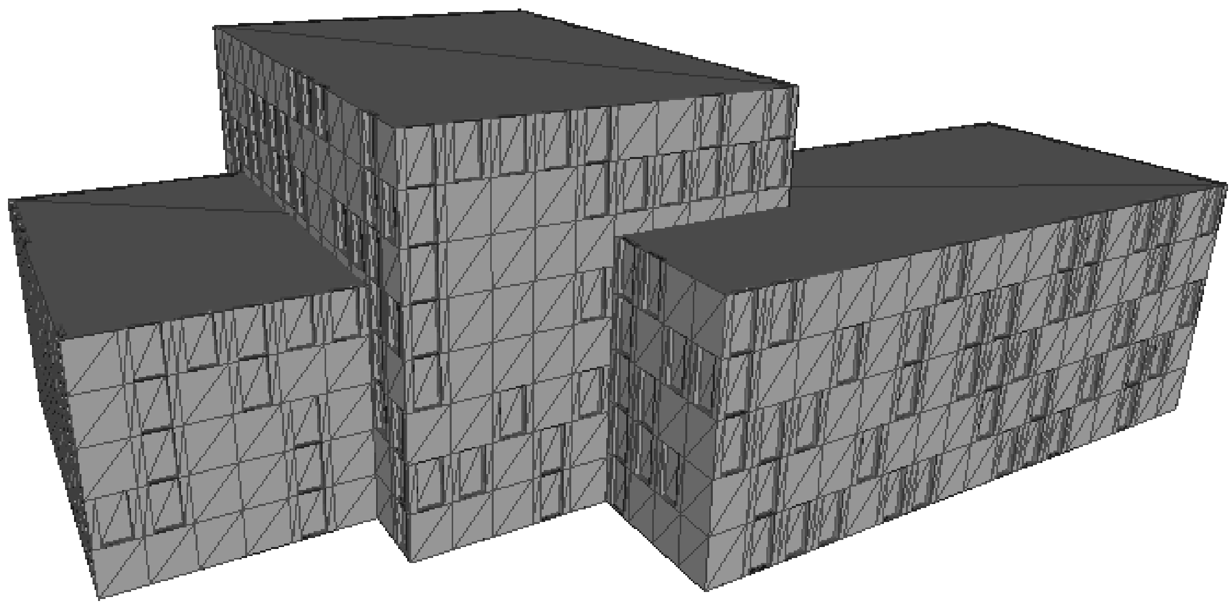
Model Driven Software Development Approach on Procedural Modeling of Buildings



Murat Kurtcephe

Oğuzcan Oğuz

Buğra M. Yıldız

Contents

Problem Statement

- Virtual cities should be modelled in order to be used in a bunch of applications.

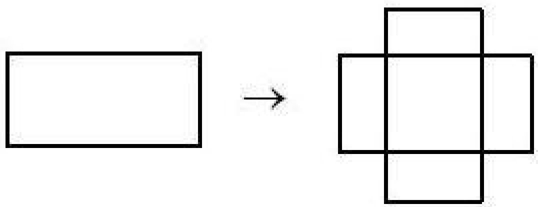
- Designing large number of building models requires extensive manual work.
- Procedural modeling: Building model generation with intended style and variation.
- **A domain specific language for procedural modeling of buildings.**
- **Model-to-Model transformations to provide better understanding and design of procedural modeling.**
- **Model-to-Text transformations to achieve portability for model generating grammars.**

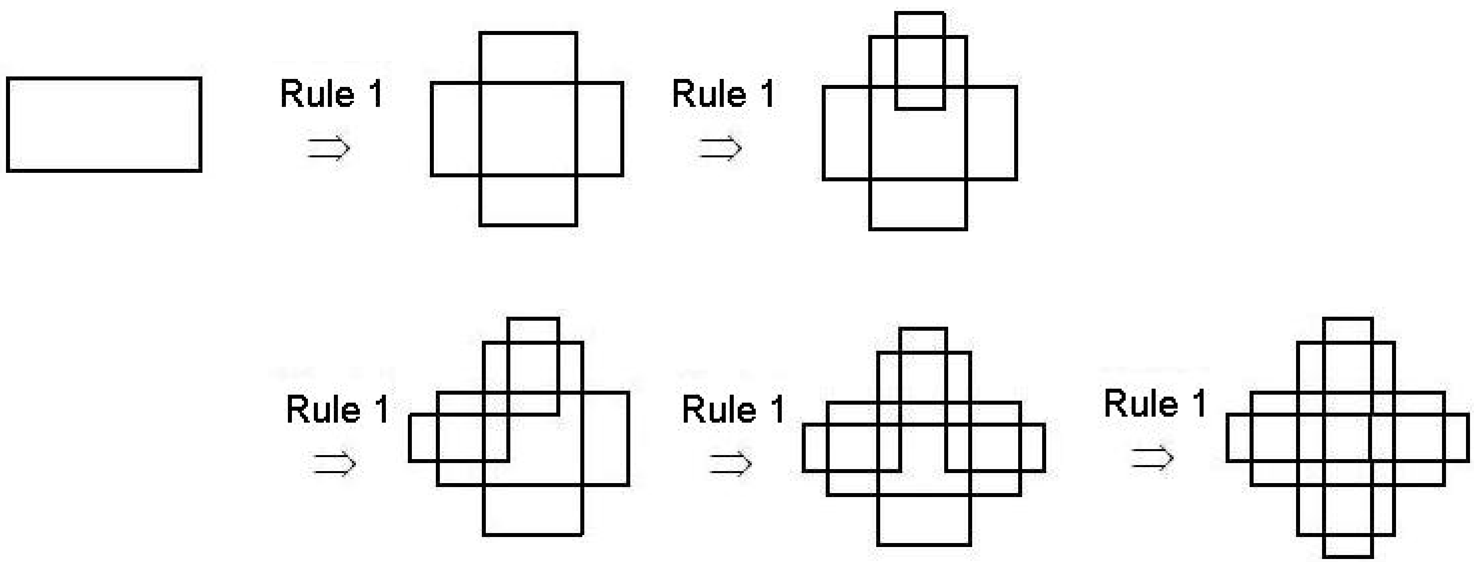
Shape Grammars

- Stiny G, Gips J, 1972
- Architectural Design with Shapes
- How does it work?
    1. Recognize a shape
    2. Replace the recognized shape with another shape
- Rules define which shape is replaced by which shape

Shape Grammar Example

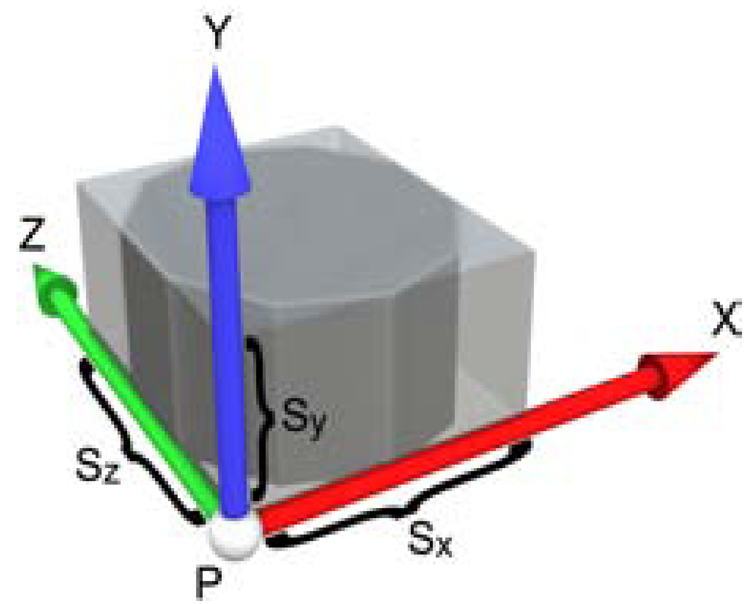
Rule 1:





Shape

- Terminal or Non-Terminal
- Non-Terminal shapes are applied to rules
- Terminal shapes have associated final geometry
- A smybol, numerical and geometric attributes
- **Scope:** An oriented bounding box: P, X, Y, Z and S



Production Rules

- Notation:

  id: predecessor : cond -> successor : prob

- Example:

  1: fac(h) : h>9 -> floor(h/3) floor(h/3) floor(h/3)

- Scope Rules: Translation, Scaling, Rotation, Insertion and Stacking:
  - 1: A -> [T(0,0,6) S(8,10,18) I(”cube”)]T(6,0,0) S(7,13,18) I(”cube”) T(0,0,16) S(8,15,8) I(”cylinder”)

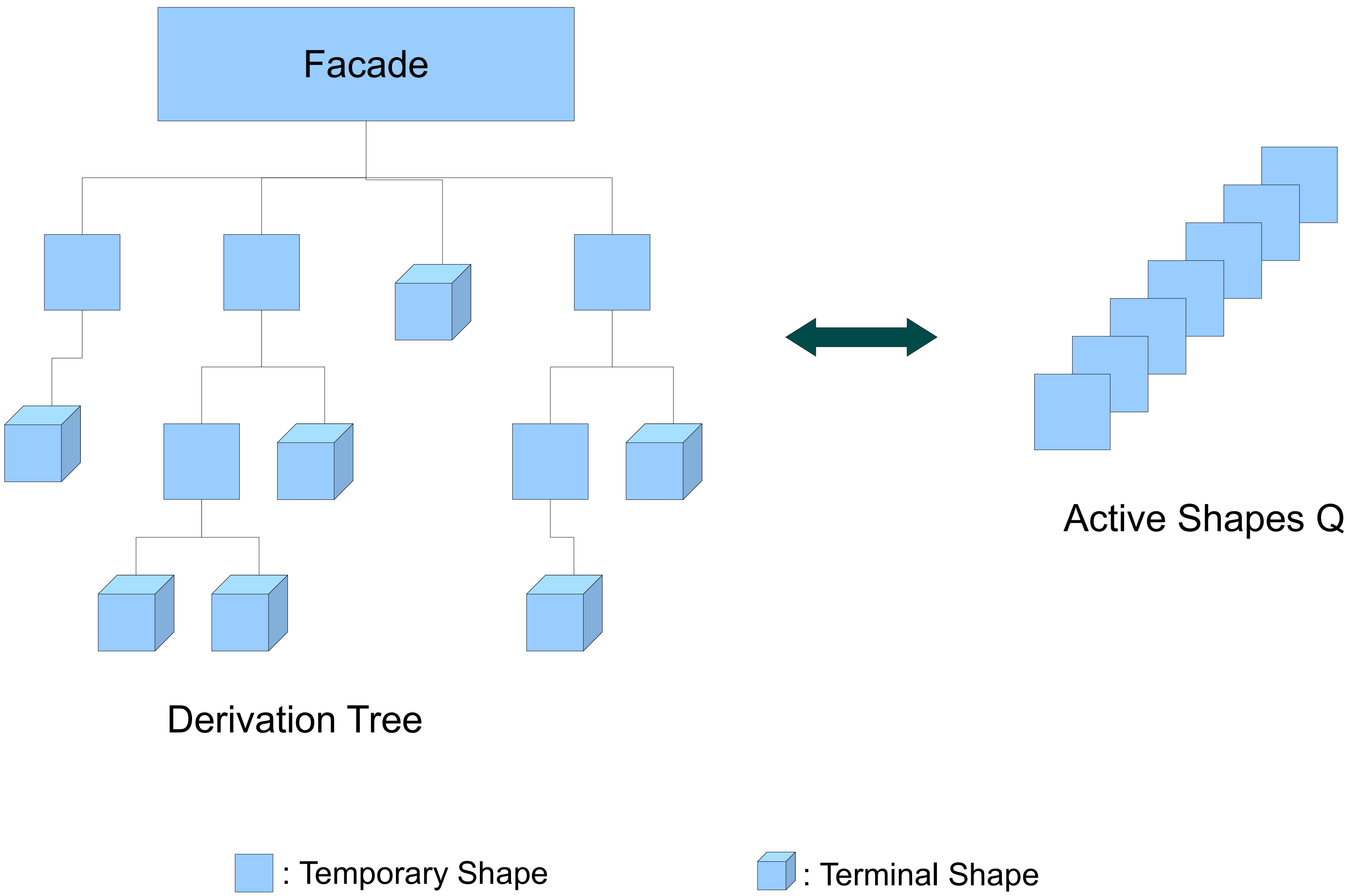
Production Rules

- Split Rules: Splits the given scope, and derives new shapes:
  - 1: floor -> Subdiv(”X”,2,1r,1r,2) {B | A | A | B}
- Repeat Rules: Repeats a successor shape in the scope of the given shape:
  - 1: floor -> Repeat(”X”,2) {B}
- Component Split Rules: To split into shapes of lesser dimension
  - 1: a -> Comp(“edge”, 3, 7) {A | B}

Production Process

- Configuration: A set of finite shapes (a runtime concept, not included in abstract syntax)
- Model Generation Process:(a runtime concept, not included in abstract syntax)
    1. Select an active shape with symbol *B* in the configuration
    2. Choose a production rule with *B* on the left hand side to compute a successor for *B*, a new set of shapes *BNEW*
    3. *Mark the shape B as inactive and add the shapes BNEW to the configuration and continue with step (1). When the configuration contains no more non-terminals, the production process terminates.*
- Priority sets to control traversal

Model Derivation



Grammar

ShapeGrammar ::= PrioritySet (PrioritySet)*

PrioritySet ::= PriorityID ProductionRuleList

PriorityID ::= Natural Number

ProductionRuleList ::= ProductionRule (ProductionRule)*

ProductionRule ::= RuleID NonTerminalShape [Condition] SuccessorList

RuleID ::= Natural Number

Condition ::= Boolean Expression

SuccessorList ::= (Successor Probability, SuccesorList) (Successor Probability, SuccesorList)*

Successor Probability ::= Float

Successor ::= Shape | ComponentSplitRule | RepeatRule | ScopeRule |
BasicSplitRule

...

The Abstract Syntax of Shape Grammar Meta-Model

- Meta-model from scratch, using Ecore
- Meta-model using UML 2.0 profiling mechanism

Static Semantics

- context RotationRule inv:

  self.angle >=0 and self.angle<=360
- context ComponentSplitRule inv:

  if self.paramList->size() = 0 then

      self.paramShapes->size() = 1

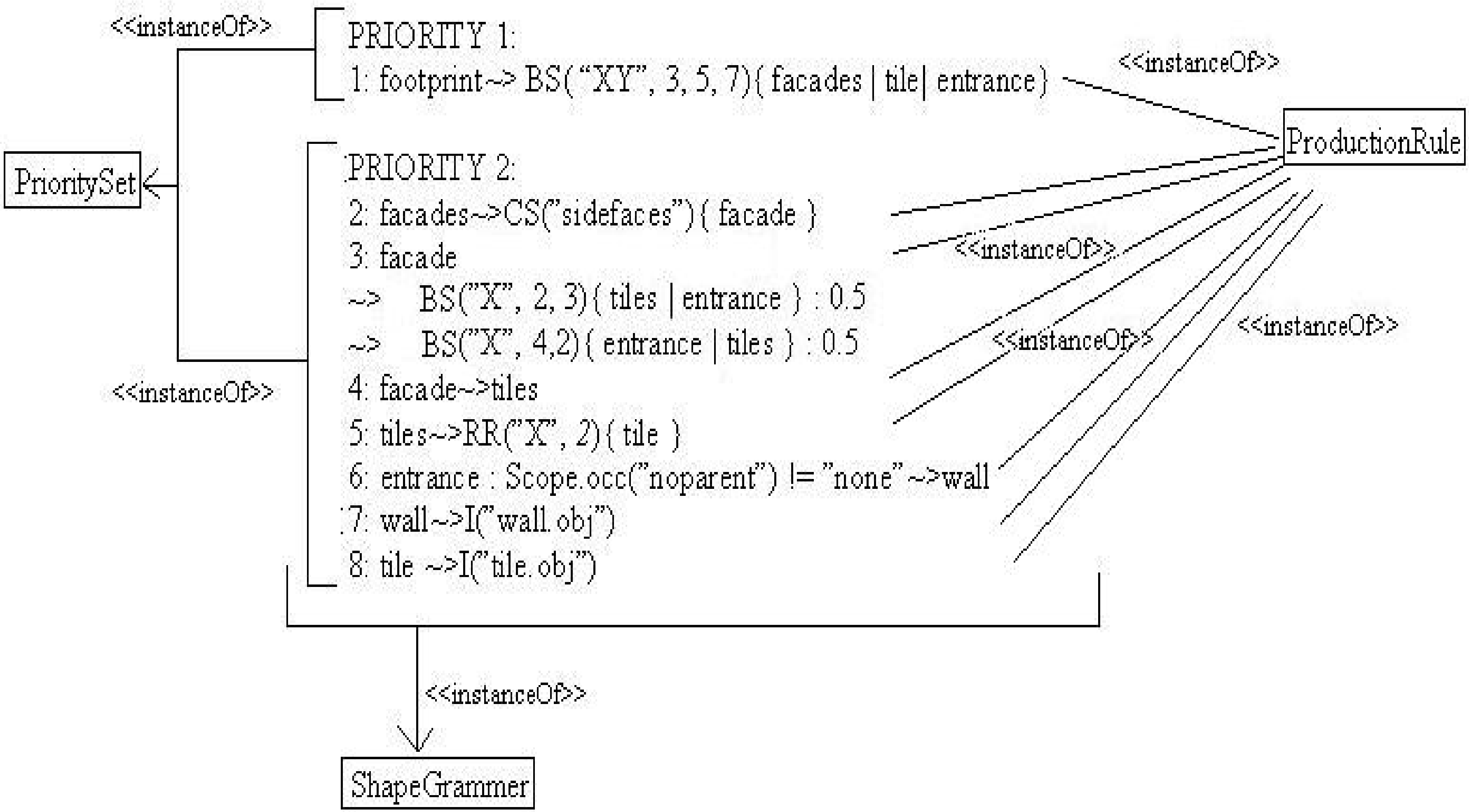
  else

      self.paramList->size() = self.paramShapes->size()

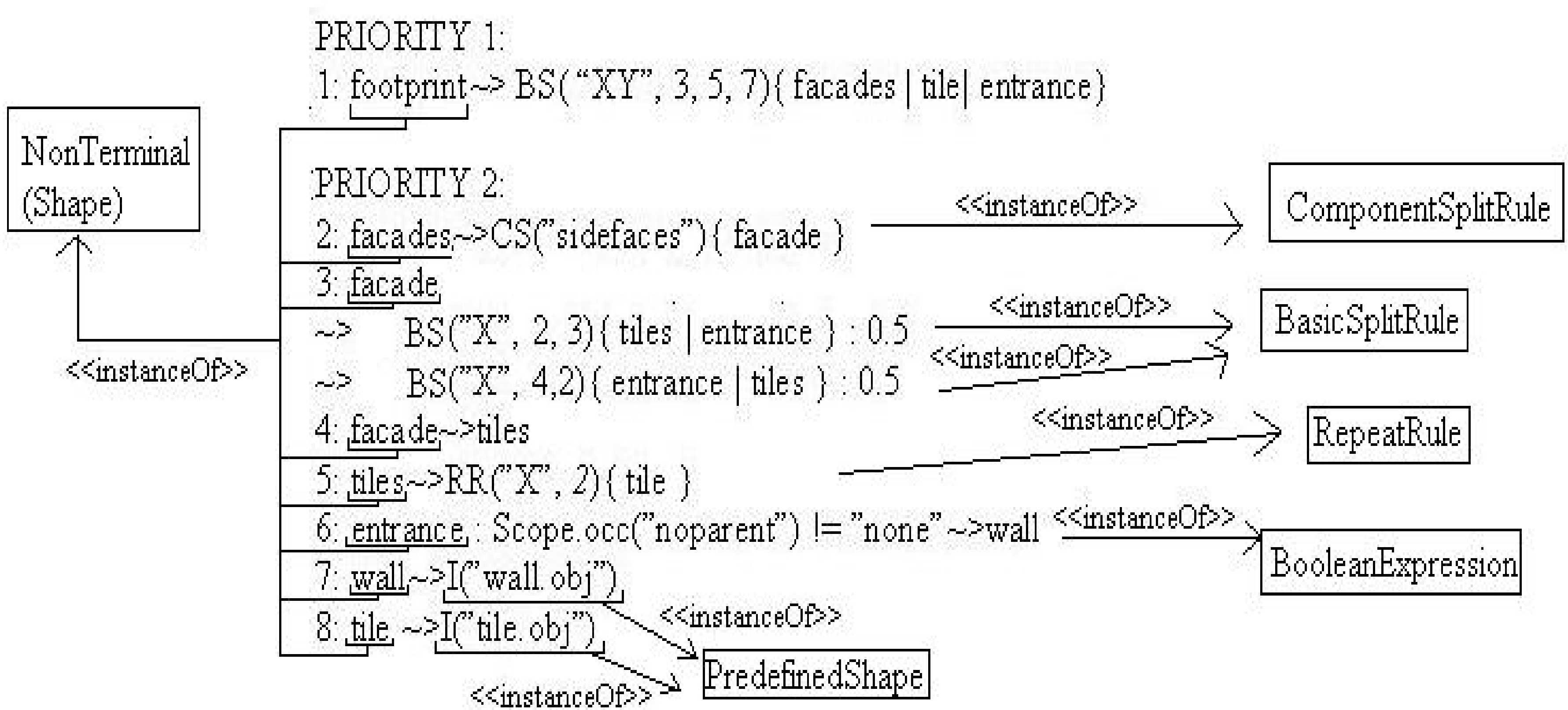
  endif
- context RulePart inv:

  self.probabilty <=1 and self.probability >0

Concrete Syntax



Concrete Syntax



Meta-modeling Issues

- Do we need to represent and how to represent geometric and numerical attributes?
  - If we don't, how to use a parameter of a shape to define a condition?
  - **Scope** is actually only needed at runtime(M0).
  - It seems, for full automation, we need to define the set of all possible geometries.
- Tools still not adequate: Associating stereotypes, Switched btw. 3 different tools, plugins (Eclipse, Topcased, Papyrus)
- The UML Profile and the ECore meta-model not so different.
- A confusing fact: For our domain, an M1 model is also like a grammar(A shape grammar).

Meta-modeling Conclusions

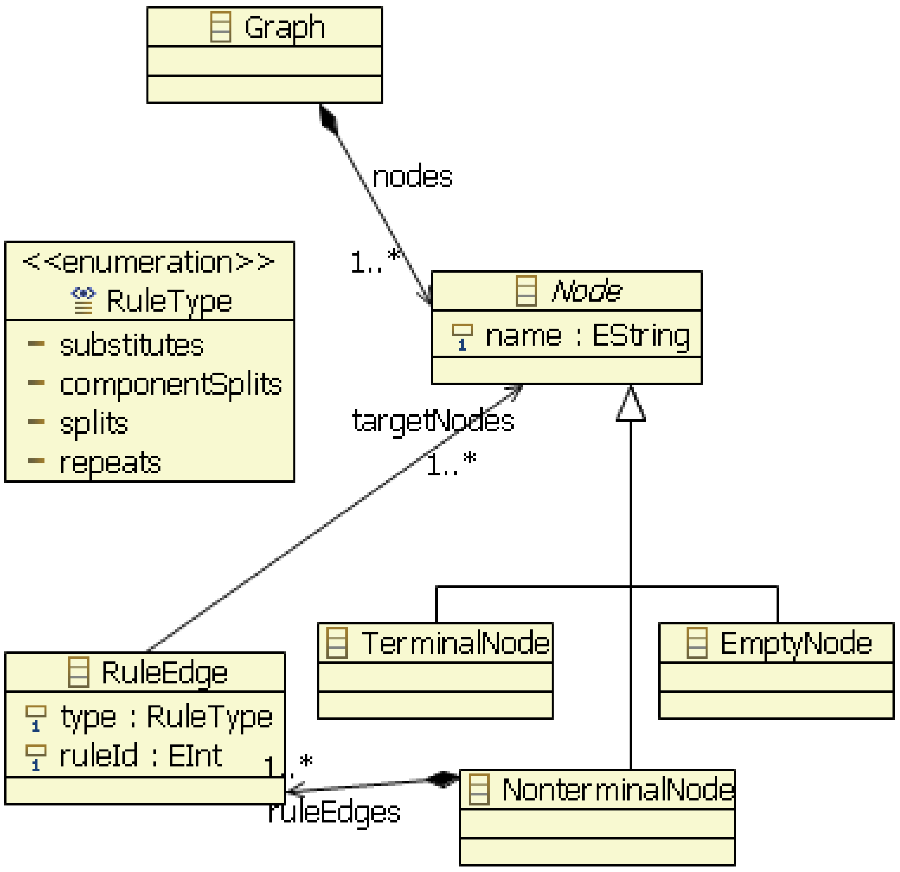
- The major limitation of grammars, for the selected domain, is the limitation of resulting tree structure: A shape cannot be connected to a number of rules.
- Using standards results in a high level tool support; had difficulty first but impressed later.
- For full automation, the meta-model needs to be coherently specified as much as possible.
- Extensive profiling needs adequate UML knowledge. But very suitable for OO similar domains.

Model Transformation

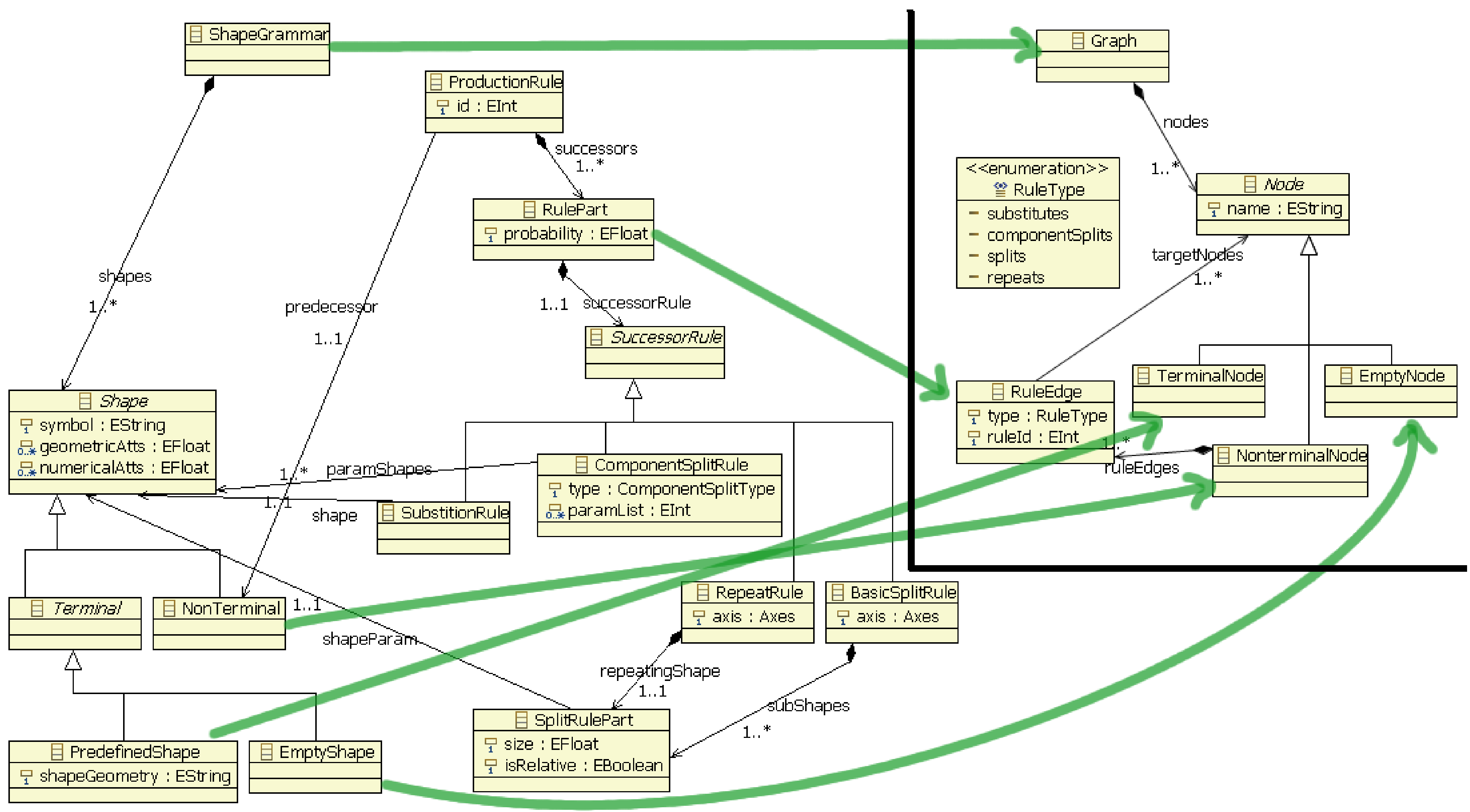
- Model to Model Transformation Motivation:
  - Understandability and productivity are increased by transforming the models into simpler and more expressive models: Derivation Graphs
- Model to Text Transformation Motivation:
  - Automatic code generation: Shape Grammars are output as XML files, and can be ported to existing model generation tools.
  - Productivity is promoted since DSL is used to define models in a higher level; manual modification reduced, automatic production is achieved.

Transformation to Derivation Graphs

- A Derivation Graph simply tells which shapes generate which shapes during the derivation process
- Non-terminal nodes have a number of rule edges which are connected to the nodes that the non-terminal node generates.
- Rule edges captures the rule properties coming from the input shape grammar model.



Transformation Specification



M2M Transformation in ATL

```
module Shape2Graph; -- Module Template
create OUT : DerivationGraph from IN : ShapeGrammar;

--transform the root: ShapeGrammar -> Graph
rule root {
    from
        s : ShapeGrammar!ShapeGrammar
    to
        t : DerivationGraph!Graph(nodes <- s.shapes)
}

--given a RulePart, get derived Shapes
helper context ShapeGrammar!RulePart def : getTargetNodes() : Set(ShapeGrammar!Shape) =
    if self.successorRule.oclIsTypeOf(ShapeGrammar!SubstitionRule) then
        self.successorRule.shape
    else
        if self.successorRule.oclIsTypeOf(ShapeGrammar!ComponentSplitRule) then
            self.successorRule.paramShapes
        else
            if self.successorRule.oclIsTypeOf(ShapeGrammar!RepeatRule) then
                self.successorRule.repeatingShape.shapeParam
            else
                self.successorRule.subShapes->collect(s | s.shapeParam)->flatten()
            endif
        endif
    endif;

--transform the rules: RulePart -> RuleEdge
rule rules {
    from
        s : ShapeGrammar!RulePart
    to
        t : DerivationGraph!RuleEdge(
            ruleId <- s.refImmediateComposite().id,
                              • • •
```

Shape Grammar to Derivation Graphs Transformation Example

- Shape Grammar
  - Priority Set
    - Production Rule 0
      - Rule Part 0.0
        - Basic Split Rule X
          - Split Rule Part 2.0
          - Split Rule Part 3.5
    - Production Rule 1
      - Rule Part 0.0
        - Substition Rule
    - Production Rule 2
      - Rule Part 0.0
        - Component Split Rule faces
  - Non Terminal floor
  - Non Terminal corner
  - Empty Shape empty
  - Predefined Shape wall
  - Non Terminal window

Input "Shape Grammar" Model

- Graph
  - Nonterminal Node floor
    - Rule Edge splits
  - Nonterminal Node corner
    - Rule Edge componentSplits
  - Empty Node empty
  - Terminal Node wall
  - Nonterminal Node window
    - Rule Edge substitutes

Output "Derivation Graph" Model

Conclusions on M2M

- The M2M transformation should be defined as precisely as possible.
- Not all the information is transformed.
- Transformation needs to satisfy conformance of the output.
- ATL should be used declaratively to avoid unintended consequences caused by the virtual machine.
- Browsing the target model should be avoided.

Shape Grammar to XML

- XML file is ported to a building generation tool: Generating different kinds of buildings easily
- The generated models can be used for different aspects such as gaming, educational or architectural purposes.
- The capabilities of the target engine is limited wrt. the shape grammar meta-model. Need to check input model validity.

Output Code Specification

```
<BuildingGen>

  <Rules>

    <Facade>
      <Occluded_Element Occlusion="Full">...
      <R1>...
    </Facade>

    <Floor>...
    <Temp>...
    <Wide_Window>...
    <Window_Center>...
    <Narrow_Window>...
    <Balcony_1>...
    <Balcony_Center>...

  </Rules>

  <Terminals>
    <Wall/>
    <Window/>
    <Balcony/>
  </Terminals>

</BuildingGen>
```

Transformation Specification

- Xpand is used for writing the templates.
- Xpand finds all non-terminal shapes which are going to be converted to new shapes and generates the conversion rules associated with non-terminal shapes.
- For a clear representation, an XML beautifier is invoked after the transformation.

M2T Transformation in Xpand

```
«IMPORT ShapeGrammar»

«DEFINE main FOR ShapeGrammar»
    «FILE "test.xml"»<?xml version="1.0" encoding="utf-8"?>
    <BuildingGen>
        <Rules>
            «EXPAND nonterminals FOREACH shapes-»
        </Rules>

        <Terminals>
            «EXPAND terminals FOREACH shapes-»
        </Terminals>
    </BuildingGen>
    «ENDFILE»
«ENDDEFINE»

«DEFINE nonterminals FOR ShapeGrammar::Shape-»
«IF this.metaType.name.compareTo("ShapeGrammar::NonTerminal")==0-»
<«symbol-»>
«IF((ShapeGrammar)eRootContainer).prioritySets.get(0).rules.exists(e|e.predecessor.symbol.compareTo(symbol)==0)-»
«LET ((ShapeGrammar)eRootContainer).prioritySets.get(0).rules.select(e|e.predecessor.symbol.compareTo(symbol)==0) AS ProRule-»
«EXPAND ruleProsess FOREACH ProRule-»
«ENDLET-»
«ENDIF-»
</«symbol-»>
«ENDIF-»
«ENDDEFINE»
```

Checking the Validity of the Input Model

- By using the check language the initiated model is checked before transformation

```
context EmptyShape ERROR
  "Empty shapes can not have a scope!" :
   scope == null;

context ShapeGrammar::ShapeGrammar ERROR
  "The default Facade object is not defined!" :
   shapes.exists(e|e.symbol.compareTo("Facade")==0);


context ShapeGrammar::SuccessorRule ERROR
  "Scope rules are not acceptable for this context!" :
   metaType.name.compareTo("ShapeGrammar::ScopeRule")!=0;

context ShapeGrammar::ShapeGrammar ERROR
  "There can be only one priority set for this context!" :
   prioritySets.size==1;


context ShapeGrammar::PrioritySet ERROR
  "All rules must have a predecessor shape!" :
   rules.forAll(e|e.predecessor!=null);
```

Conclusions on M2T

- Writing code generation templates involves extensive work.
- Templates not easy to use, unintuitive and not expressive enough. PL's designed for humans. These lead code failures.
  - Better template languages required or better
  - The 'T' layer should be skipped at all.
- Generated code is untidy, use beautifer.