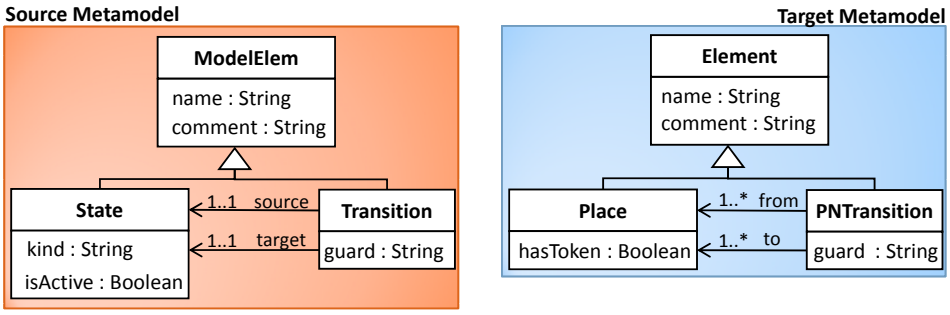
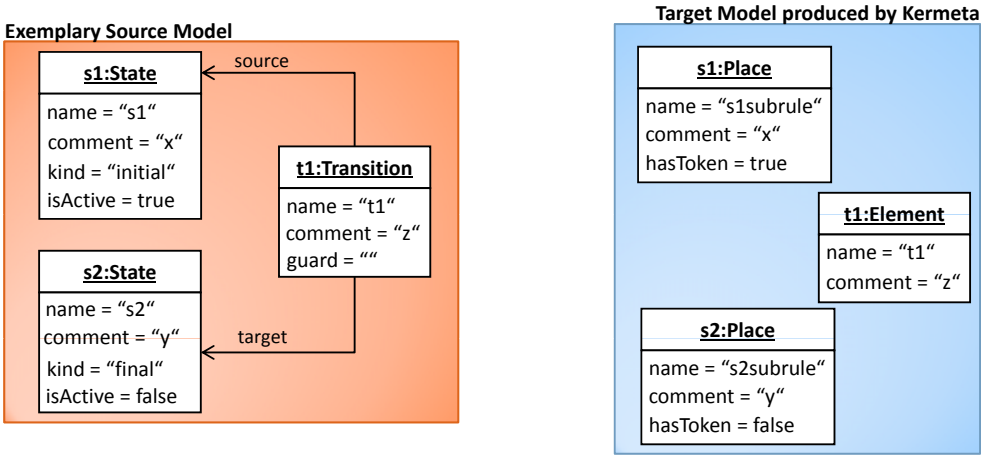
EXAMPLE 3

- **Description**
  - In this example `ModelElems` should be transformed into `Elements`
  - Furthermore, `States` should be transformed into `Places`; thereby this rule should again inherit from the base rule, but this time the inherited assignment should be overridden
- **Goal(s) of Evaluation**
  (1) Check whether the languages support for the overriding of inherited assignments



EXAMPLE 3 – KERMETA (1/3)



- **Results/Findings** (according to goals)
  (1) Overridings of assignments are supported

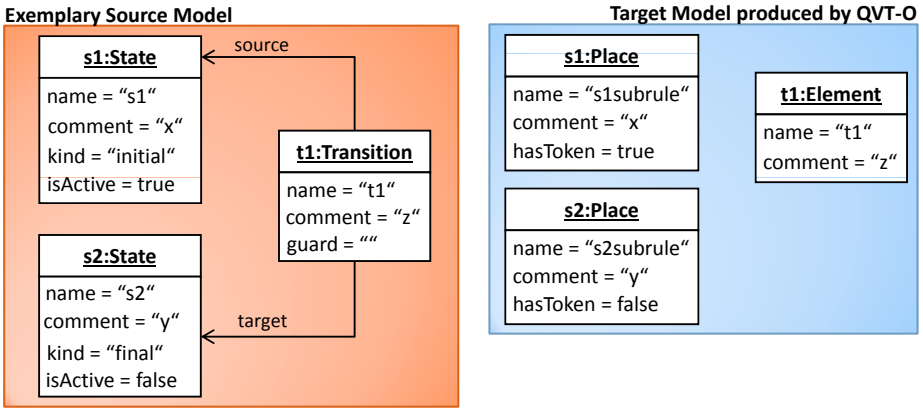
EXAMPLE 3 – KERMETA (2/3)

```
//transformation code for Statemachine2PetriNet
class Statemachine2PetriNet{
  operation conditionFulFilled(s : Statemachine) : kermeta::standard::Boolean is do
    result := true
  end

  operation assignments(s : Statemachine, p : PetriNet) is do
  end

  operation referenceAssignments(s : Statemachine, p : PetriNet, trace: Trace<Object, Object>) is do
    s.elements.each{ e |
      if trace.getTargetElem(e) != void then
        p.elements.add(trace.getTargetElem(e).asType(Element))
      end
    }
  end
}

//transformation code for ModelElem2Element
class ModelElem2Element{
  operation conditionFulFilled(m : ModelElem) : kermeta::standard::Boolean is do
    result := true
  end

  operation assignments(m : ModelElem, e : Element) is do
    e.name := m.name
    e.comment := m.comment
  end

  operation referenceAssignments(m : ModelElem, e : Element, trace: Trace<Object, Object>) is do
  end
}
```

EXAMPLE 3 – KERMETA (3/3)

```
//transformation code for State2Place
class State2Place inherits ModelElem2Element{
  method conditionFulFilled(m : ModelElem) : kermeta::standard::Boolean is do
    result := super(m)
  end

  method assignments(m : ModelElem, e : Element) is do
    super(m,e)
    (e.asType(Place)).hasToken := (m.asType(State)).isActive
    e.name := m.name + "subrule"
  end

  method referenceAssignments(m : ModelElem, e : Element, trace: Trace<Object, Object>) is do
    super(m,e,trace)
  end
}
```

EXAMPLE 3 – QVT-O (1/2)



- **Results/Findings** (according to goals)

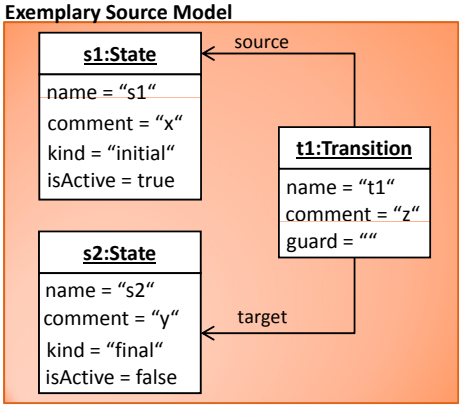
  (1) Overridings of assignments are supported
  This is inferred from the fact that the name assignments have been realized according to the definition as given by the subrule (when adding debugging messages with `dump()`, one may find that the original assignments are executed before the overriding assignments in the execution and thus, the assignments are executed parent-driven)

EXAMPLE 3 – QVT-O (2/2)

```
transformation testTrafo(in inModel : sm, out outModel : pn);
  main() {
    inModel.rootObjects()[Statemachine] -> map SM2Petri();
  }

  mapping Statemachine::SM2Petri() : PetriNet {
    //please note that specific rules must be called first!
    elements := self.elements[State] -> map State2Place();
    elements += self.elements[ModelElem] -> map ModelElem2Element();
  }

  mapping ModelElem::ModelElem2Element() : Element {
    name := self.name;
    dump('superregel fuer ' + self.name);
    comment := self.comment;
  }

  mapping State::State2Place() : Place inherits ModelElem::ModelElem2Element {
    name := self.name + 'subrule';
    dump('subregel fuer ' + self.name);
    hasToken := self.isActive;
  }
```

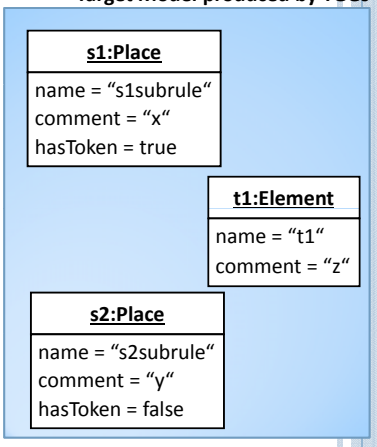
EXAMPLE 3 – TGGS (1/2)

Exemplary Source Model



Rule definitions see next slide

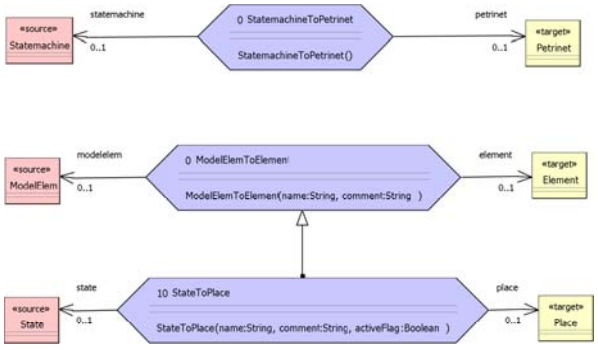
Target Model produced by TGGs



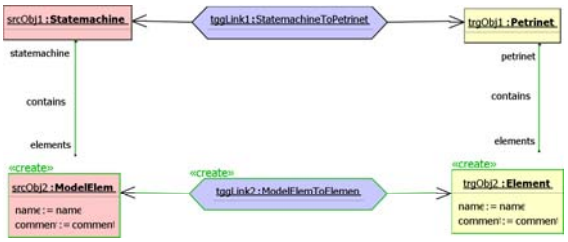
- **Results/Findings** (according to goals)
  - (1) Overridings of assignments are supported as long as the assignment of the subrule is at least as strict as the assignment of the superrule. TGGs do not allow a subrule to match less strict than all its superrules;

EXAMPLE 3 – TGGS (2/2)

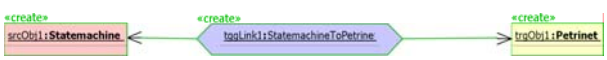
TGG-Schema (type level)



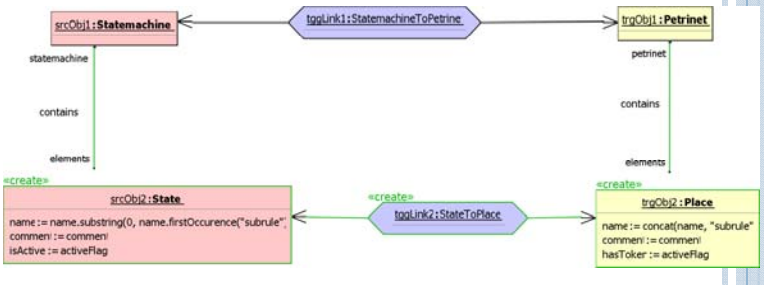
ModelElemToElement(…)



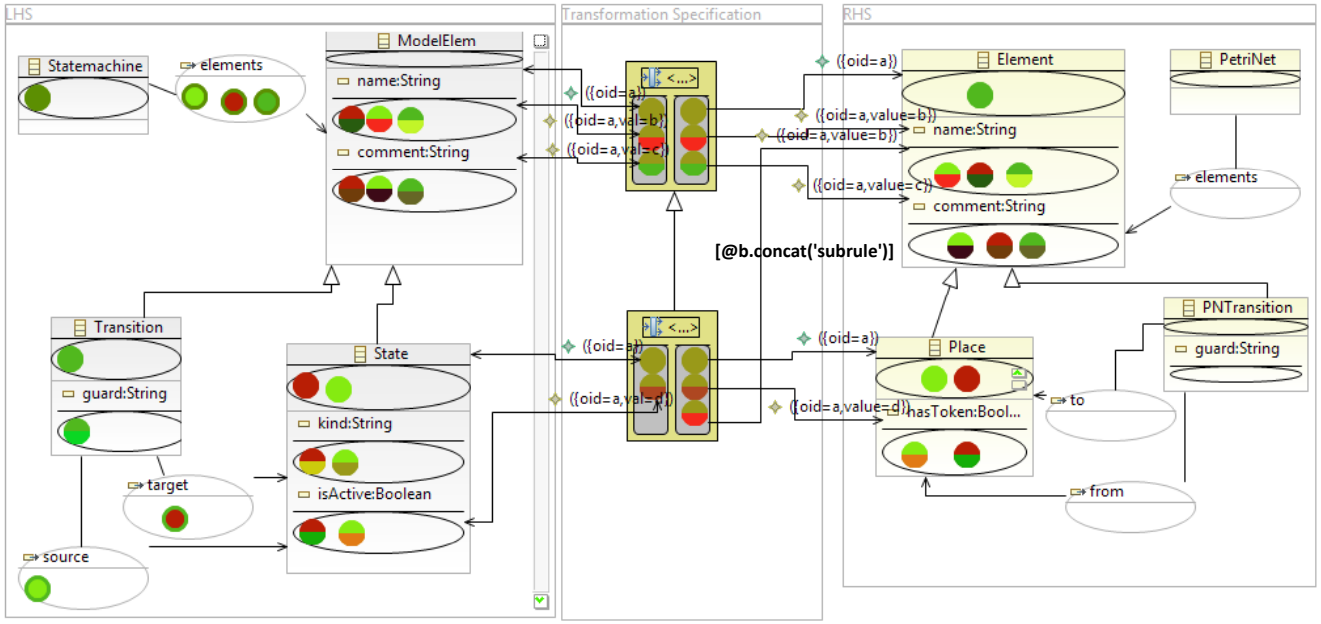
StatemachineToPetrinet(…)



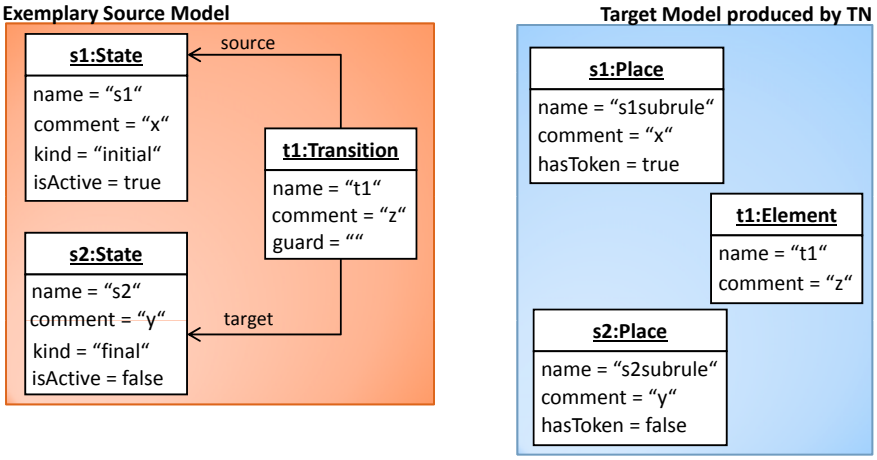
StateToPlace(…)



EXAMPLE 3 – TNS (1/2)



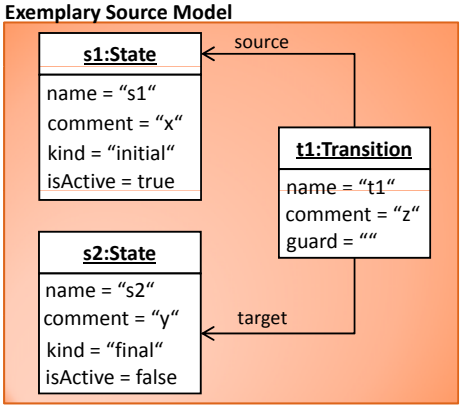
EXAMPLE 3 – TNS (2/2)



- **Results/Findings** (according to goals)
  - (1) Overridings of assignments are supported
    This is inferred from the fact, that the name assignments have been realized according to the definition as given by the subrule (the original assignments are substituted by the overriding assignments; since the patterns are merged to a common transition, it is not decidable which assignments are executed first)

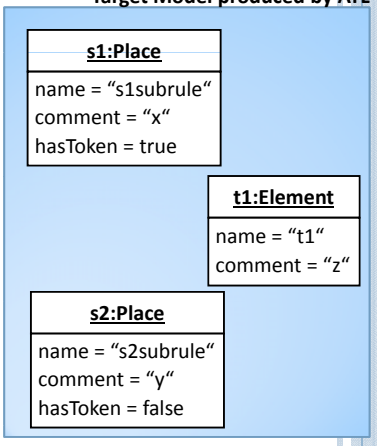
EXAMPLE 3 – ATL

Exemplary Source Model




```
rule ModelElem2Element{
 from mElem : Statemachine!ModelElem
 to elem : Petrinet!Element (
  name <- mElem.name,
  comment <- mElem.comment
 )
}

rule State2Place extends ModelElem2Element {
 from mElem : Statemachine!State
 to elem : Petrinet!Place (
  name <- mElem.name + 'subrule',
  hasToken <- mElem.isActive
 )
}
```

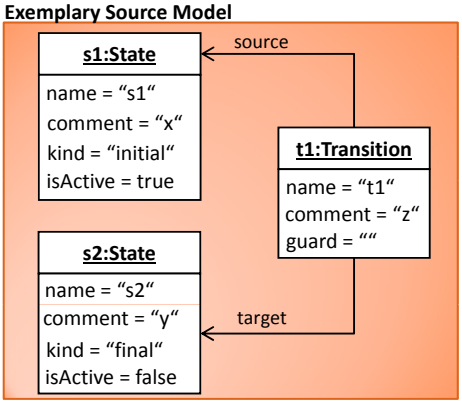
Target Model produced by ATL



- **Results/Findings** (according to goals)
  - (1) Overridings of assignments are supported
    This is inferred from the fact that the name assignments have been realized according to the definition as given by the subrule (when adding debugging messages, one can find that the original assignments are substituted by the overriding assignments in the execution and that the assignments are executed descendant-driven)

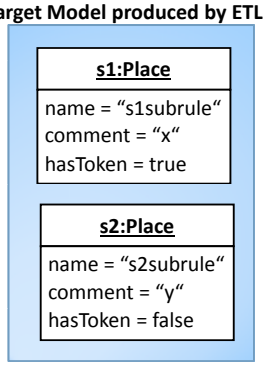
EXAMPLE 3 – ETL

Exemplary Source Model




```
rule ModelElem2Element
 transform mElem : Statemachine!ModelElem
 to elem : Petrinet!Element {
   elem.name := mElem.name;
   elem.comment := mElem.comment;
 }

rule State2Place
 transform mElem : Statemachine!State
 to elem : Petrinet!Place
 extends ModelElem2Element {
  elem.name := mElem.name + 'subrule' ;
  elem.hasToken := mElem.isActive;
 }
```

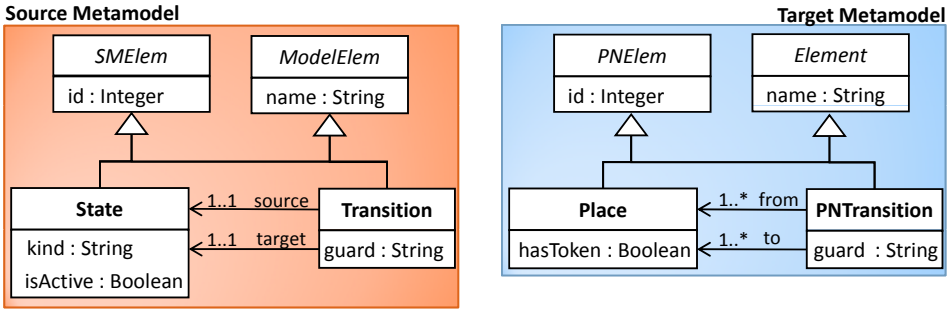
Target Model produced by ETL



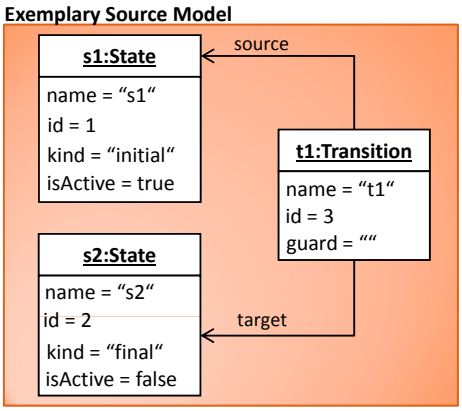
- **Results/Findings** (according to goals)
  - (1) Overridings of assignments are supported
    This is inferred from the fact that the name assignments have been realized according to the definition as given by the subrule (when adding `println()` messages, one may find that the original assignments are executed first, then the assignments of the subrule, i.e., parent-driven)

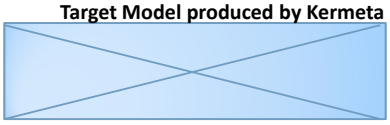
Example 4

- **Description**
  - In this example `ModelElems` should be transformed into `Elements` and `SMElems` into `PNElems`, both requiring abstract rules, since the classes are `abstract` themselves
  - Moreover, `State` instances should be transformed into `Place` instances, inheriting from both `abstract` rules to avoid code duplicaton, thus multiple inheritance is needed
- **Goal(s) of Evaluation**
  (1) Check whether multiple inheritance is supported and if all inherited assignments are done correspondingly



Example 4 – Kermeta (1/3)





- **Results/Findings** (according to goals)
  (1) Although Kermeta supports multiple inheritance, equally named operations with different signatures (as in the example) result in problems; this may also not be resolved by explicitly choosing a certain operation with the `from` keyword
  (2) Anyway the example is not resolvable, since one operation must be chosen and thus, either the `name` assignment or the `id` assignment would be lost

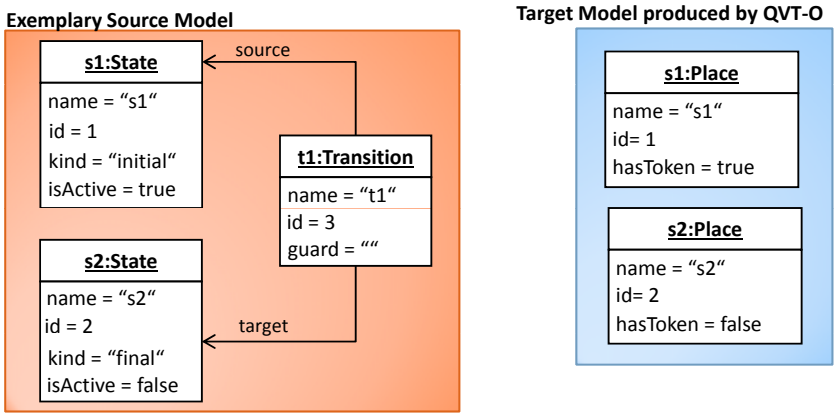
EXAMPLE 4 – KERMETA (2/3)

```
//transformation code for Statemachine2PetriNet
class Statemachine2PetriNet{

  operation conditionFulFilled(s : Statemachine) : kermeta::standard::Boolean is do
    result := true
  end

  operation assignments(s : Statemachine, p : PetriNet) is do
  end

  operation referenceAssignments(s : Statemachine, p : PetriNet, trace: Trace<Object, Object>) is do
    s.elements.each{ e |
      if trace.getTargetElem(e) != void then
        p.elements.add(trace.getTargetElem(e).asType(Element))
      end
    }
  end
}

//transformation code for ModelElem2Element
abstract class ModelElem2Element{

  operation conditionFulFilled(m : ModelElem) : kermeta::standard::Boolean is do
    result := true
  end

  operation assignments(m : ModelElem, e : Element) is do
    e.name := m.name
  end

  operation referenceAssignments(m : ModelElem, e : Element, trace: Trace<Object, Object>) is do
  end
}
```

EXAMPLE 4 – KERMETA (3/3)

```
//transformation code for SMElem2PNElem
abstract class SMElem2PNElem{

  operation conditionFulFilled(m : SMElem) : kermeta::standard::Boolean is do
    result := true
  end

  operation assignments(m : SMElem, e : PNElem) is do
    e.id := m.id
  end

  operation referenceAssignments(m : SMElem, e : PNElem, trace: Trace<Object, Object>) is do
  end
}

//transformation code for State2Place
class State2Place inherits ModelElem2Element, SMElem2PNElem{

  method conditionFulFilled(m : ModelElem) : kermeta::standard::Boolean from ModelElem2Element is do
    result := super(m)
  end

  method assignments(m : ModelElem, e : Element) from ModelElem2Element is do
    super(m,e)
    (e.asType(Place)).hasToken := (m.asType(State)).isActive
  end

  method referenceAssignments(m : ModelElem, e : Element, trace: Trace<Object, Object>)
  from ModelElem2Element is do
    super(m,e,trace)
  end
}
```

EXAMPLE 4 – QVT-O (1/2)

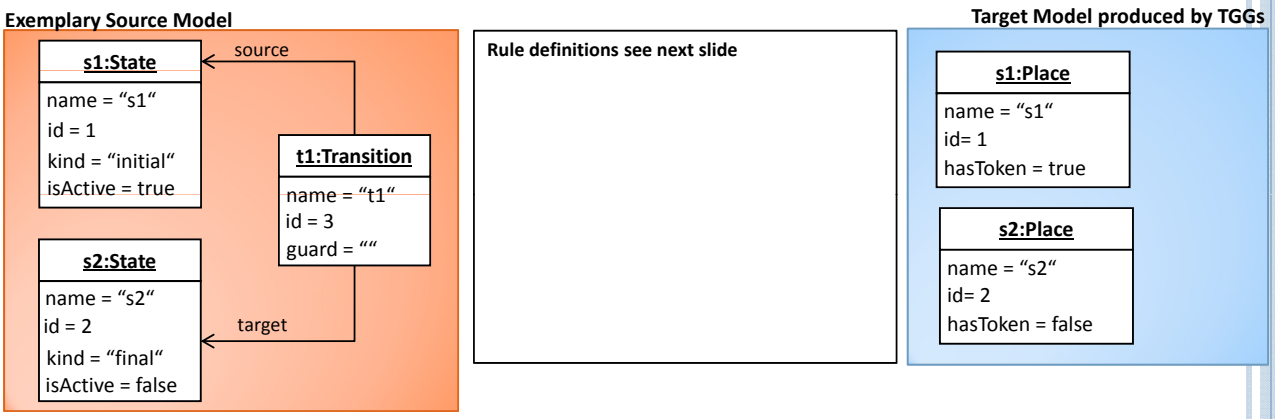


- **Results/Findings** (according to goals)
  (1) QVT-O supports multiple inheritance. The assignments are again executed in parent-driven manner, i.e., first the assignment of `ModelElem2Element` are executed, followed by the assignments of `SMElem2PNElem` and finally, the assignments of `State2Place`

EXAMPLE 4 – QVT-O (2/2)

```
transformation testTrafo(in inModel : sm, out outModel : pn);

  main() {
    inModel.rootObjects()[Statemachine] -> map SM2Petri();
  }

  mapping Statemachine::SM2Petri() : PetriNet {
    elements := self.elements[State] -> map State2Place();
  }

  abstract mapping ModelElem::ModelElem2Element() : Element {
    dump('ModelElem2Element fuer ' + self.name);
    name := self.name;
  }

  abstract mapping SMElem::SMElem2PNElem() : PNElem {
    dump('SMElem2PNElem fuer ' + self.id.toString());
    id := self.id;
  }

  mapping State::State2Place() : Place inherits ModelElem::ModelElem2Element, SMElem::SMElem2PNElem {
    dump('State2Place fuer ' + self.name);
    hasToken := self.isActive;
}
```

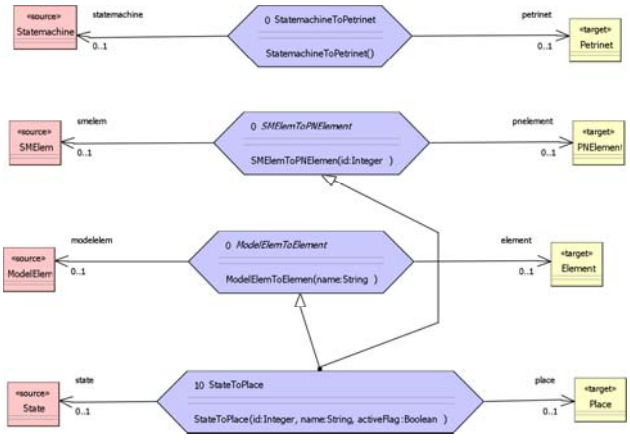
EXAMPLE 4 – TGGS (1/2)



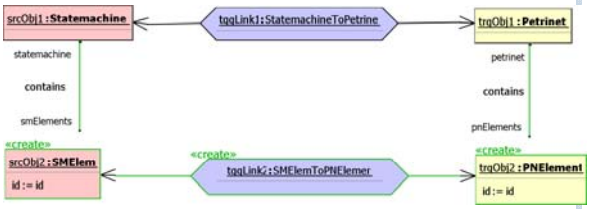
- **Results/Findings** (according to goals)
  (1) TGGs support multiple inheritance. Since all assignments of a superrule are copied explicitly into subrules, all assignments of multiple superrules are executed in subrules.
  Nevertheless, the inherited assignments must not be adjusted in a way that they match less restrictive than the original assignments (assured by static analysis).

EXAMPLE 4 – TGGS (2/2)

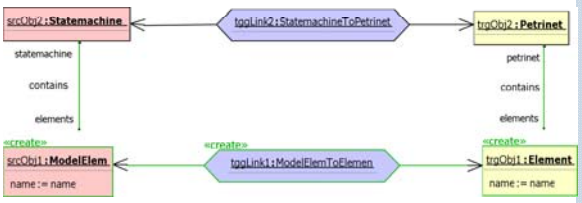
TGG-Schema (type level)



SMElemTOPNElement(…)

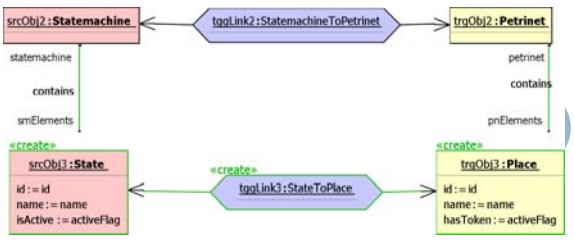


ModelElemToElement(…)

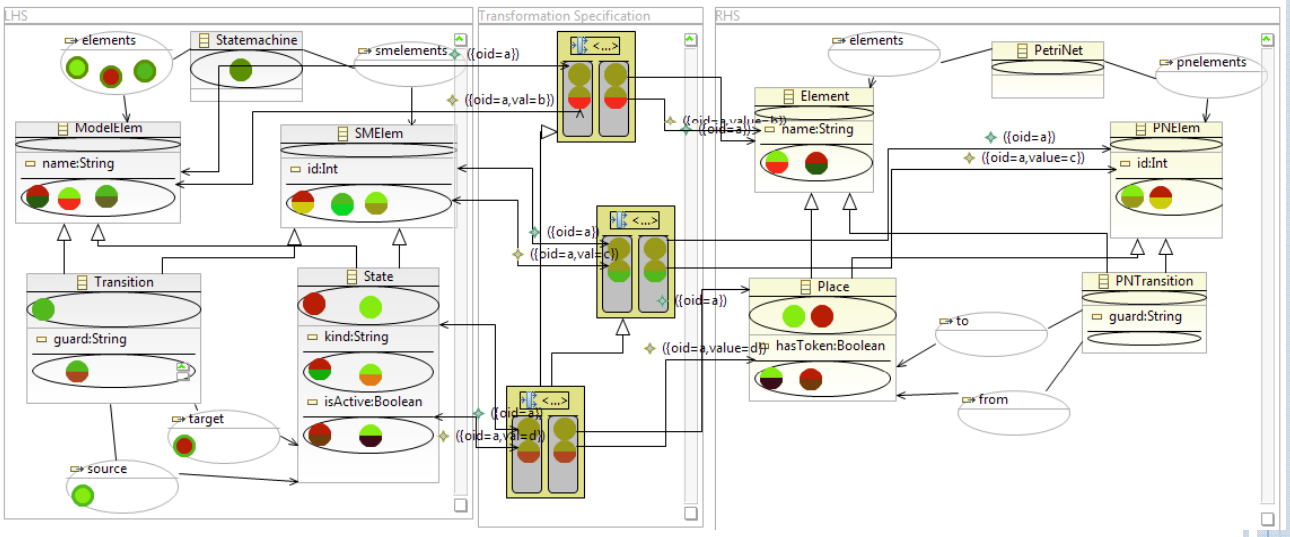


StatemachineToPetrinet(…)

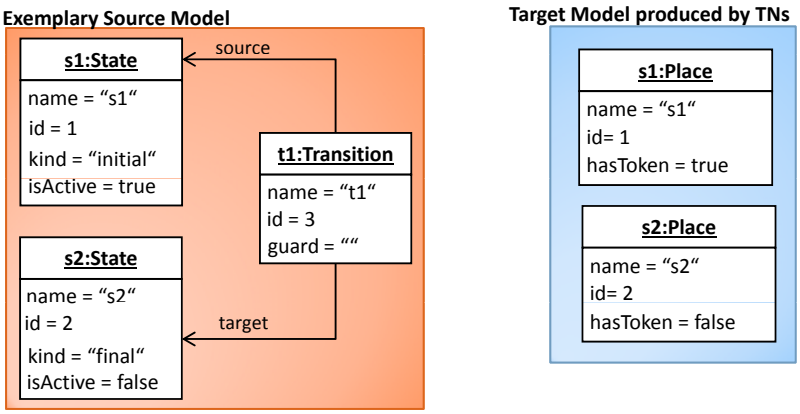
StateToPlace(…)



EXAMPLE 4 – TNS (1/2)



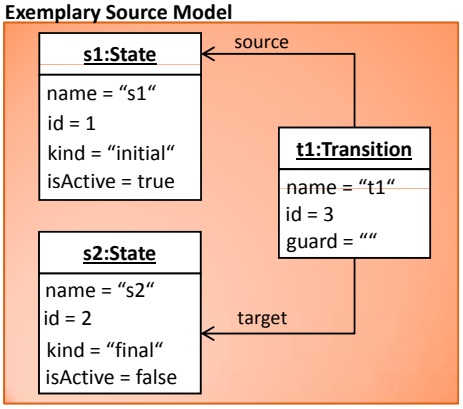
EXAMPLE 4 – TNS (2/2)



- **Results/Findings** (according to goals)
  - (1) TNs support multiple inheritance. Since all assignments of a superrule are copied explicitly into subrules, all assignments of multiple superrules are executed in subrules.

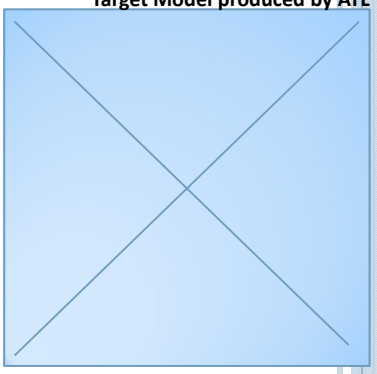
EXAMPLE 4 – ATL

Exemplary Source Model



???

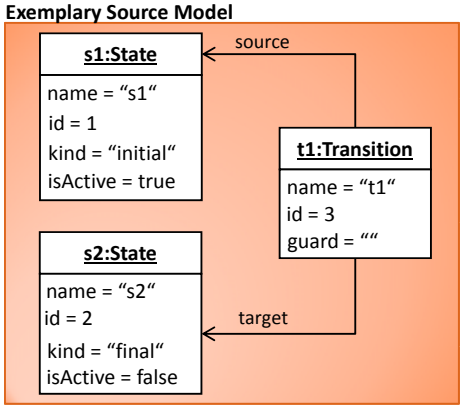
Target Model produced by ATL



- **Results/Findings** (according to goals)
  - (1) No support for multiple inheritance available

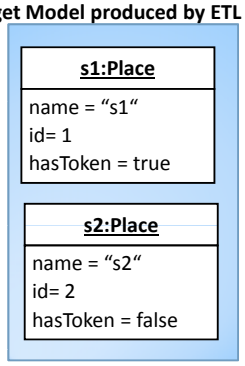
EXAMPLE 4 – ETL

Exemplary Source Model




```
@abstract
rule ModelElem2Element
  transform mElem : Statemachine!ModelElem
  to elem : Petrinet!Element {
     elem.name := mElem.name;
  }

@abstract
rule SMElem2PNElem
  transform mElem : Statemachine!SMElem
  to elem : Petrinet!PNElem {
     elem.id := mElem.id;
  }

rule State2Place
  transform mElem : Statemachine!State
  to elem : Petrinet!Place
  extends ModelElem2Element,
SMElem2PNElem {
     elem.hasToken := mElem.isActive;
  }
```

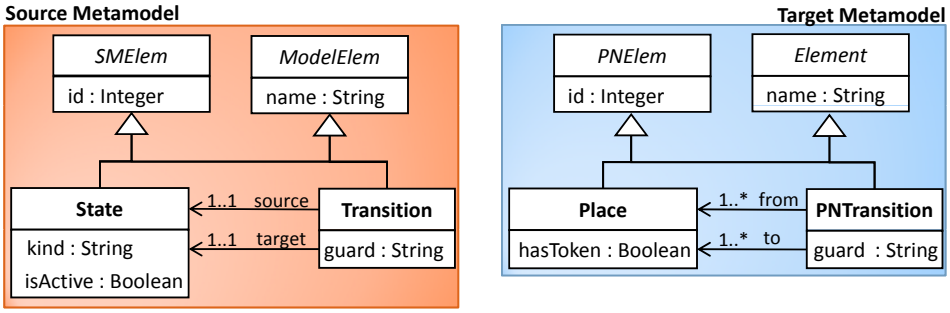
Target Model produced by ETL



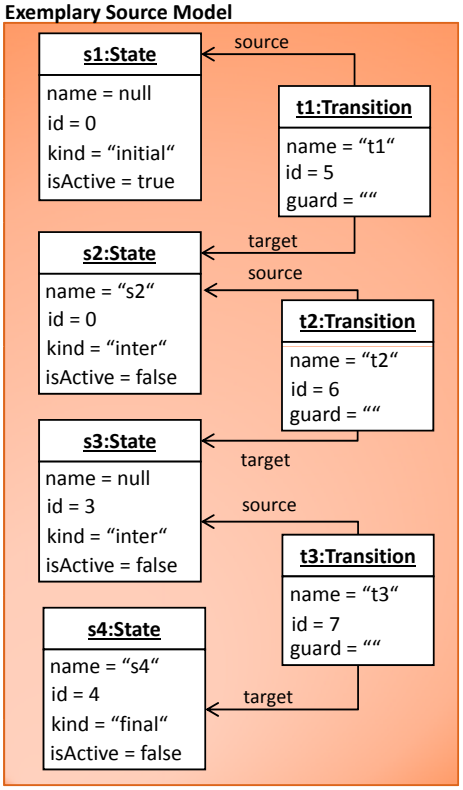
- **Results/Findings** (according to goals)
  - (1) ETL supports multiple inheritance. The assignments are again executed in a parent-driven manner, i.e., first the assignments of `ModelElem2Element` are executed, followed by the assignments of `SMElem2PNElem` and finally, the assignments of `State2Place`

Example 5

- **Description**
  - In this example `ModelElems` should be transformed into `Elements` and `SMElems` into `PNElems`, both requiring abstract rules, since the classes are abstract themselves; additionally conditions are specified, i.e., only those `ModelElems` whose `name` is not `null` and those `SMElems` whose `id` is greater `0` should be transformed
  - Moreover, `State` instances should be transformed into `Place` instances, inheriting from both abstract rules to avoid code duplicaton, thus multiple inheritance is needed
- **Goal(s) of Evaluation**
  - (1) Check how conditions are interpreted in case of multiple inheritance



Example 5 – Kermeta



See Example 4

Target Model produced by Kermeta