apply* the axiom:



find a match of the source pattern in the source model

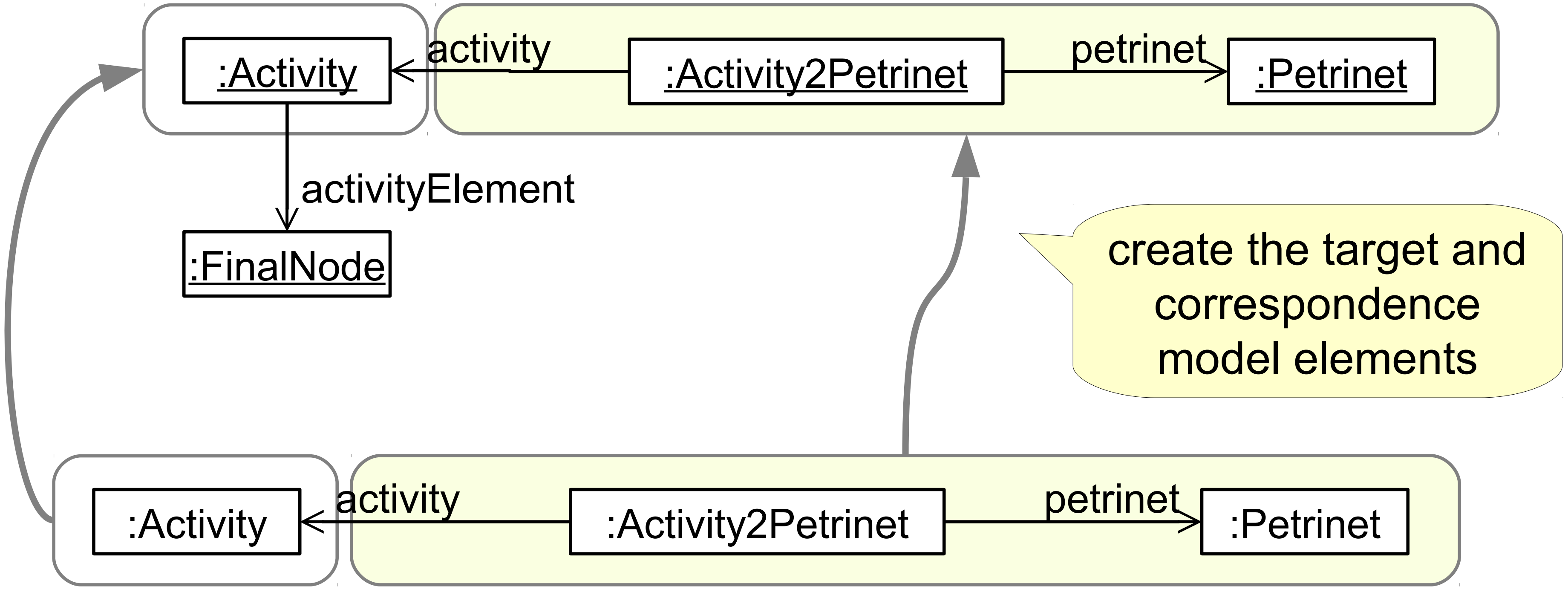
Forward Transformation Scenario

- Given is a source model
- First, we *apply* the axiom:



Forward Transformation Scenario

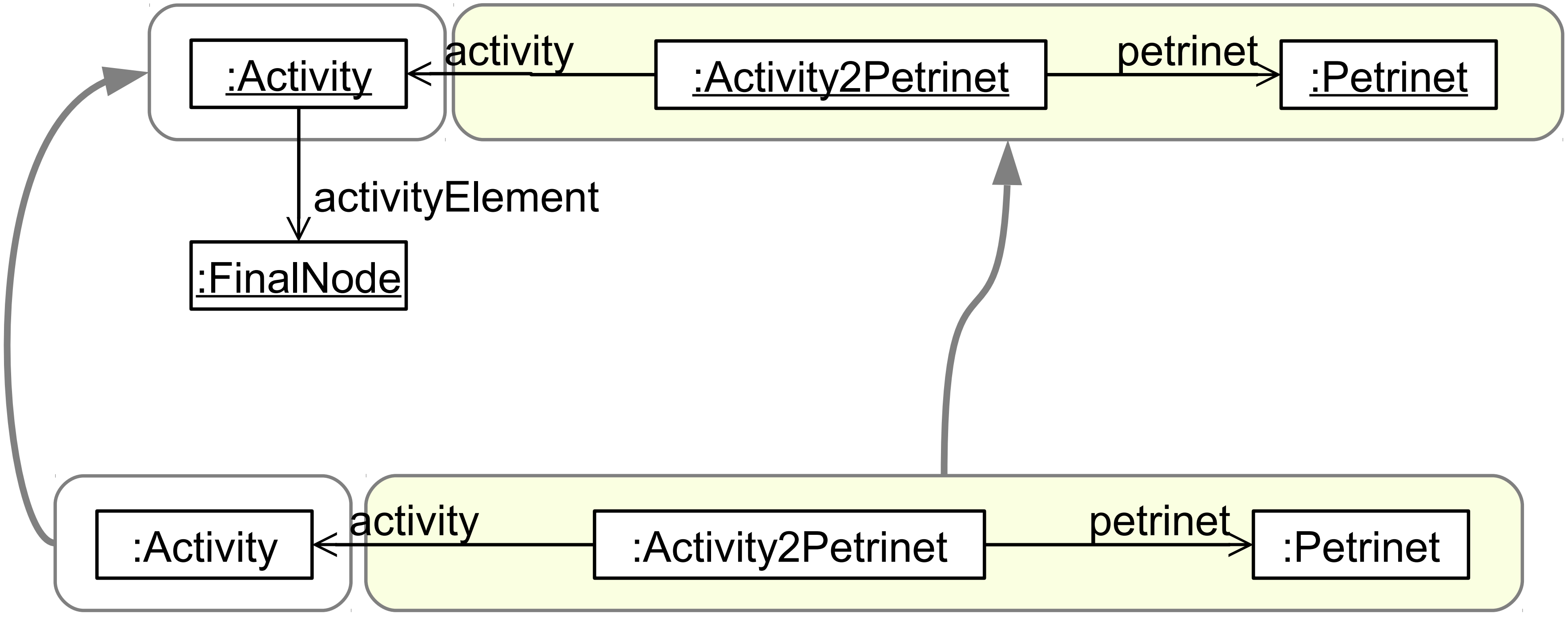
- Given is a source model
- First, we *apply* the axiom:



create the target and correspondence model elements

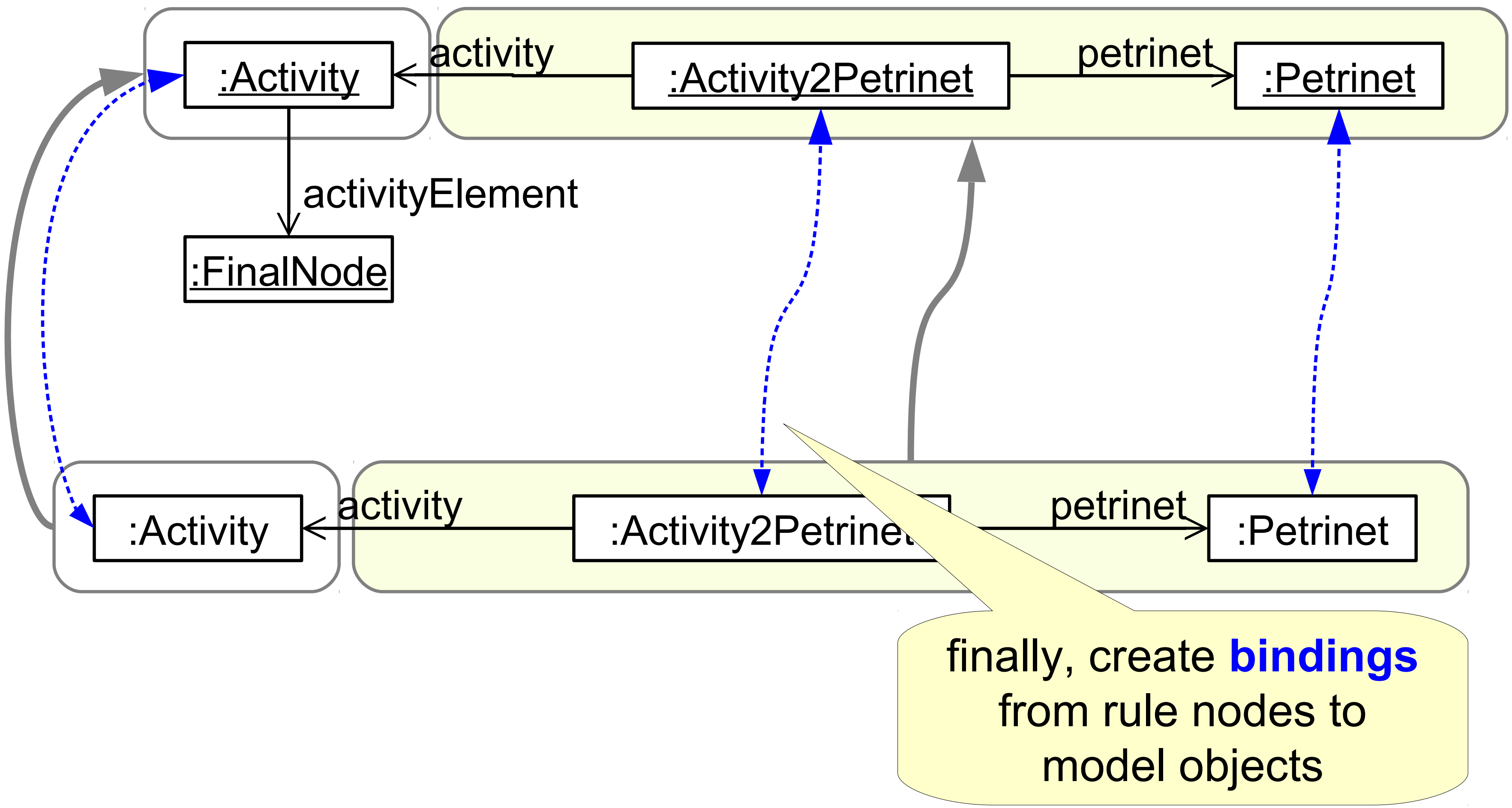
Forward Transformation Scenario

- Given is a source model
- First, we *apply* the axiom:



Forward Transformation Scenario

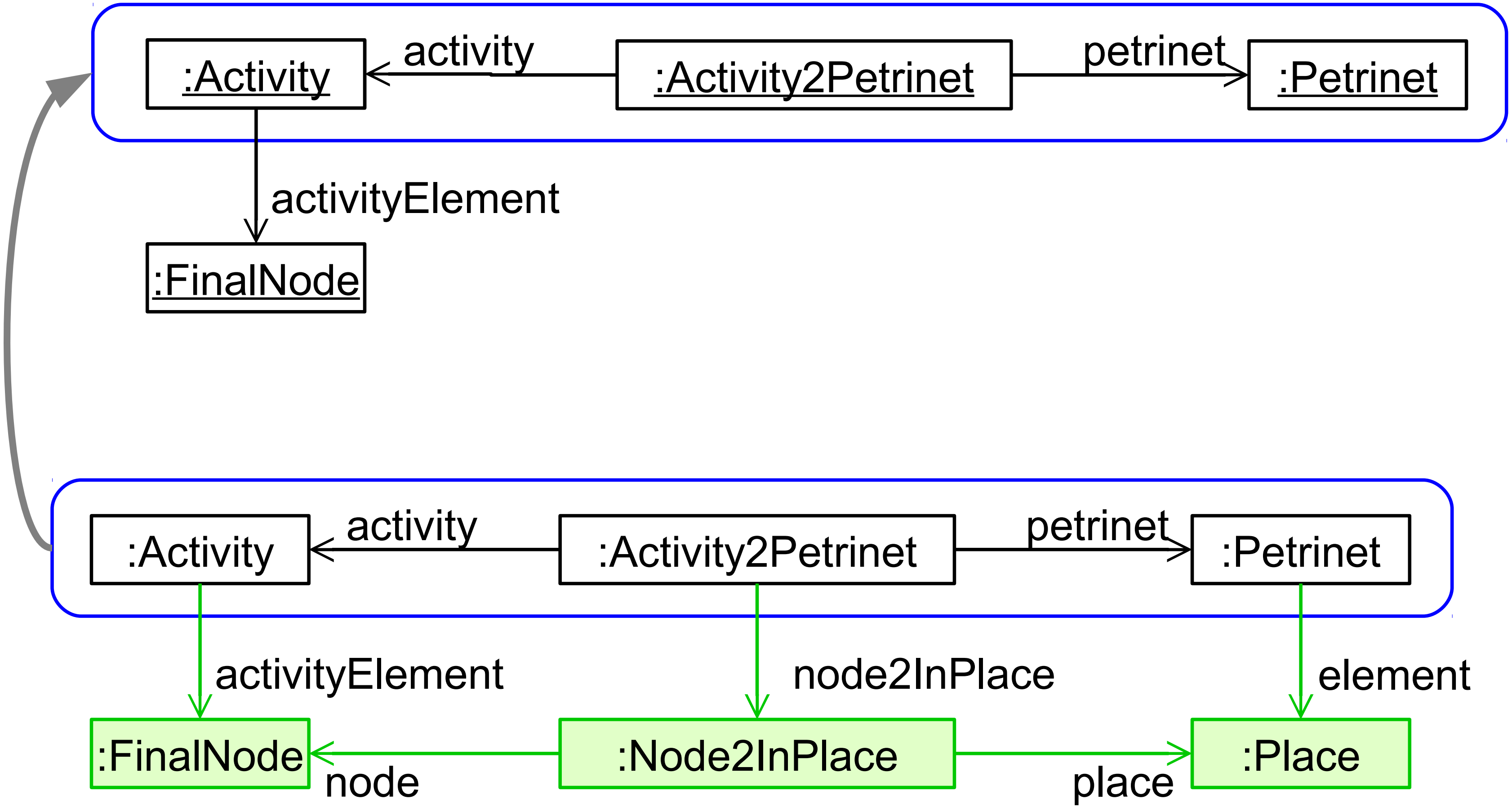
- Given is a source model
- First, we *apply* the axiom:



finally, create **bindings** from rule nodes to model objects

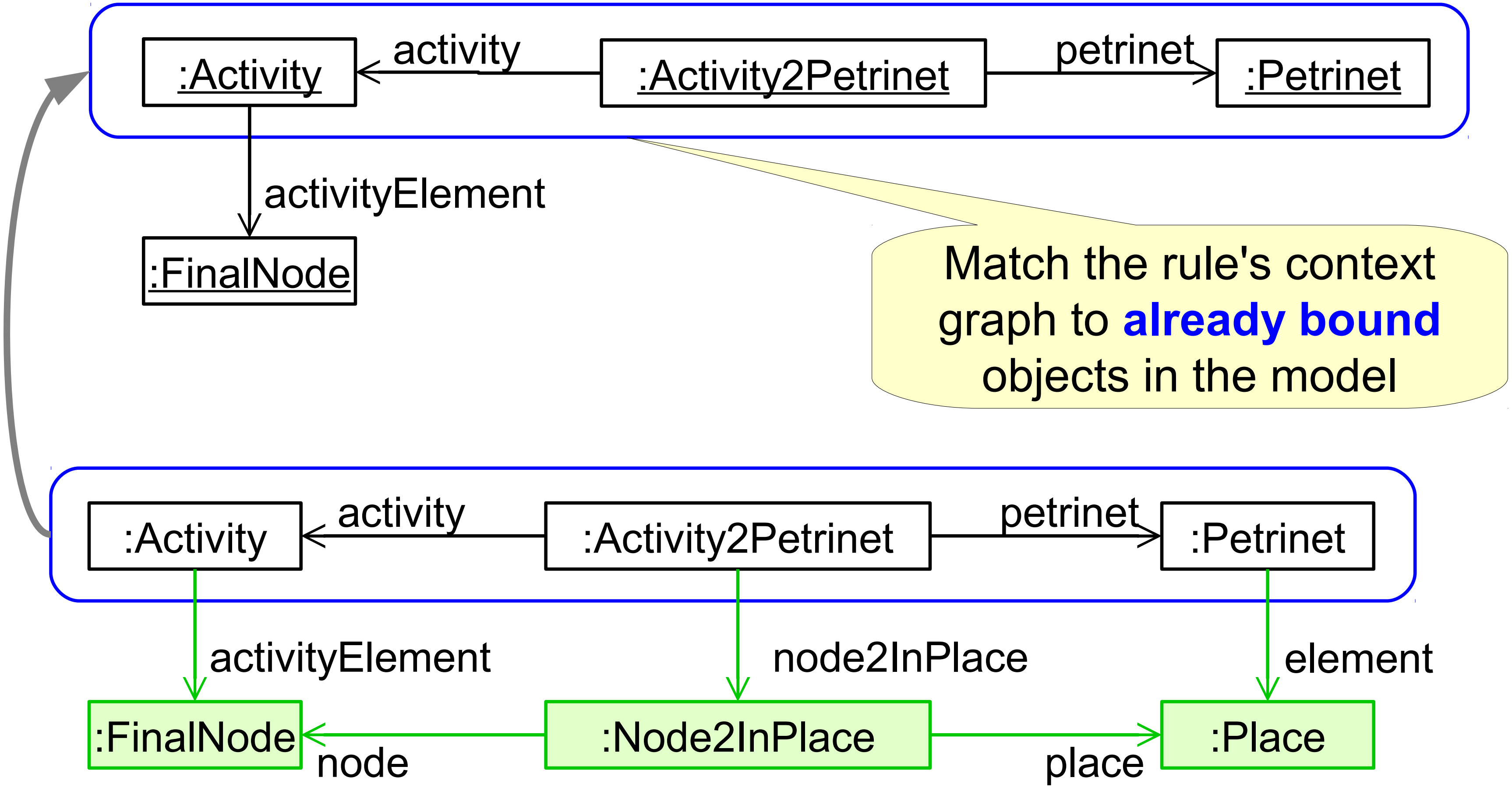
Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



Forward Transformation Scenario

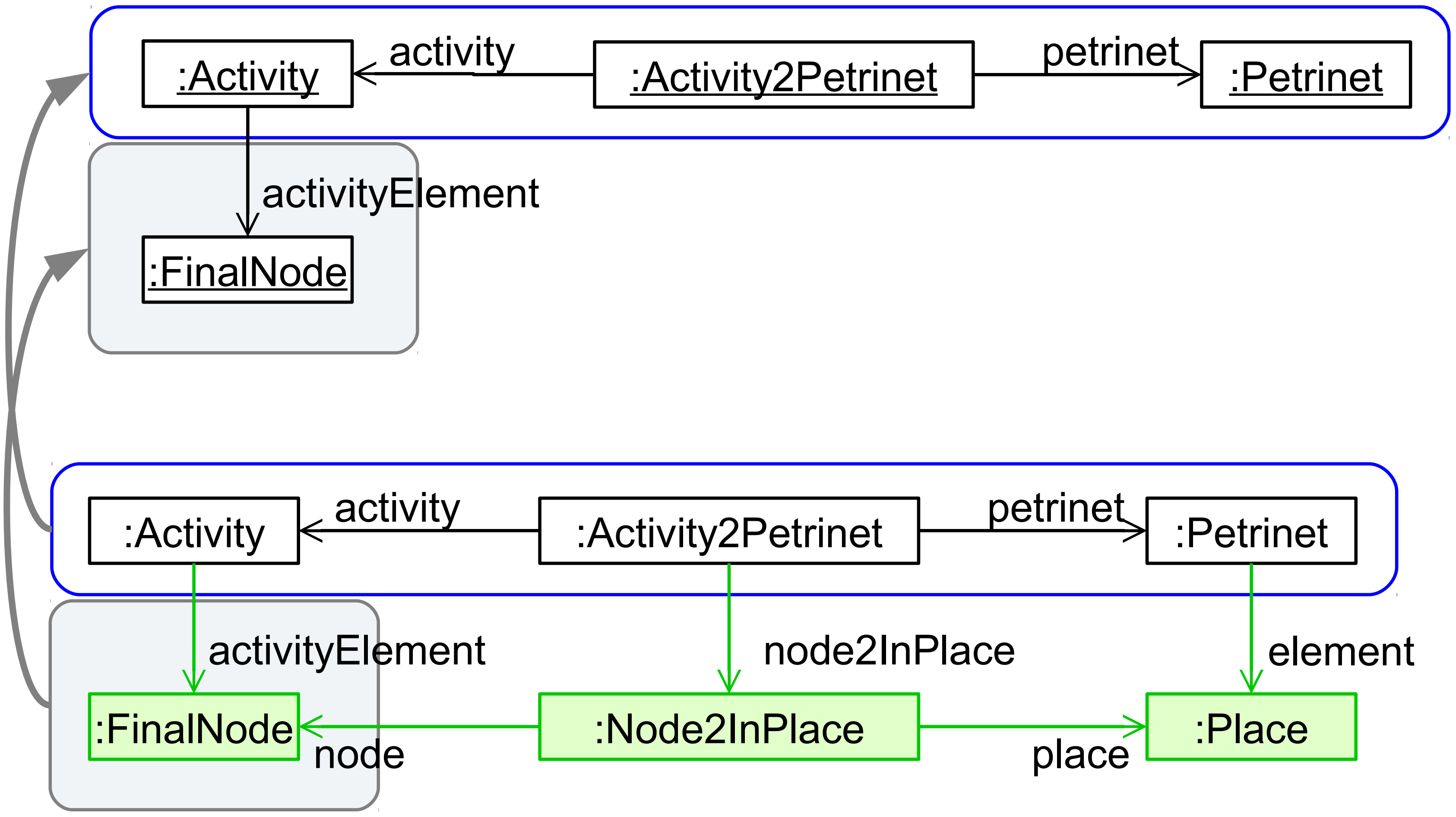
- Then, find TGG rule that can be applied as follows:



Match the rule's context graph to **already bound** objects in the model

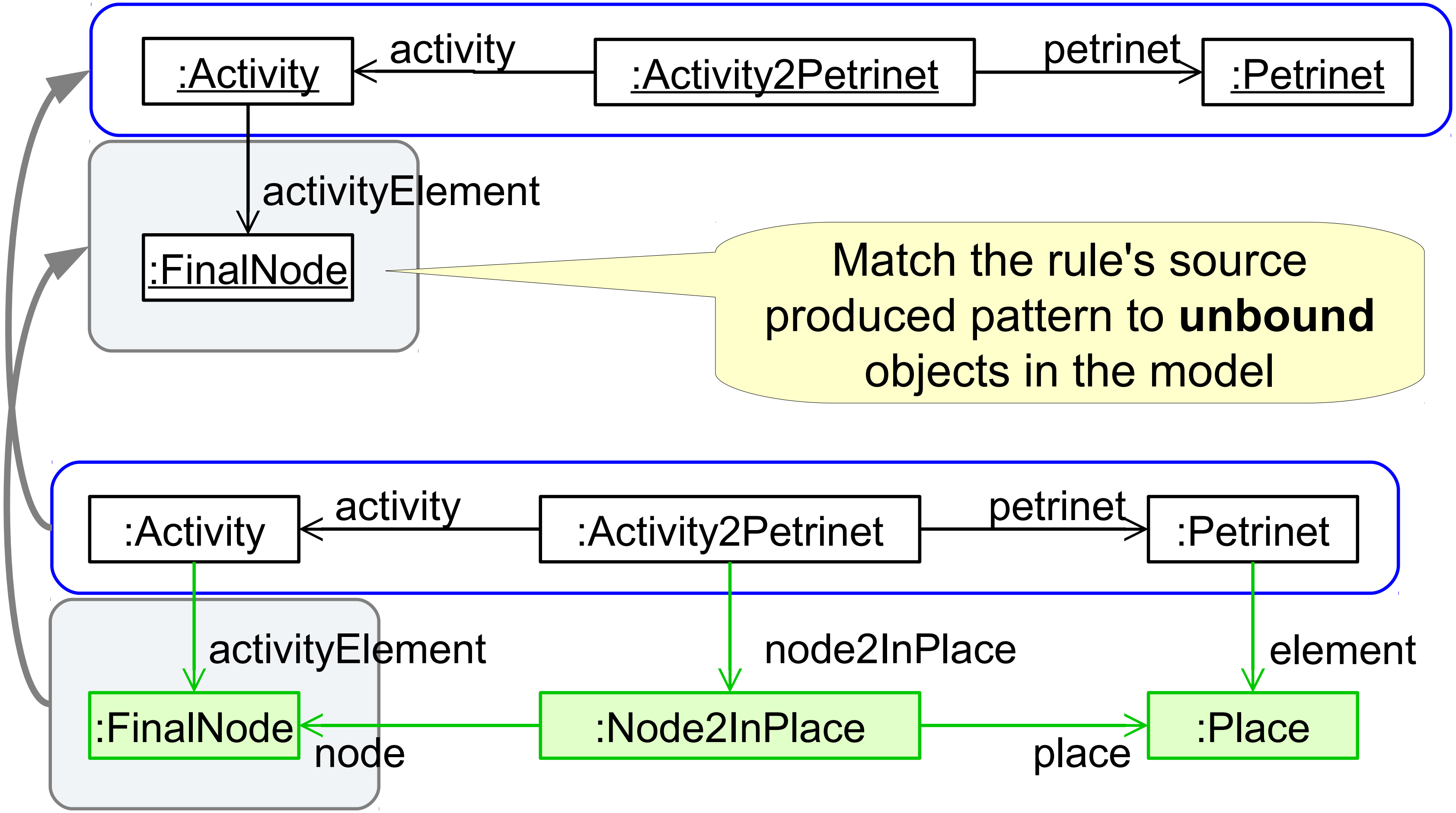
Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



Forward Transformation Scenario

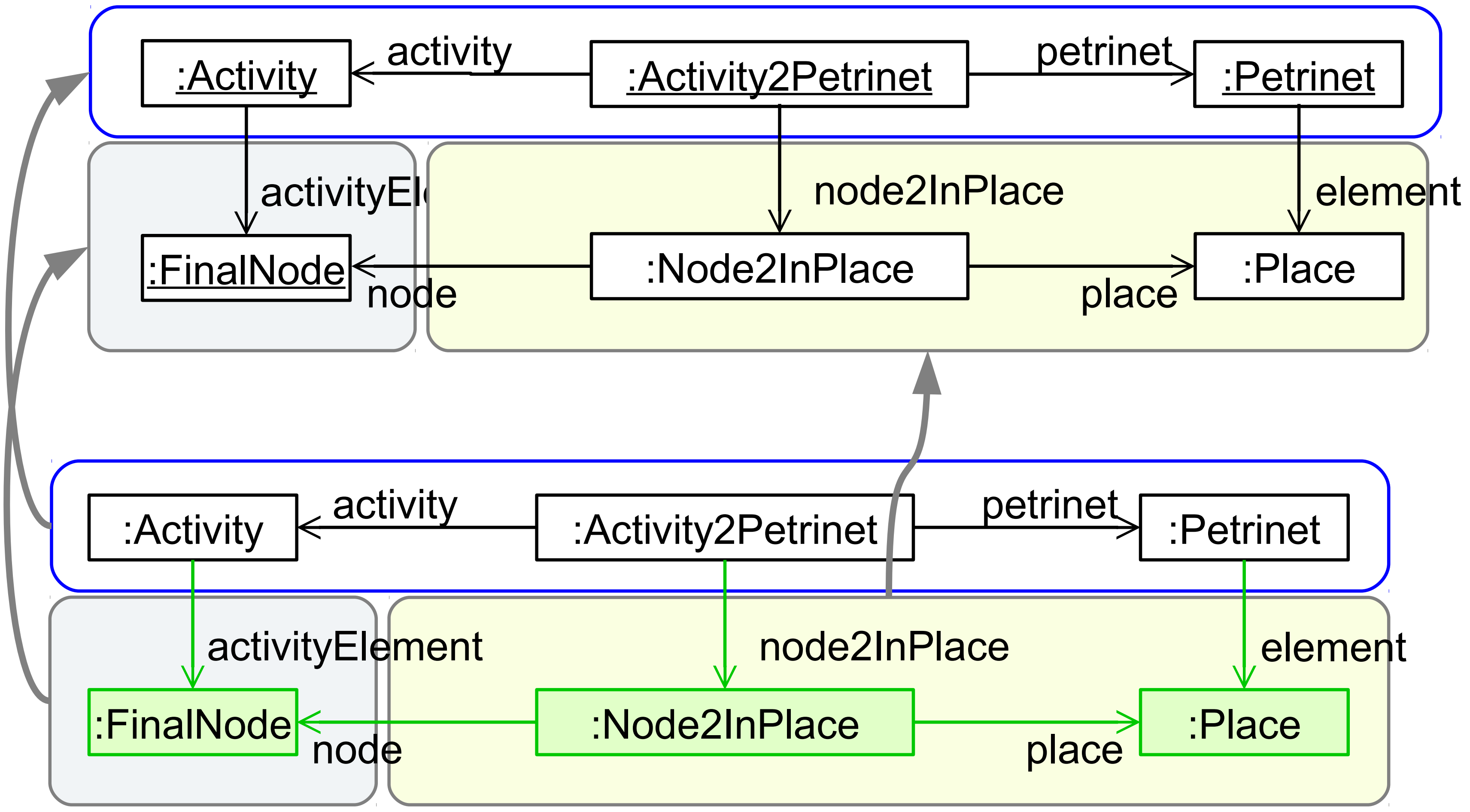
- Then, find TGG rule that can be applied as follows:



Match the rule's source produced pattern to **unbound** objects in the model

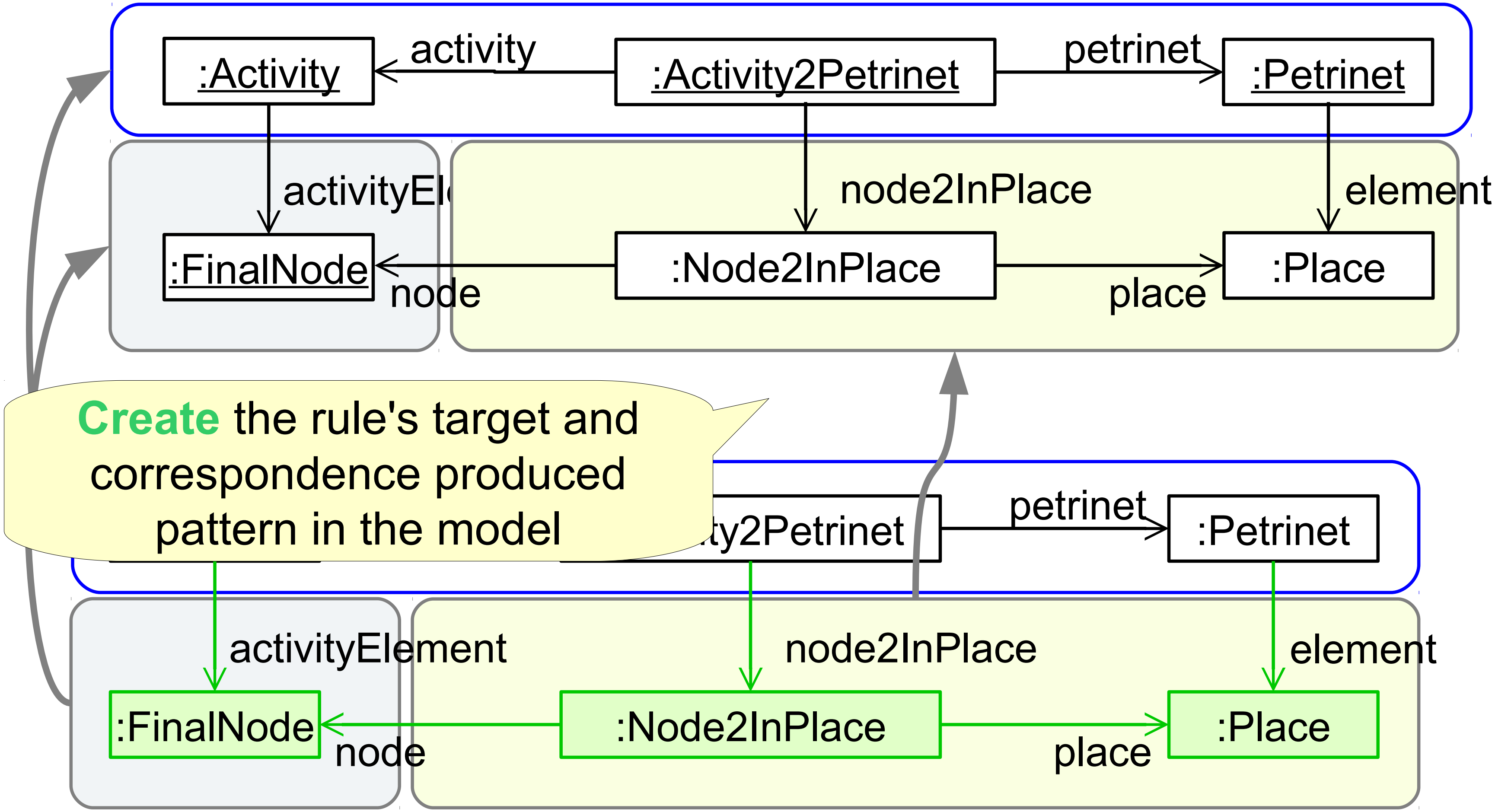
Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



Forward Transformation Scenario

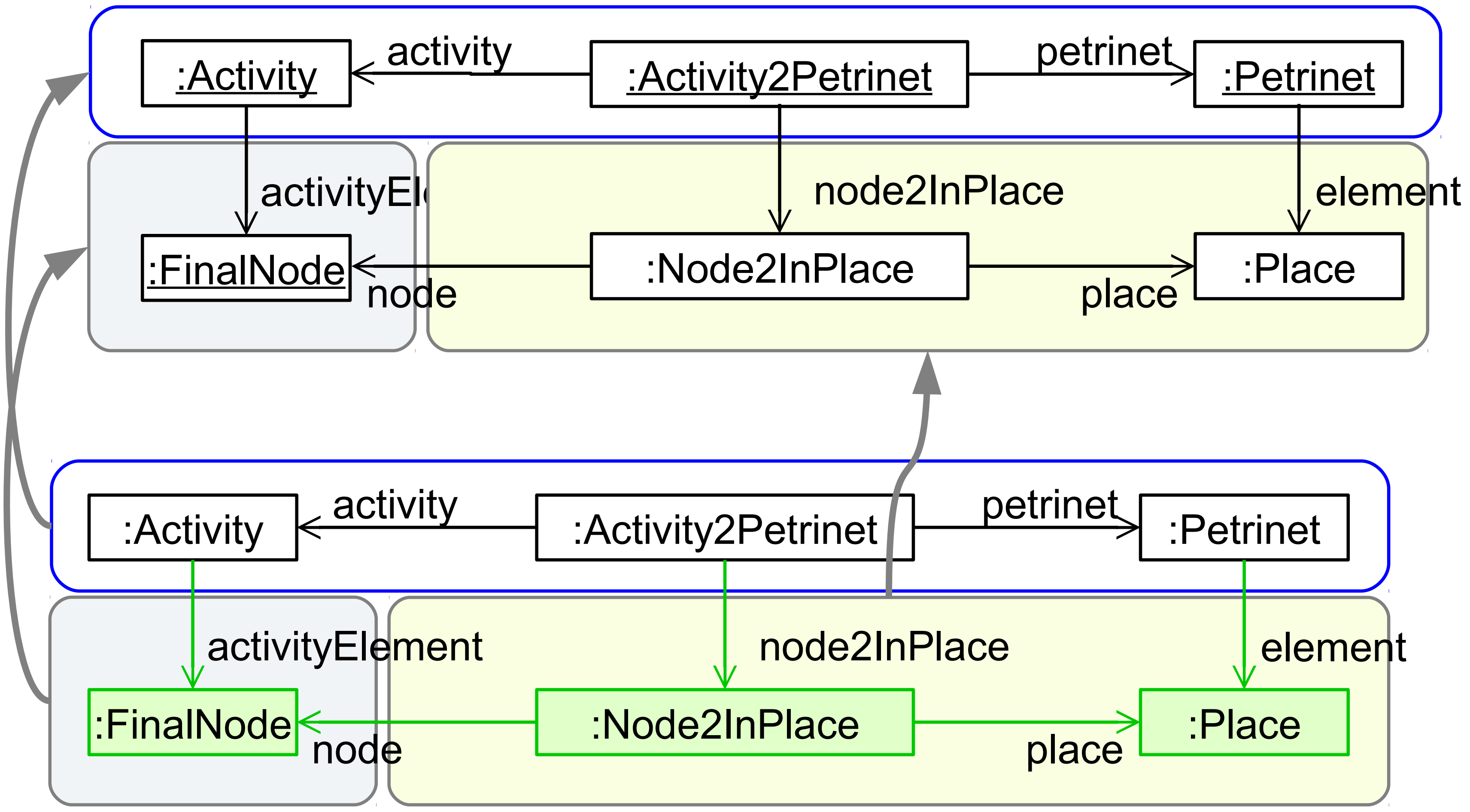
- Then, find TGG rule that can be applied as follows:



Create the rule's target and correspondence produced pattern in the model

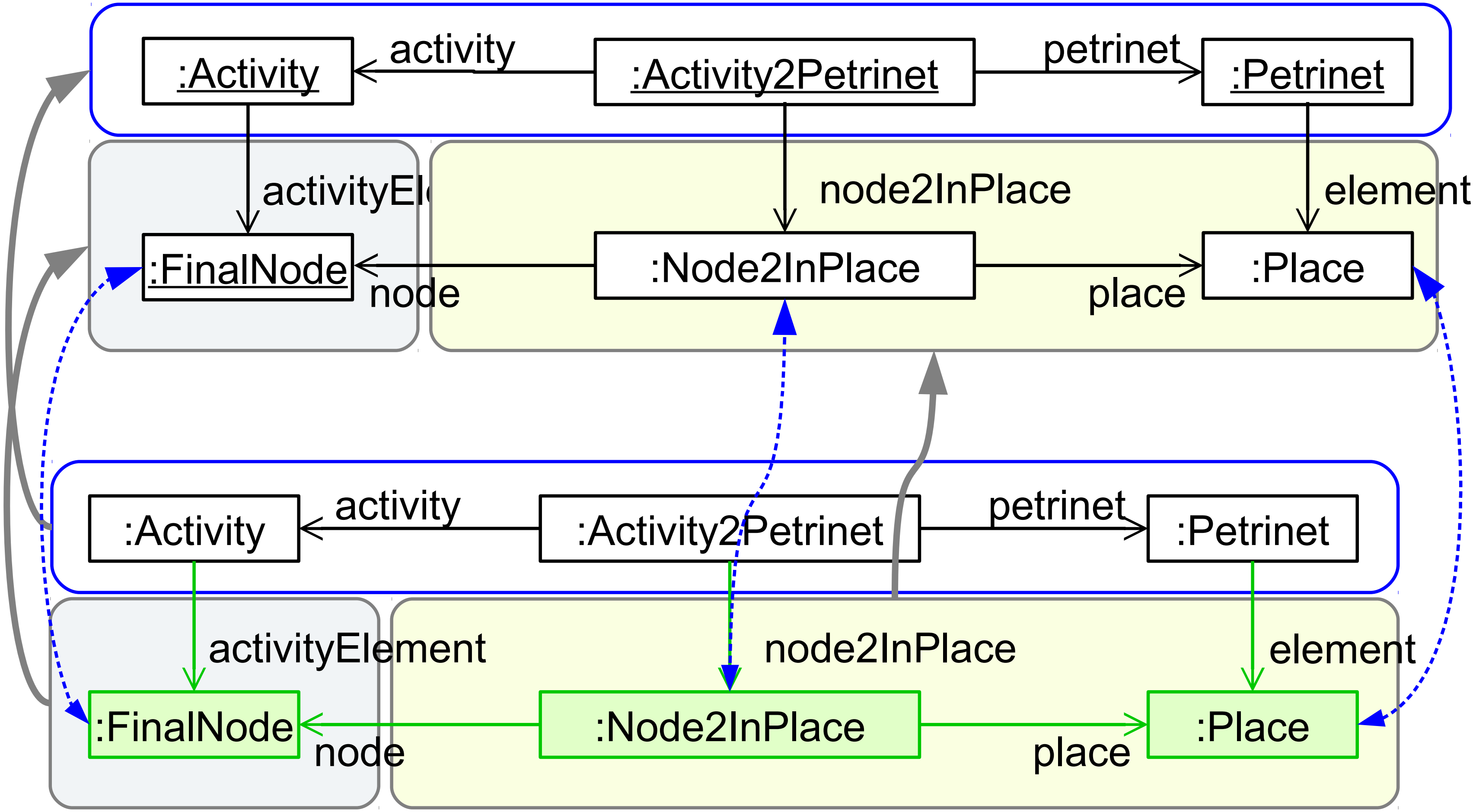
Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



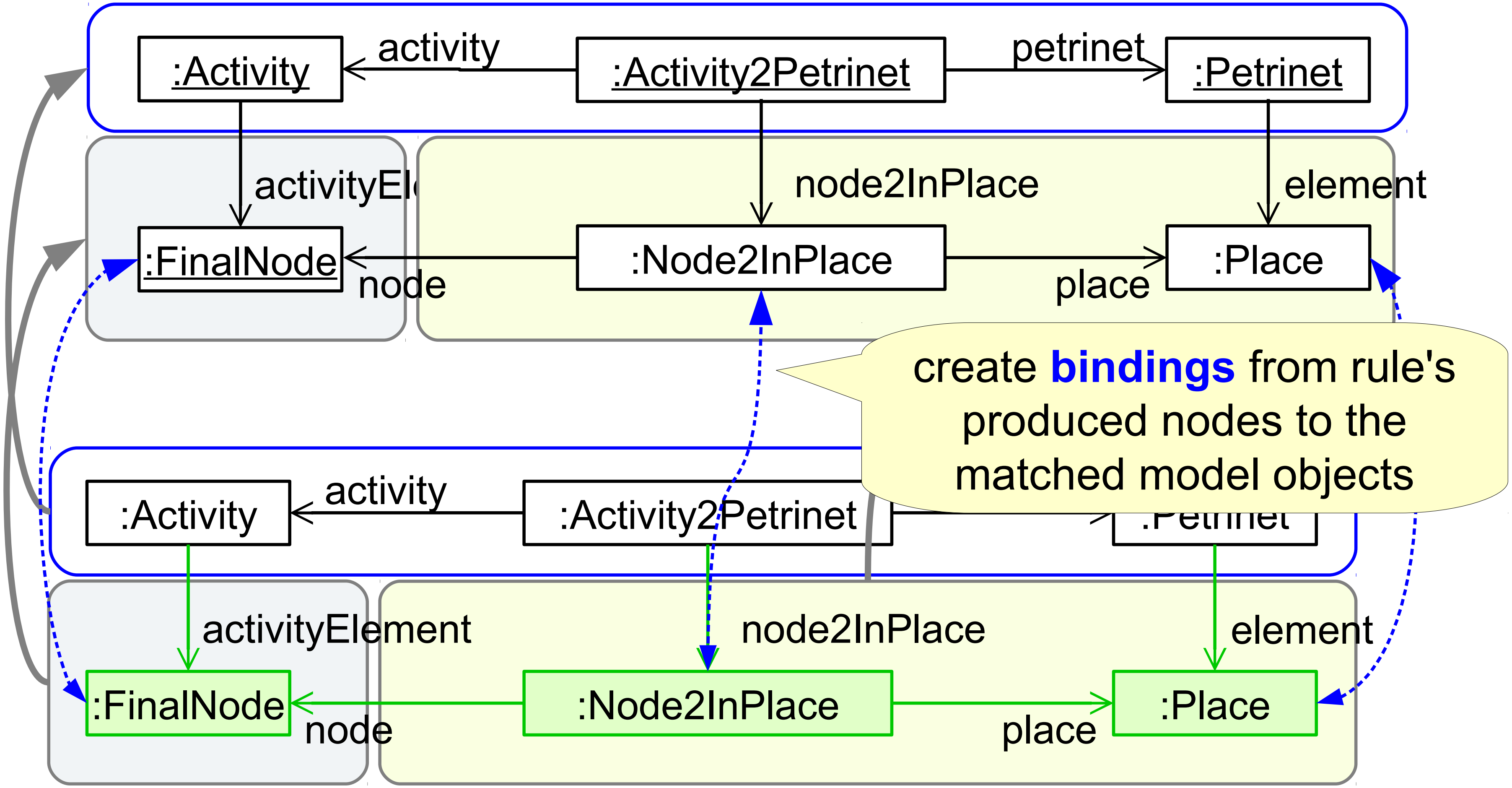
Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



Forward Transformation Scenario

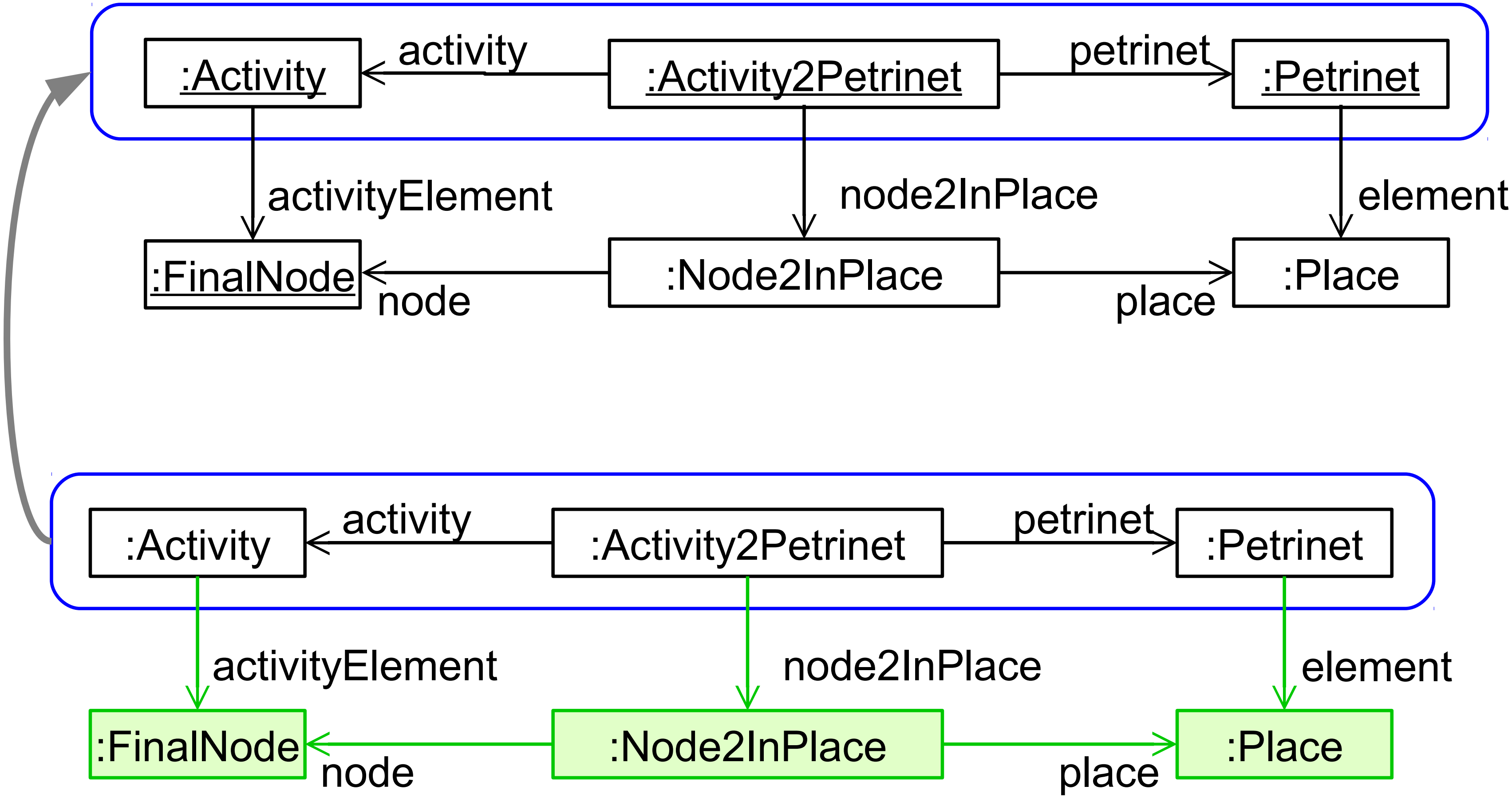
- Then, find TGG rule that can be applied as follows:



create **bindings** from rule's produced nodes to the matched model objects

Forward Transformation Scenario

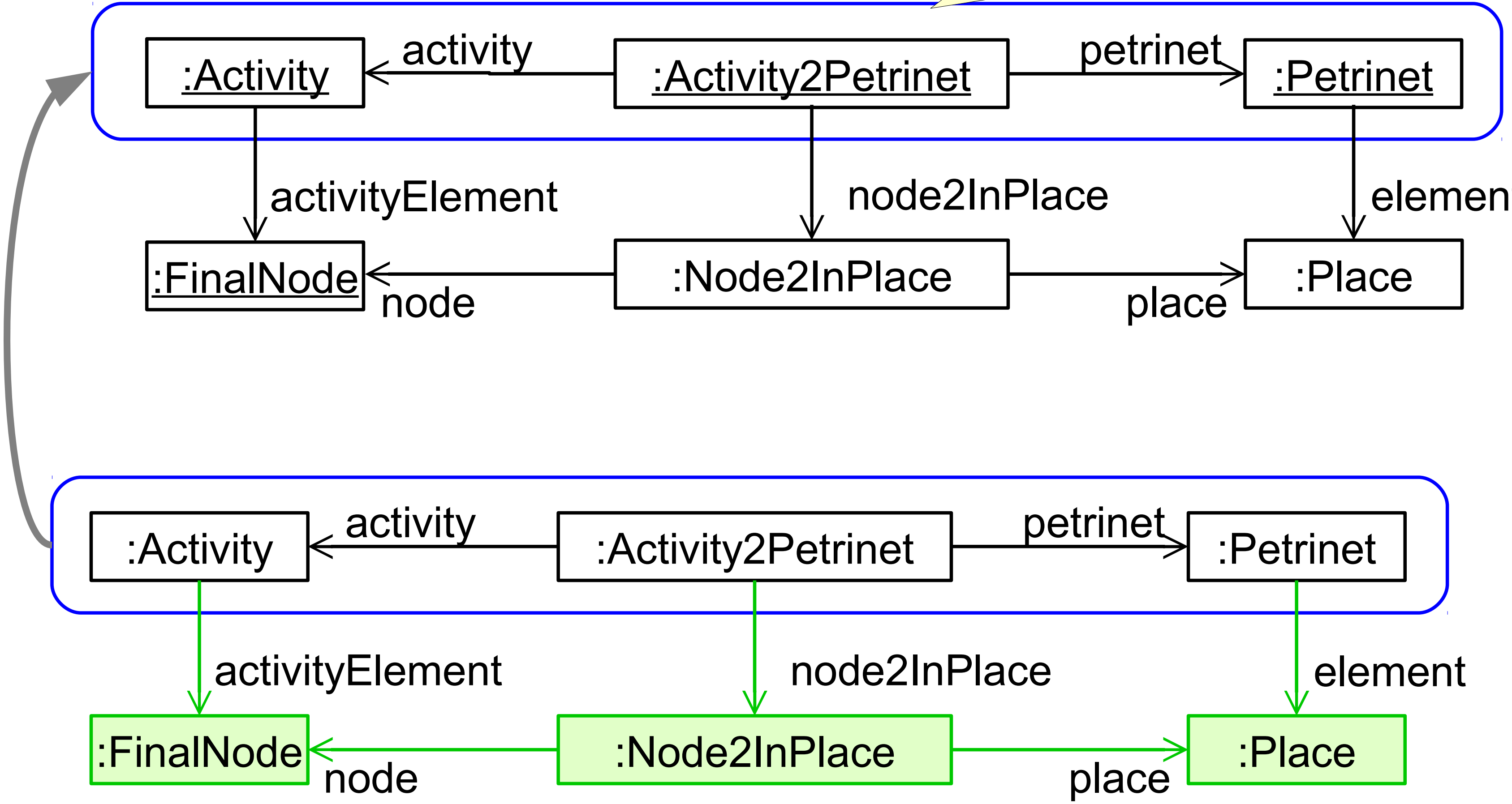
- Can we apply the rule again?



Forward Transformation Scenario

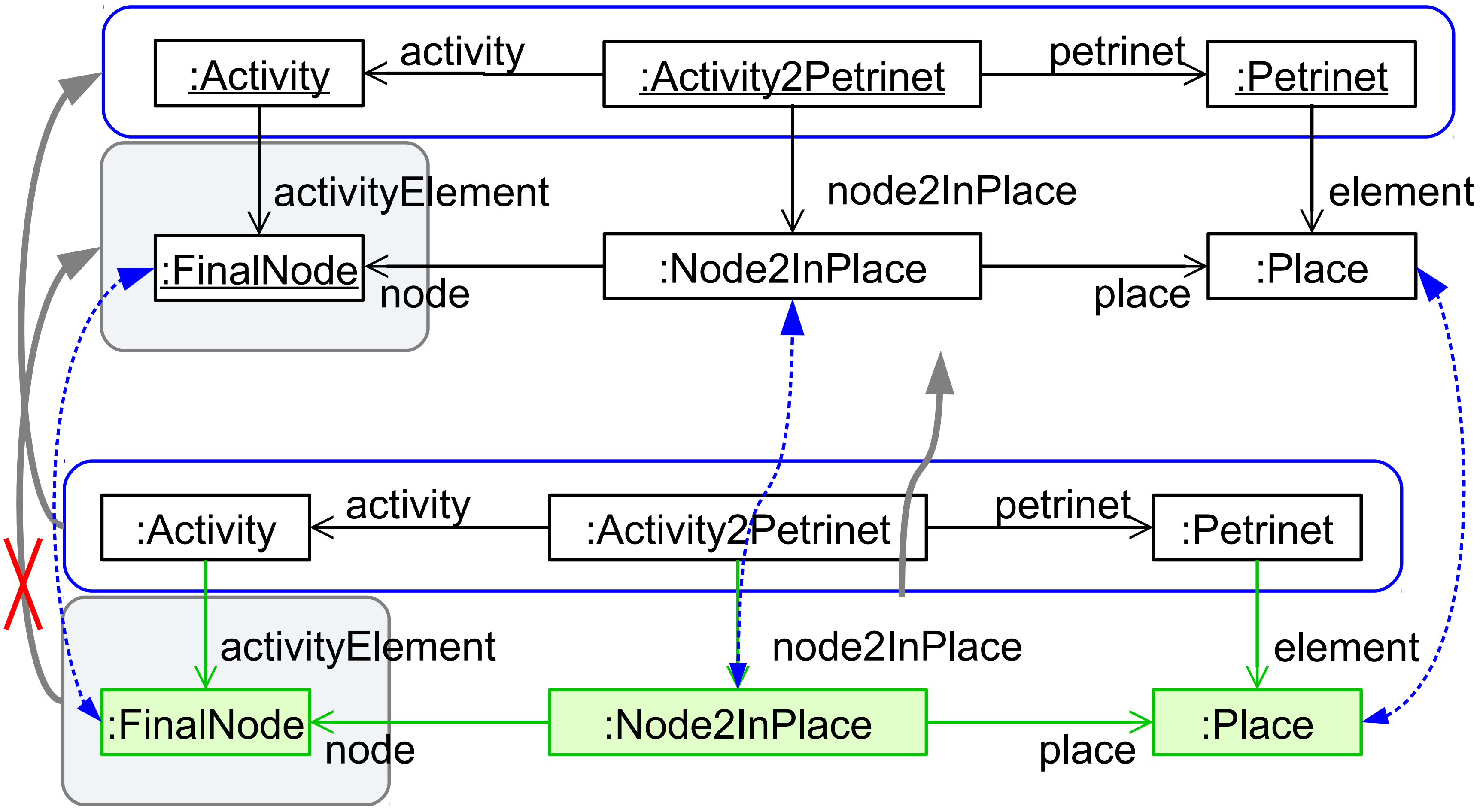
- Can we apply the rule again?

We can match the context graph pattern to already bound model elements, but...



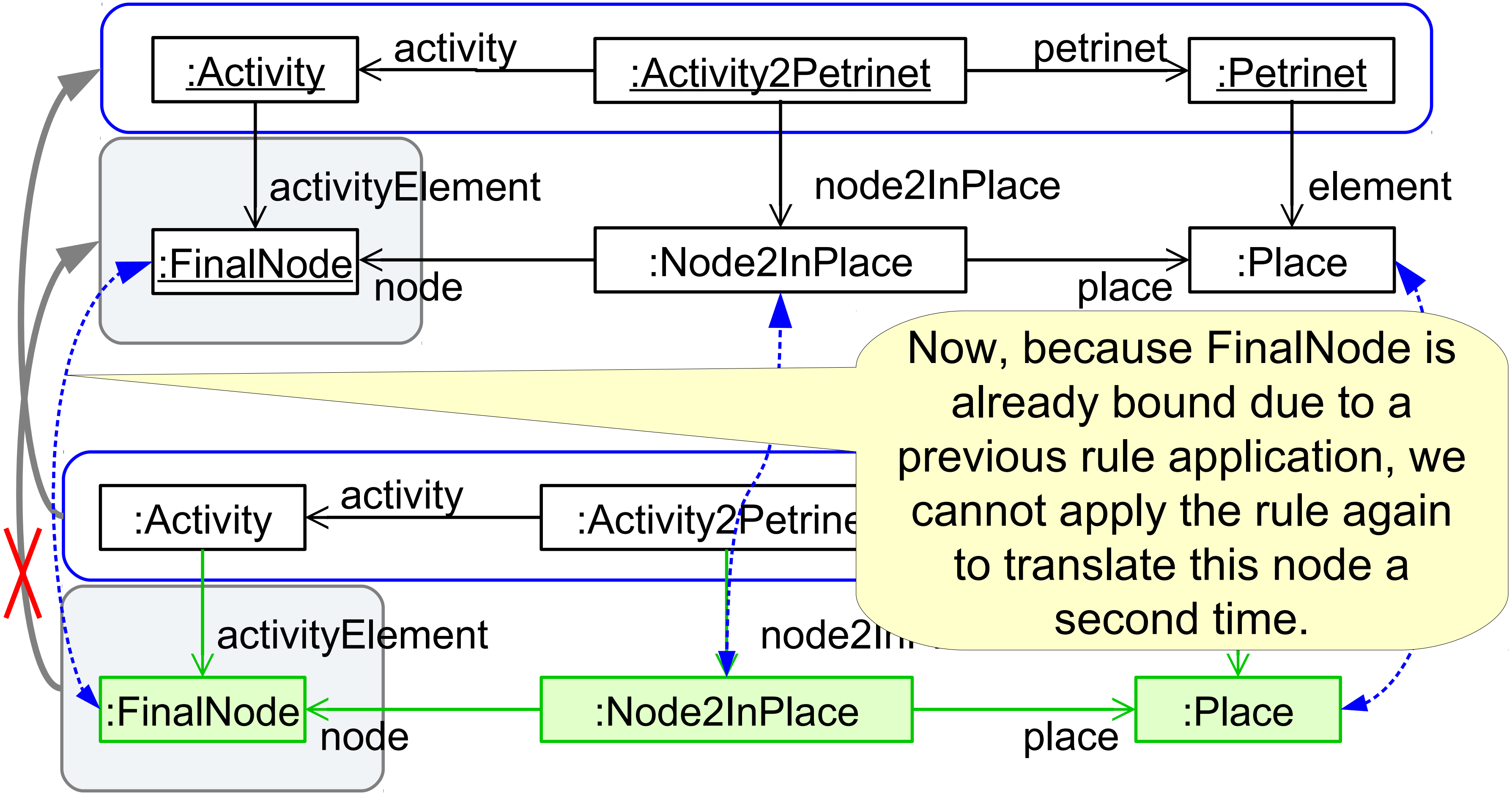
Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



Forward Transformation Scenario

- Then, find TGG rule that can be applied as follows:



Now, because FinalNode is already bound due to a previous rule application, we cannot apply the rule again to translate this node a second time.

TGG Extensions

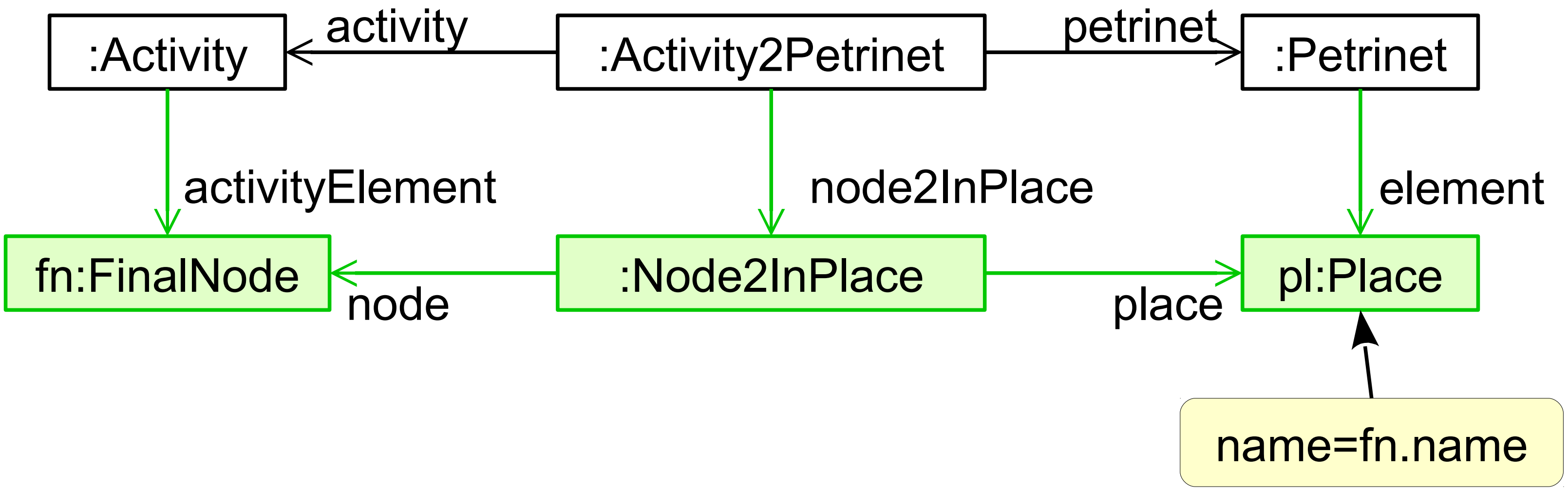
- Extensions of TGGs include:
  - support for reuse in rules: rule inheritance
  - support for attribute constraints

TGG Extensions

- Extensions of TGGs include:
  - support for reuse in rules: rule inheritance
  - support for attribute constraints
- Example: Attribute constraints

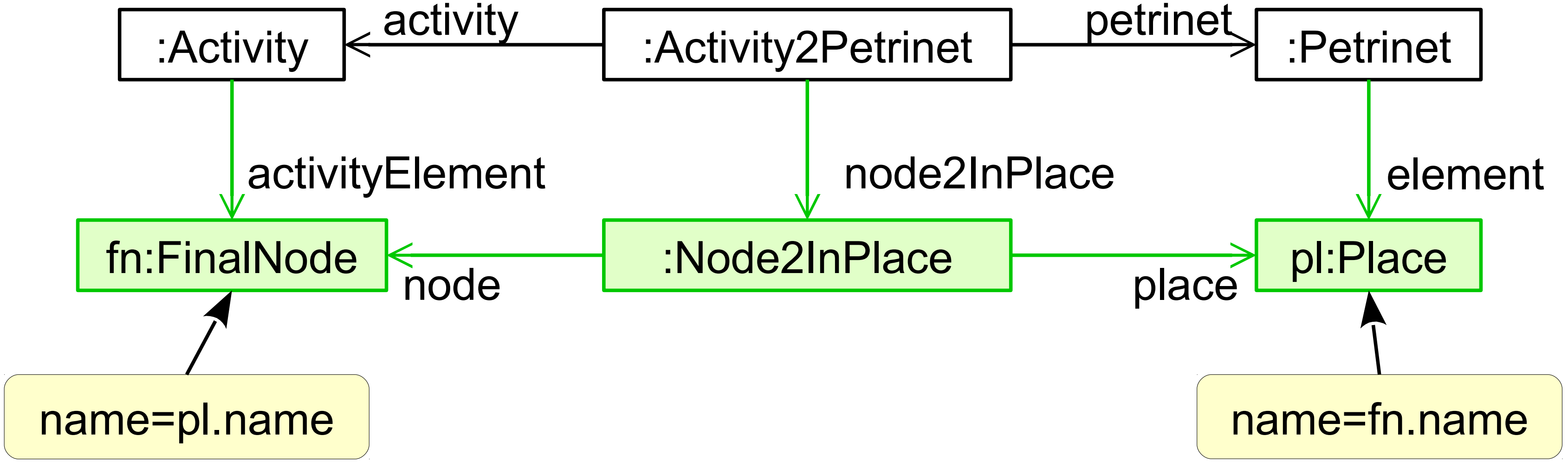
TGG Extensions

- Extensions of TGGs include:
  - support for reuse in rules: rule inheritance
  - support for attribute constraints
- Example: Attribute constraints



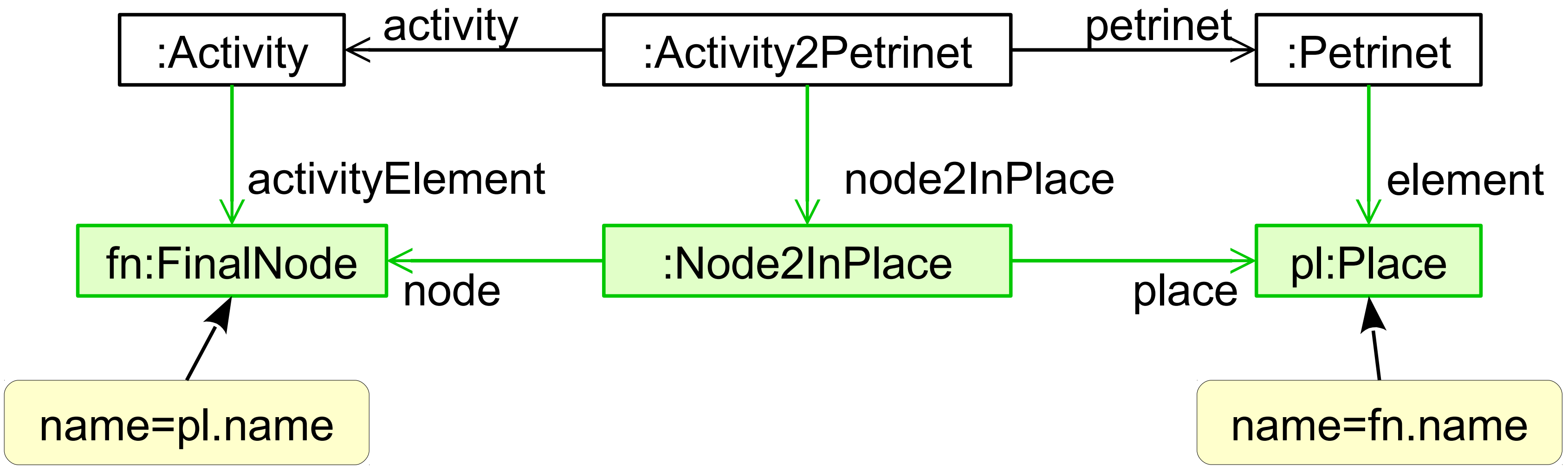
TGG Extensions

- Extensions of TGGs include:
  - support for reuse in rules: rule inheritance
  - support for attribute constraints
- Example: Attribute constraints



TGG Extensions

- Extensions of TGGs include:
  - support for reuse in rules: rule inheritance
  - support for attribute constraints
- Example: Attribute constraints



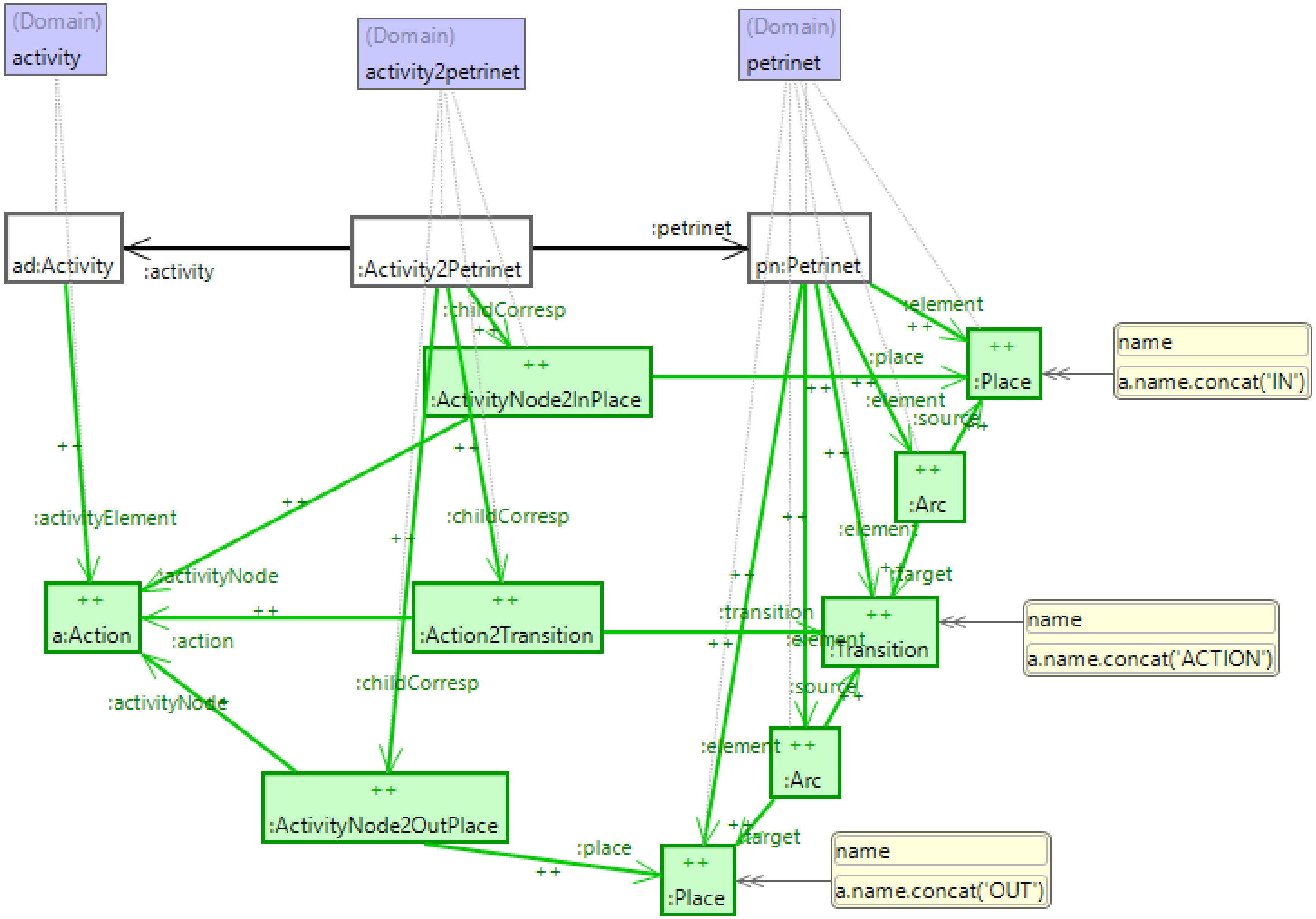
bidirectional assignments required for a bidirectional execution of the transformation

Tool Support

- Different academic and industrial tools exist:
  - TGG-Interpreter
  - eMOFLON
  - Fujaba
  - MDELab
  - EMorF

TGG Interpreter

- Screenshot of a TGG rule diagram in the editor of the TGG Interpreter tool:



TGG Transformations

- **Advantages**

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation
- **Disadvantages**

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation
- **Disadvantages**
  - visual rules are nice as long as rules are not bigger than screen

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation
- **Disadvantages**
  - visual rules are nice as long as rules are not bigger than screen
  - **Debugging** is difficult

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation
- **Disadvantages**
  - visual rules are nice as long as rules are not bigger than screen
  - **Debugging** is difficult
    - Approaches exist, but still hard to find problems, especially when rule application is non-deterministic

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation
- **Disadvantages**
  - visual rules are nice as long as rules are not bigger than screen
  - **Debugging** is difficult
    - Approaches exist, but still hard to find problems, especially when rule application is non-deterministic
  - **Performance**: optimized engines exist, but rule matching is always potentially slower than an imperative program

TGG Transformations

- **Advantages**
  - **declarative**: model corresponding patterns instead of programming the exact transformation procedure
    - Equivalent imperative program often **significantly** more complex!
  - **visual representation** of the corresponding graph structures
    - enhances comprehension of the transformation
- **Disadvantages**
  - visual rules are nice as long as rules are not bigger than screen
  - **Debugging** is difficult
    - Approaches exist, but still hard to find problems, especially when rule application is non-deterministic
  - **Performance**: optimized engines exist, but rule matching is always potentially slower than an imperative program
  - Only works well if source and target models have a similar structure

5.5. Model-to-model transformation – Query/View/Transformation

QVT (Query/View/Transformations)

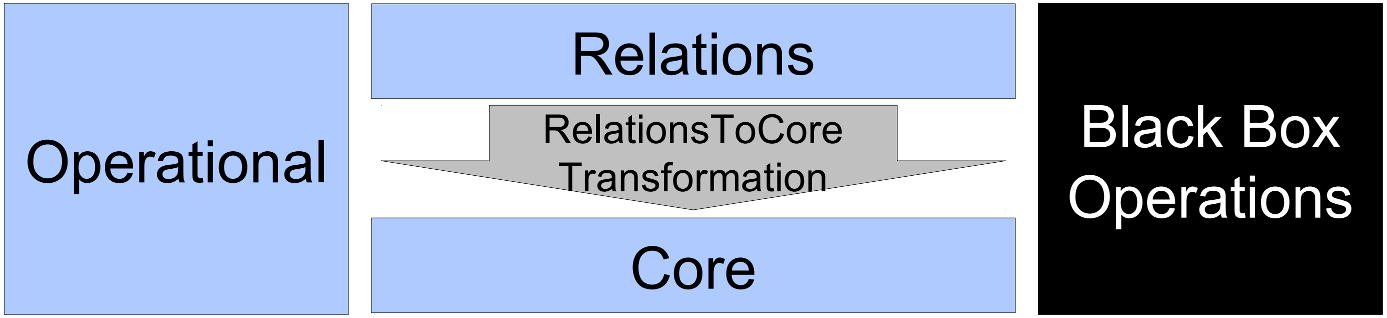
- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/

QVT (Query/View/Transformations)

- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/
- Defines different languages

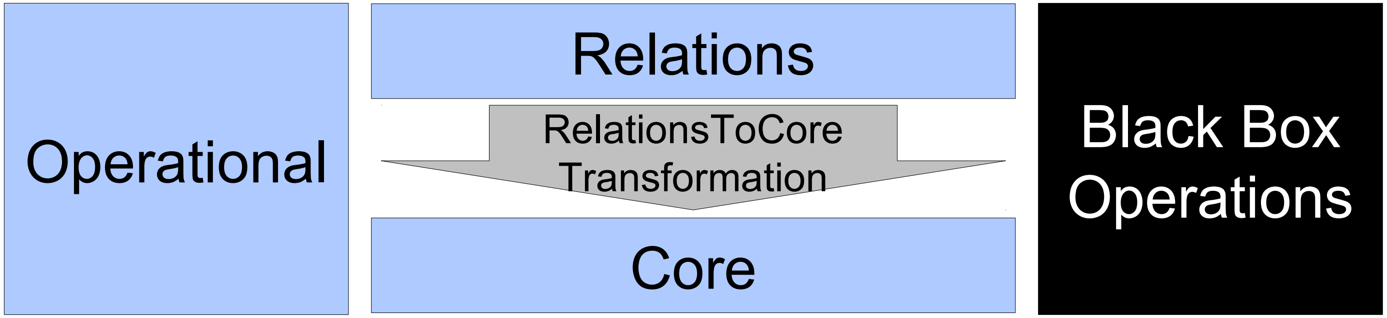
QVT (Query/View/Transformations)

- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/
- Defines different languages



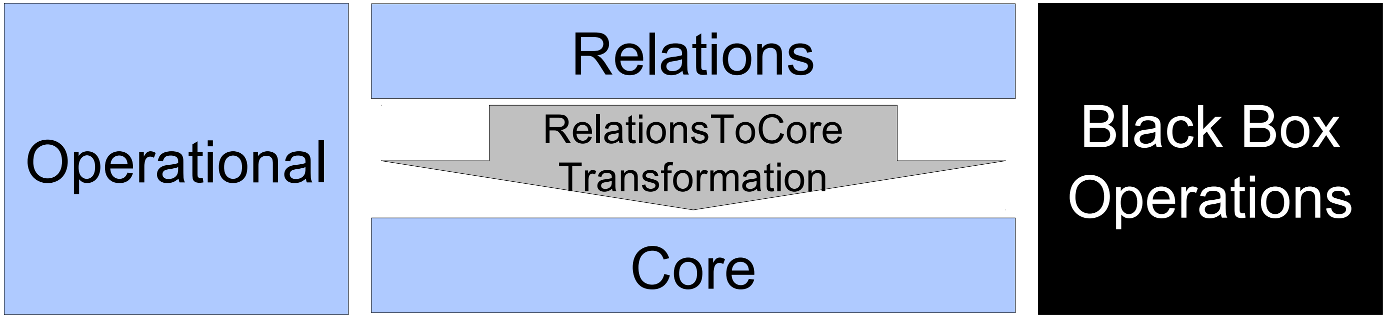
QVT (Query/View/Transformations)

- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/
- Defines different languages
  - **QVT-Relations** (QVT-R), declarative language, similar to TGGs



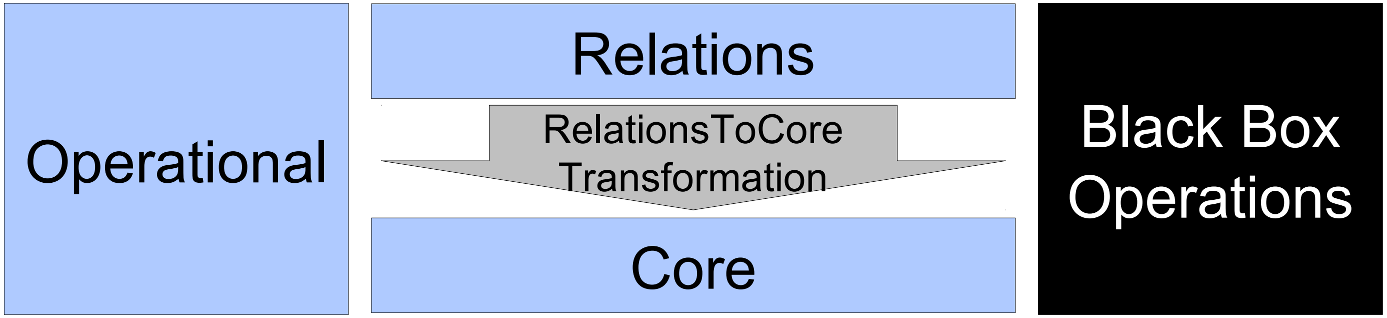
QVT (Query/View/Transformations)

- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/
- Defines different languages
  - **QVT-Relations** (QVT-R), declarative language, similar to TGGs
  - **QVT-Core**: declarative language, more simple than QVT-R, something that QVT-R can be compiled to for execution



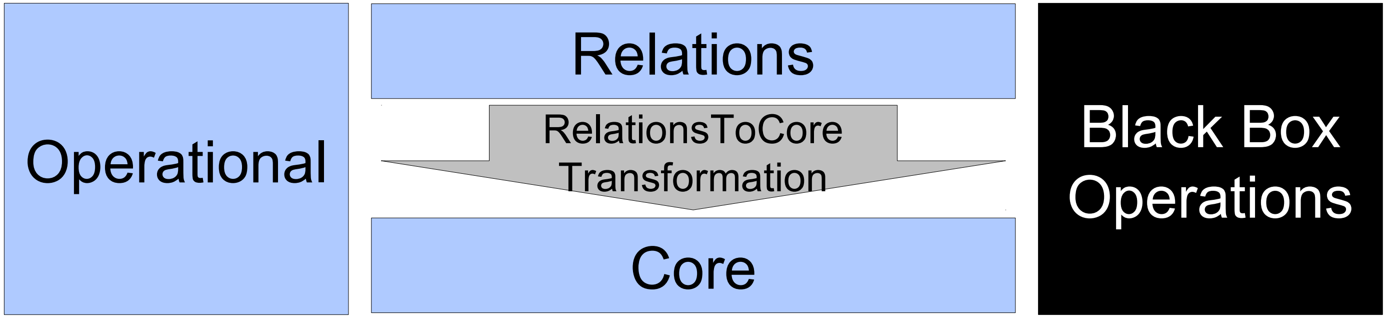
QVT (Query/View/Transformations)

- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/
- Defines different languages
  - **QVT-Relations** (QVT-R), declarative language, similar to TGGs
  - **QVT-Core**: declarative language, more simple than QVT-R, something that QVT-R can be compiled to for execution
  - **QVT-Operational**: An imperative language



QVT (Query/View/Transformations)

- QVT is an OMG standard for model transformations
  - see http://www.omg.org/spec/QVT/1.1/PDF/
- Defines different languages
  - **QVT-Relations** (QVT-R), declarative language, similar to TGGs
  - **QVT-Core**: declarative language, more simple than QVT-R, something that QVT-R can be compiled to for execution
  - **QVT-Operational**: An imperative language
  - **Black-Box Operations**: Ability to integrate other model transformation or programming languages



Operational

Relations

RelationsToCore Transformation

Core

Black Box Operations