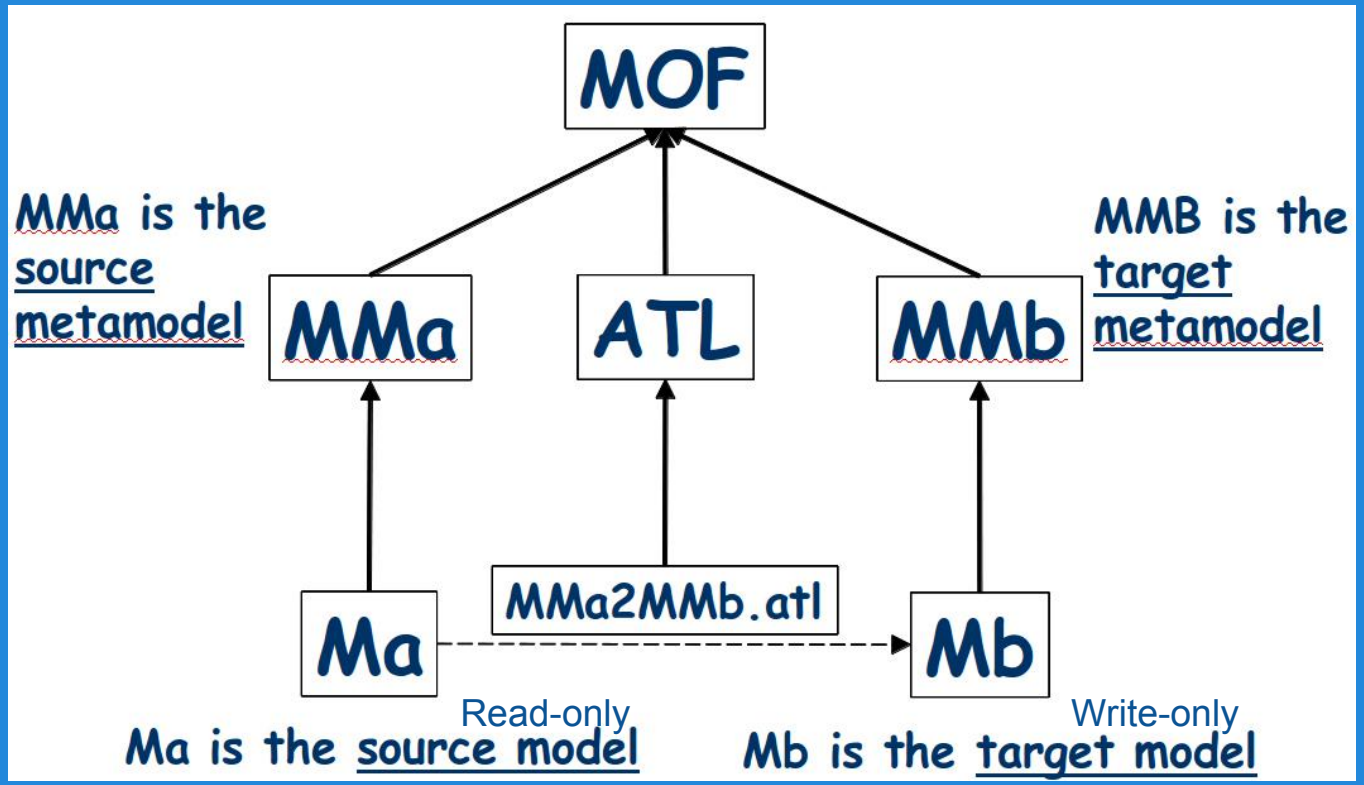
Demystifying ATL

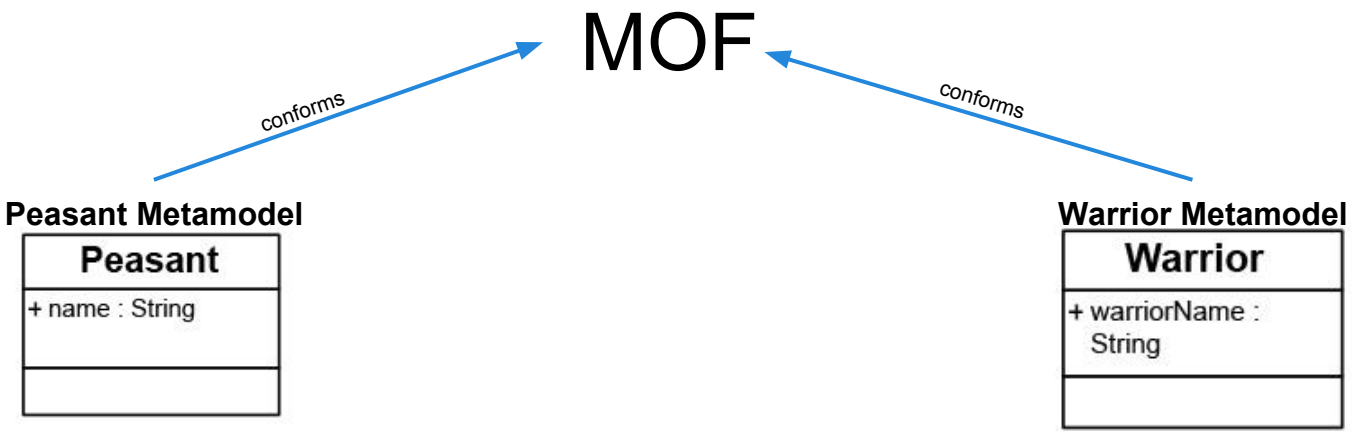
and an Introduction to the RPG2Petrinet Project

Demystifying ATL

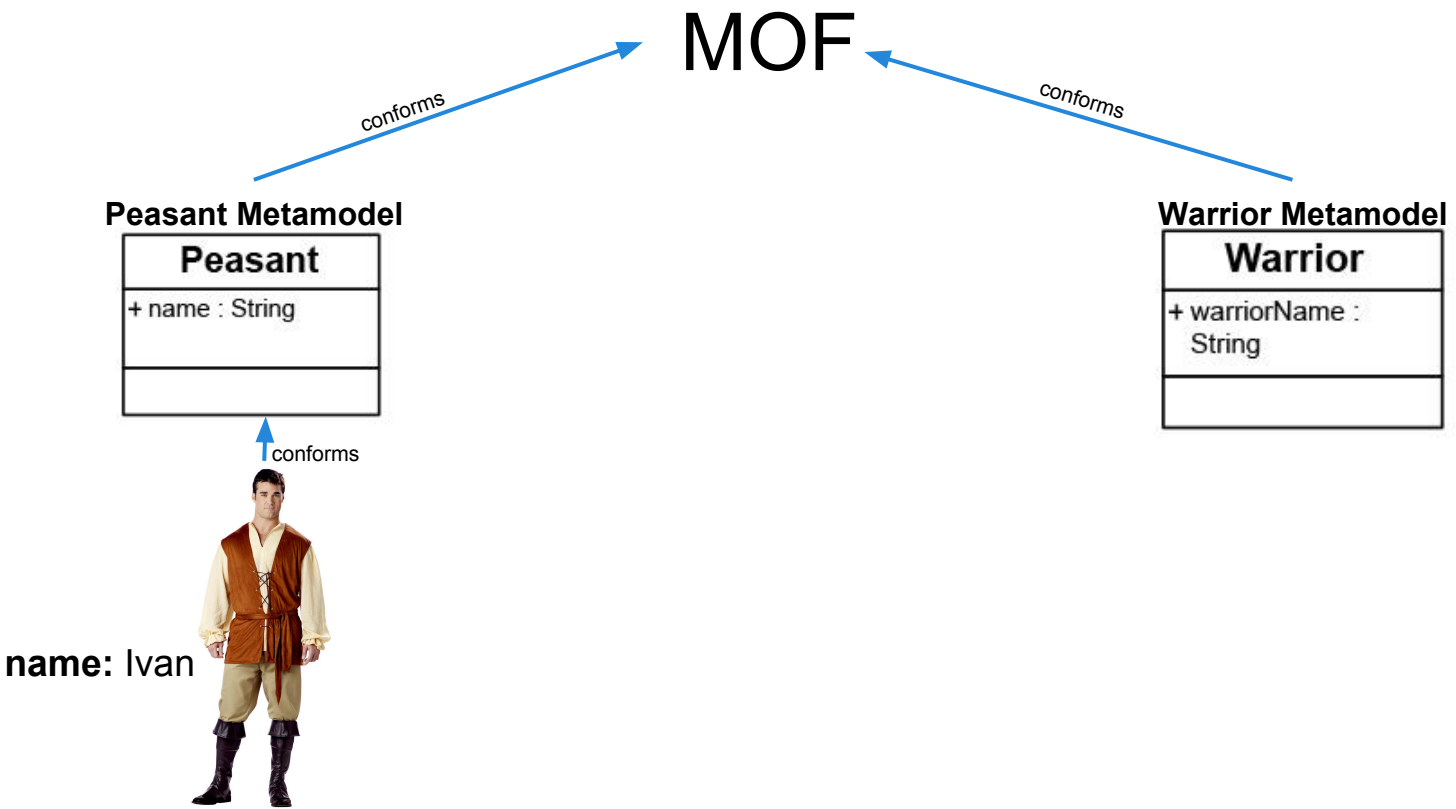


Transformation pattern

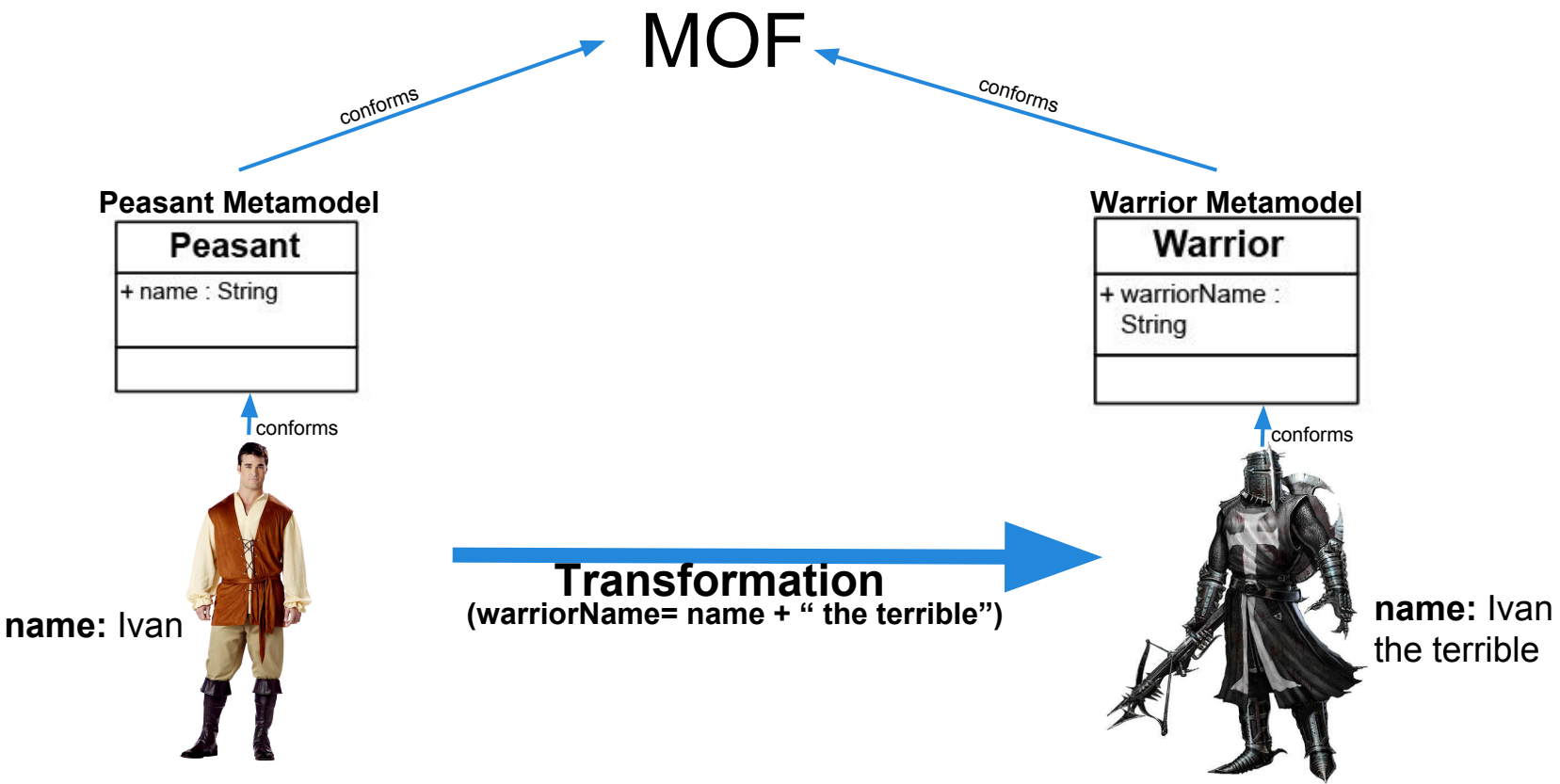
Simple transformation example



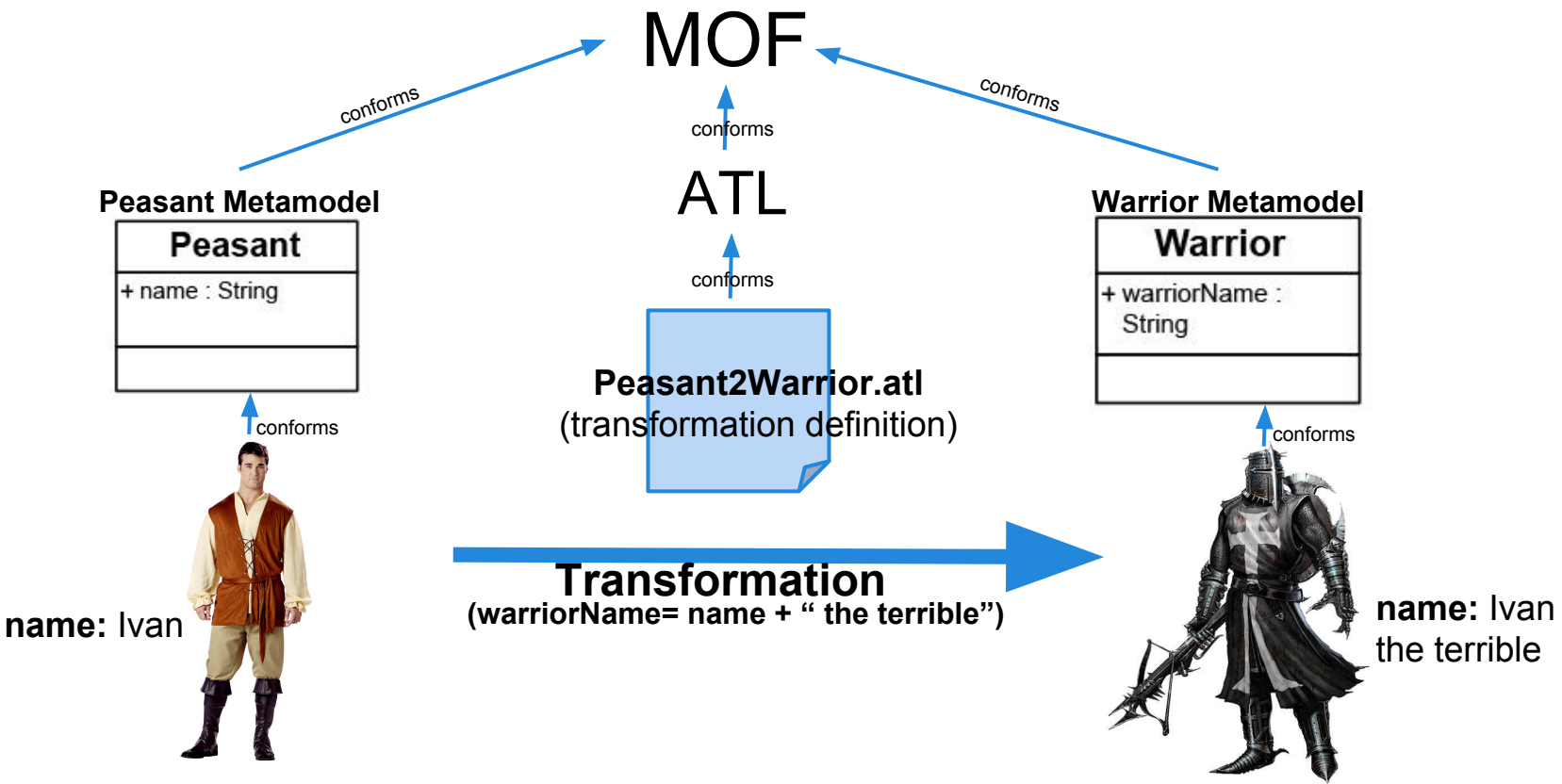
Simple transformation example



Simple transformation example



Simple transformation example



Peasant2Warrior.atl

```
module Peasant2Warrior;
create OUT : Warrior from IN : Peasant;

rule upgradeToWarrior {
    from
        p: Peasant!Peasant
    to
        w: Warrior!Warrior (
        warriorName <- p.name + " the terrible")
}
```

Peasant2Warrior.atl

```
module Peasant2Warrior;
create OUT : Warrior from IN : Peasant;

rule upgradeToWarrior {
    from
        p: Peasant!Peasant (p.name == "ivan")
    to
        w: Warrior!Warrior (
        warriorName <- p.name + " the terrible")
}
```

GUARD!

IN-PATTERN
set of source types and guard
ATL Engine tries to find set of matches of this pattern in source model

OUT-PATTERN
set of target type elements being created when applied and a set of bindings.

The power of ATL

- **Traceability** between: *the rule, the match and the newly created target elements.*
- **HOT:** Transformations itself are models -> used as input.
- **In-Place transformations:** newer versions! simply by putting instead of from:
  create OUT : MMa refining IN : MMa

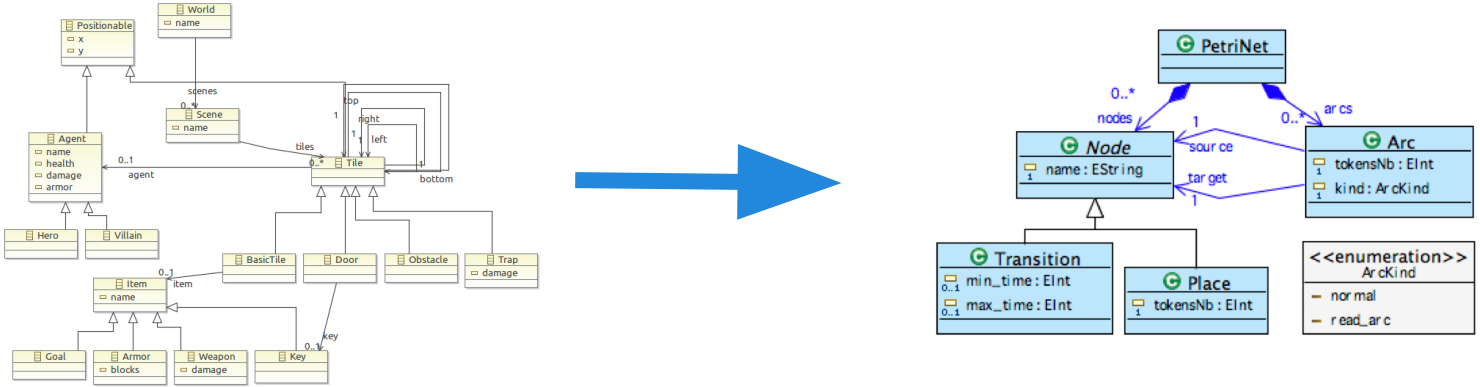
Project

Operational semantics

- Initially planned implementing operational semantics of RPG.
- ATL doesn’t support step-wise execution. -> applies all found matches for all rules -> goes from init state to final state in just a blink.

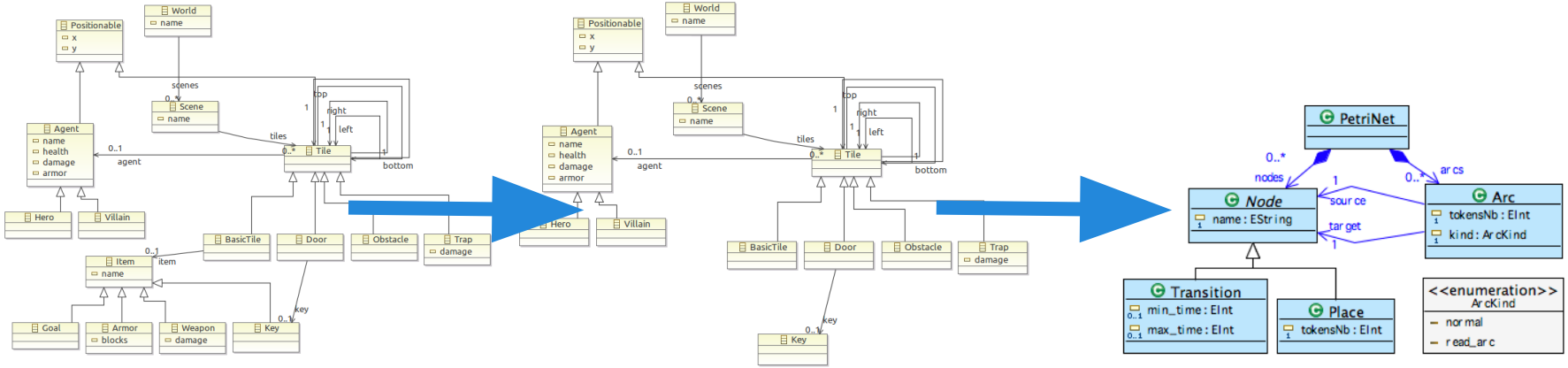
The RPG2Petrinet experiment

Focus on denotational part:



Idea based on paper:"*The RPG DSL: A Case Study of Language Engineering Using MDD for Generating RPG Games for Mobile Phones*" by Marques et al

The RPG2Petrinet experiment



RPG MM

μRPG MM
(filter, keeps only essential entities)

Petrinet MM

Thank you