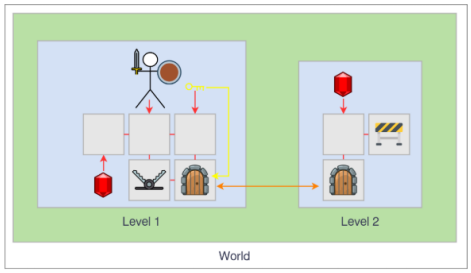
Graphical Abstract

Rule-Based Translational Semantics using ATL

Niels Van den Broeck



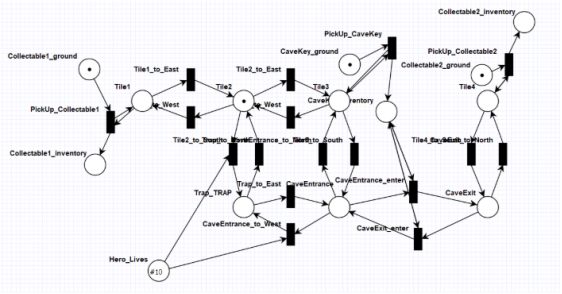
```
rule Door2Place {
    from
        s : RPG!Door
    to
        p : PetriNet!Place {
            name <- s.name,
            tokens <- if s.hasHero() then 1 else 0 endif
        },
        t_teleport : PetriNet!Transition {
            name <- s.name + '_enter',
            input <- if not s.requiresKey.oclIsUndefined()
                     then Sequence{p, thisModule.resolveTemp(s.requiresKey, 'p_inv')}
                     else Sequence{p} endif,
            output <- if not s.requiresKey.oclIsUndefined()
                      then Sequence{s.leadsTo, thisModule.resolveTemp(s.requiresKey, 'p_inv')}
                      else Sequence{s.leadsTo} endif
        )
}

rule Item2PetriNet {
    from
        s : RPG!Item
    to
        p_ground : PetriNet!Place { name <- s.name + '_ground', tokens <- 1 },
        p_inv : PetriNet!Place { name <- s.name + '_inventory', tokens <- 0 },
        t_pickup : PetriNet!Transition {
            name <- 'PickUp_' + s.name,
            input <- Sequence{ p_ground, s.refImmediateComposite() },
            output <- Sequence{ p_inv, s.refImmediateComposite() }
        )
}

lazy rule MoveNorth {
    from s : RPG!Tile
    to t : PetriNet!Transition {
        name <- s.name + '_to_North',
        -- INPUT: Check if destination (s.north) is a Trap
        input <- if s.north.oclIsTypeOf(RPG!Trap)
                 then Sequence { s, thisModule.resolveTemp(thisModule.getHero(), 'p_lives') }
                 else Sequence { s } endif,

        output <- s.north
    )
}

lazy rule MoveSouth {
    from s : RPG!Tile
    to t : PetriNet!Transition {
```



Rule-Based Translational Semantics using ATL

Niels Van den Broeck

University of Antwerp, Computer Science, Antwerp, Belgium

Abstract

This project presents a study on Rule-Based translational Semantics within the context of Model Driven Engineering (MDE). The project highly correlates with the last assignment of the MA course 'Model Driven Engineering' in University of Antwerp. The primary objective is to demonstrate the translation of static rules of an RPG into a dynamic Petri Net model using the ATLAS Transformation Language (ATL). The workflow uses the Eclipse Modeling Framework (EMF) and is executed in three phases: metamodeling, instantiation, and transformation. We first defined both the source (RPG) as the target (Petri Net) metamodels using Ecore. Then, an RPG model instance is transformed into a Petri Net using ATL. The final output is an XMI file converted into TAPN, allowing for visualization and execution of the game logic within TAPAAL.

Keywords: ATLAS Transformation Language (ATL), Metamodeling (Ecore), Model Driven Engineering (MDE), Petri Nets, Eclipse Modeling Framework (EMF)

1. Introduction

In this project, we will present a study on Rule-Based Translational Semantics using ATL (ATLAS Transformation Language), demonstrating how we can translate the static rules of an RPG into Petri Nets. The project workflow relies on the Eclipse Modeling Framework (EMF) and is divided into three phases: metamodeling, instantiation, and transformation. In section 2, we will look at what ATL is exactly and what it has to offer. Section 3 is about the creation of both the source (RPG) and the target (Petri Net) metamodels. In section 4, an example model is created which we will then translate into a Petri Net using rules explained in section 5.

2. ATL

As described by Allilaire et al. (2006), the ATLAS Transformation Language (ATL) is a model transformation language to specify how source models can be translated into target models. It is developed as part of the ATLAS Model Management Architecture (AMMA) platform (Bézivin et al., 2004). It operates within the Eclipse ecosystem and relies on the Eclipse Modeling Framework (EMF) to handle model conformance.

2.1. Hybrid Language Design

An important characteristic of ATL is its hybrid approach between declarative and imperative programming constructs:

- **Declarative Style**: The encouraged style of specifying transformations. Developers describe what the target elements should look like in relation to the source elements. This is done using "Matched Rules".
- **Imperative Style**: For when declarative solutions are not feasible, "Called Rules" or "Action Blocks" are provided which allow developers to define explicit execution sequences.

2.2. Transformation Architecture

The transformation pattern of ATL is shown in Figure 1. In this pattern, we have a source model Ma, which must conform to a specific source Metamodel MMa. This model is transformed into a target model Mb, which must once again be conformed to its Metamodel MMb. This translation is done according to a transformation definition mma2mmb.atl written in the ATL language. The transformation definition is a model conforming to the ATL metamodel. All metamodels conform to the MOF (Meta Object Facility) which is the metametamodel.

3. Metamodels

Obviously, we need to define our Metamodels before we can translate any models. This section explores the creation of the source (RPG) as the target (Petri Net) Metamodels. both of these are defined in Ecore, the language that is used to define other languages in EMF.

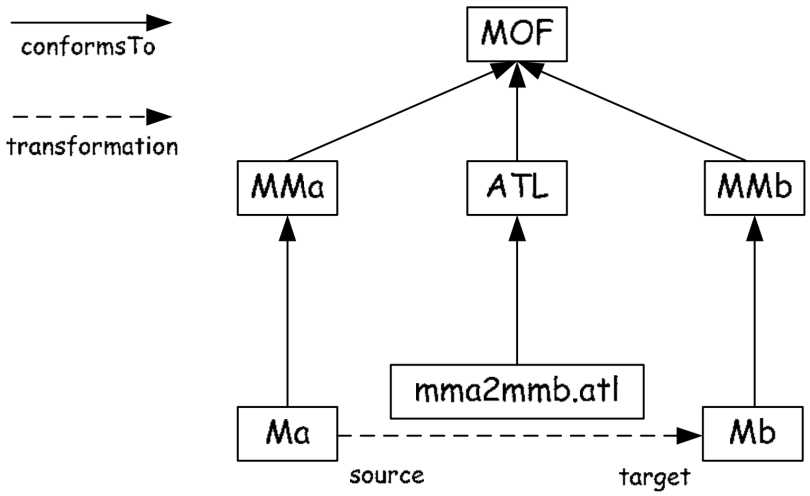


Figure 1: Overview of ATL transformational approach

3.1. The Source Metamodel: RPG

I created a custom Role-Playing Game Metamodel which is conceptually identical to the Metamodel created in assignment 2 of the Model Driven Engineering course[1] (Van den Broeck and Stuer). The only difference is that it was made in muMLE (Bézivin et al., 2004), while this Metamodel is defined in Ecore. Both have a class hierarchy, cardinality, constraints, attributes, and references (or associations), making it easy and fast to copy into the new formalism.

This Metamodel has a world, containing at least one level and exactly 1 hero. The hero has an attribute 'lives', which can be any positive number. Levels contain tiles (at least 1). The tile is an abstract class, with three different children:

- StandardTile: Normal tile on which the hero can stand on. It may contain an item (see later).
- Trap: Another tile where the hero can stand on, but loses a life each time he/she does.

[1]For simplicity reasons, I did not implement the behavior of monsters, and the order of events.

- Obstacle: The hero cannot stand on this tile. This tile has currently no functionality, but can be useful for future expansions like breaking obstacles to clear paths. It is included in this project to demonstrate an edge case while translating to the target model.
- Door: A door is connected to another door in a different level. It requires a key to use the door.

Additionally, tiles are connected to each other using references: North, East, South, and West. A tile can contain an item. There are two types of items:

- Objective: An objective is a collectible that is introduced to give the game a goal. Collecting all objectives wins you the game.
- Key: A key is required to open doors.

Lastly, all these Classes have a name attribute.

3.2. The target Metamodel: Petri Net

For the target domain, we adopted the standard Petri Net Metamodel. Rather than designing this from scratch, I reused an existing definition from GEMOC Studio. The class diagram in Figure 3 shows that the Metamodel consists of three classes:

- Net: This acts as the root container for the entire model. It has references to places and transitions.
- Place: The place represents the state-holding elements of the network. It has a name, and possible Tokens (markings).
- Transition: This represents the node that change the state of the net. Each transition has a name and it's connected to the rest of the graph via either input arcs (coming from a place to the transition) or output arcs (coming out of the transition into another place).

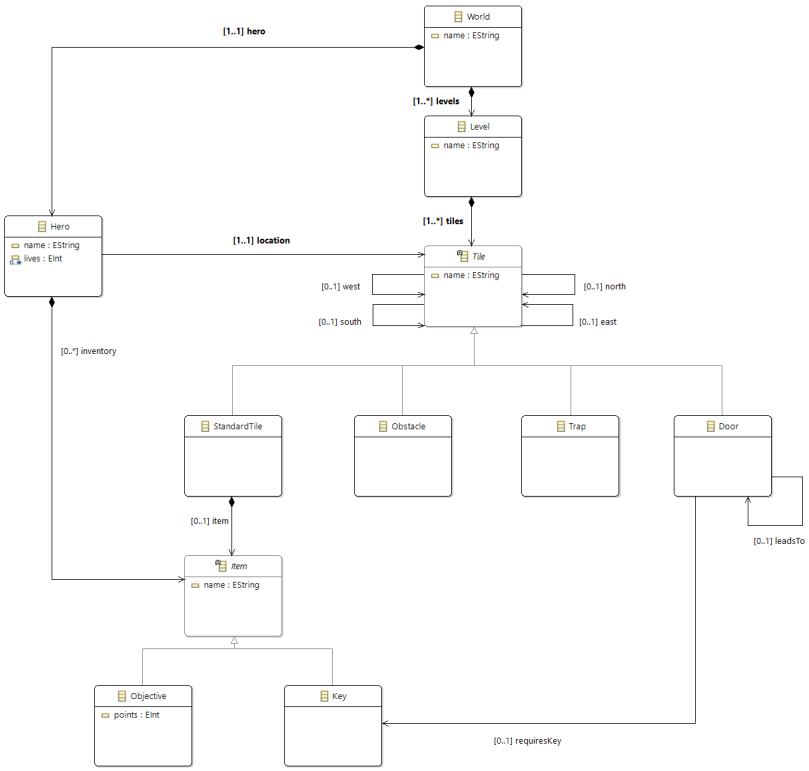


Figure 2: RPG Class Diagram

4. Instance RPG

Now that we have our Metamodels, we can create an instance of our source Metamodel (the source model), which we will later transform into the target model. The Eclipse Modeling Framework (EMF) facilitates this process by automatically generating a tree-based editor from the RPG.ecore definition. As shown in figure 4a, the interface allows the user to create objects and populate attributes and references, while ensuring that it conforms to the Metamodel. For the purpose of this study, I recreated the same model instance, visually depicted in Figure 4b, that is used in the assignments (3-6) of the MDE course.

5. Translation Logic

With the source and target metamodels defined, the core of this project lies in the ATL transformation logic. A single ATL file `RPG2PetriNet.atl`

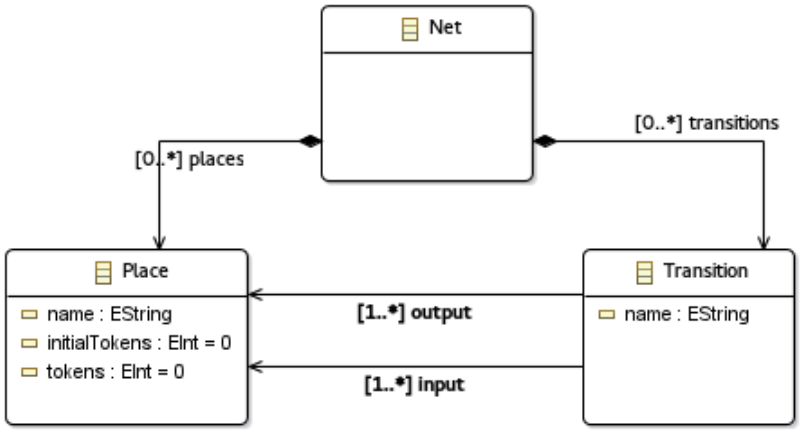


Figure 3: Petri Net Class Diagram

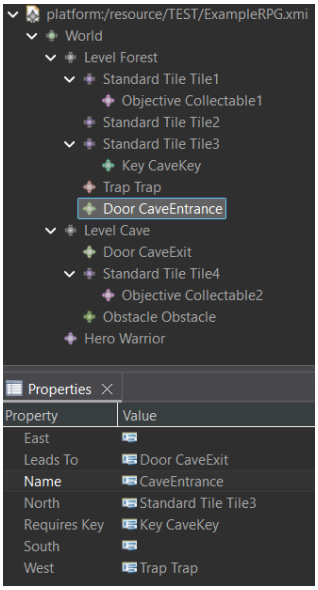
is created where rules are introduced to correctly transform from a source model to a conforming target model. Our goal is to make a Petri Net just like in assignment 6 of the MDE course. The structure of the Petri Net is shown in Figure 5. The process of this translation is divided into three parts: structural initialization, topological mapping, and interactive state management.

5.1. State Initialization and Helpers

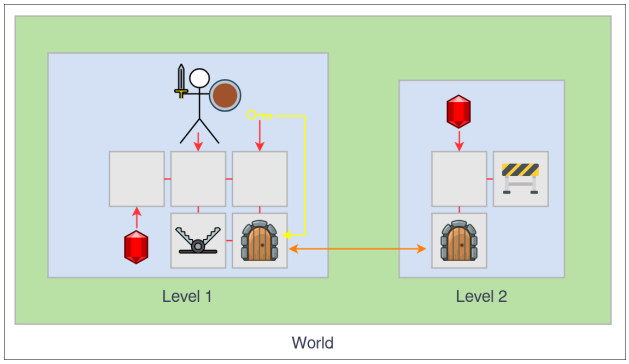
Before we start mapping elements, we need to specify the initial state of the system (ATLAS Group, 2006). We specify to which Metamodels both the source as the target model should conform to. In this case, it is `create OUT : PetriNet from IN : RPG`. Then, we have some helper functions we can later use to reduce code duplication. The helper `getHero()` retrieves the Hero from the input model. Another helper `hasHero()` checks if a specific tile contains the hero in the initial state. This will later be used to place a marking in the place of this tile.

5.2. Structural Mapping: The Container Rule

The entry point of the transformation is the rule `World2Net`. This rule maps the RPG!World to a PetriNet!Net container. In the right hand side of the rule, places and transitions are gathered as well. Specific collection strategies are used:



(a) The Model Instance of the RPG.



(b) The Model Instance visually shown.

Figure 4: Representation of the Source Model: (a) The Eclipse EMF Editor interface and (b) The conceptual visual representation.

- Places: It collects all places for tiles (except obstacles) and places items (one for where it is located and one for when it is collected). Additionally, it also gets the special place for the hero's lives.
- Transitions: Unlike standard transformations that map 1-to-1, connection transitions (movement) are generated conditionally. The rule filters potential connections using OCL selection statements (e.g., not t.north.oclIsUndefined()). This ensures that transitions are only created for valid, non-obstacle neighbors.

A challenge in this translation is preventing the generation of transitions for invalid paths. We do not want outgoing transitions from places which lead nowhere (if for example the tile has no neighboring tile in the north direction). Lazy rules are used to solve this problem. Unlike other rules, they are not executed automatically, giving us more control over which transitions can be created. In the main `World2Net` rule, these lazy rules (`MoveNorth`, `MoveEast`, `MoveSouth`, `MoveWest`) are called so that they are executed only when needed.

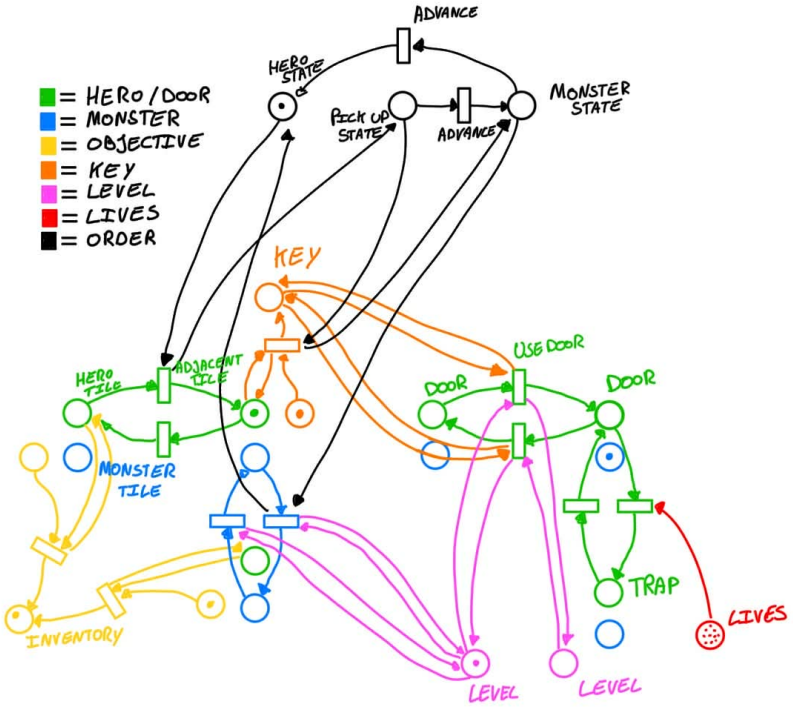


Figure 5: Petri Net Overview

```
lazy rule MoveNorth {
  from s : RPG!Tile
  to t : PetriNet!Transition (
    name <- s.name + '_to_North',
    -- INPUT: Check if destination (s.north) is a Trap
    input <- if s.north.oclIsTypeOf(RPG!Trap)
      then Sequence { s, thisModule.resolveTemp(thisModule.getHero(), 'p_lives') }
      else Sequence { s } endif,
    output <- s.north
  )
}
```

5.3. Interactive State Management

As shown above, the input consists of the tile s, and the output is a transition in Petri Nets. The logic also reveals another crucial game mechanic, namely Trap Detection. The rule first checks if the tile s is a Trap, if not, there are no additional input arcs (only s). In case it is a trap, we need to add an arc from the lives place to the transition. This way, when this transition is fired, one life will be consumed.

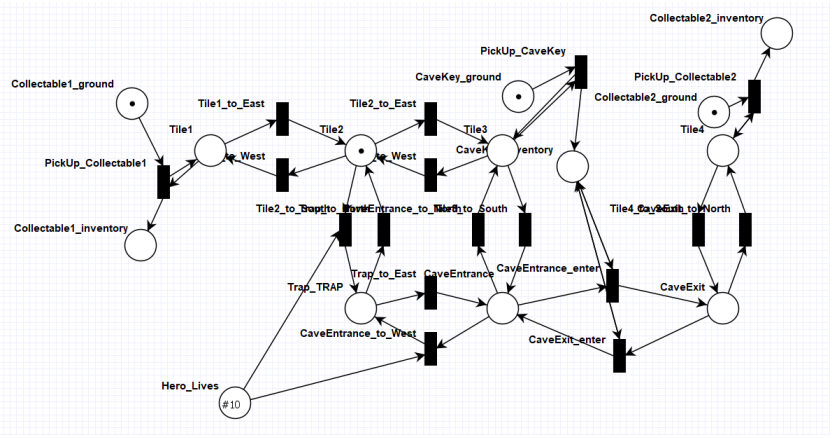


Figure 6: Resulting Petri Net

There are some other rules like `Item2PetriNet` where the logic for collecting items is defined, and `Door2Place` Where transitions from required keys are made + transitions between connecting doors.

6. Resulting Petri Net

The translation process outputs an xmi file. This is the target model in Petri Net style, generated from the source RPG model. As this format does not directly support visualization or execution, we convert it into a TAPN file using a script. The resulting file is then opened in TAPAAL, and after rearranging the places, we get a visually pleasing Petri Net shown in Figure 6. Notably, this structure follows the overview from Figure 5, which was used to visualize the Petri net for Assignment 6 of the MDE course.

7. Conclusion

The study successfully demonstrates the application of ATL to perform model-to-model transformations. Specifically, converting an RPG into an executable Petri Net. The Eclipse Modeling Framework provides a three-phase workflow of metamodeling, instantiation, and transformation. The key findings are:

- Hybrid Transformation Approach: ATL allows for a flexible combination of declarative rules for mapping and imperative logic for defining specific behaviors.

- Complex Logic Handling: The implementation effectively utilized OCL constraints and lazy rules to manage edge cases, such as preventing movement into non-existing areas and implementing logic for trap detection where tokens (lives) are consumed.
- Visual Validation: The transformation resulted in a valid Petri Net structure that correctly represented the topological mapping and interactive state management of the source game.

References

Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I., 2006. Atl-eclipse support for model transformation, in: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France, Citeseer.

ATLAS Group, 2006. Specification of the ATL Virtual Machine. URL: https://eclipse.dev/atl/documentation/old/ATL_VMSpecification%5Bv00.01%5D.pdf. version 00.01.

Van den Broeck, N., Stuer, R., . Mde project. GitHub Repository. URL: https://github.com/NielsVandenBroeck/MDE_Projects.

Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P., 2004. Modeling in the large and modeling in the small. European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004 3599. doi:10.1007/11538097$_3$.

Exelmans, J., . muMLE: Tiny (meta-)modeling framework. GitHub Repository. URL: https://github.com/joeriexelmans/muMLE.

GEMOC Studio, . Petrinet v1 ecore metamodel. GitHub Repository. URL: https://github.com/gemoc/petrinet.