Implementing FSML using JGralab

Marcel Heinz

January 30, 2014

Parts of the implementation

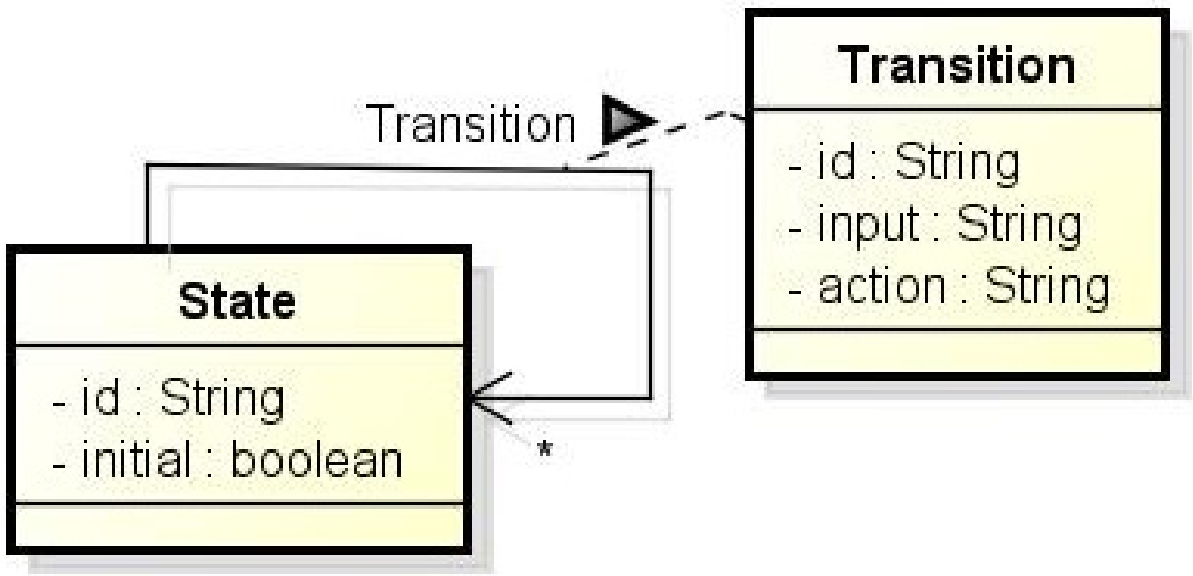
1. Abstract Syntax Definition using grUML
2. FSML Parser using ANTLR
3. Checking wellformedness via GReQL
4. Model Simulation by traversing the graph
5. Visualization using GraphViz

grUML overview

M3layer	MOF Metametamodel		*Metaschema layer*	grUML Metaschema
M2layer	Metamodel		*Schema layer*	Graph Schema
M1layer	Model		*Instance layer*	TGraph

Table : Vergleich zwischen UML und grUML

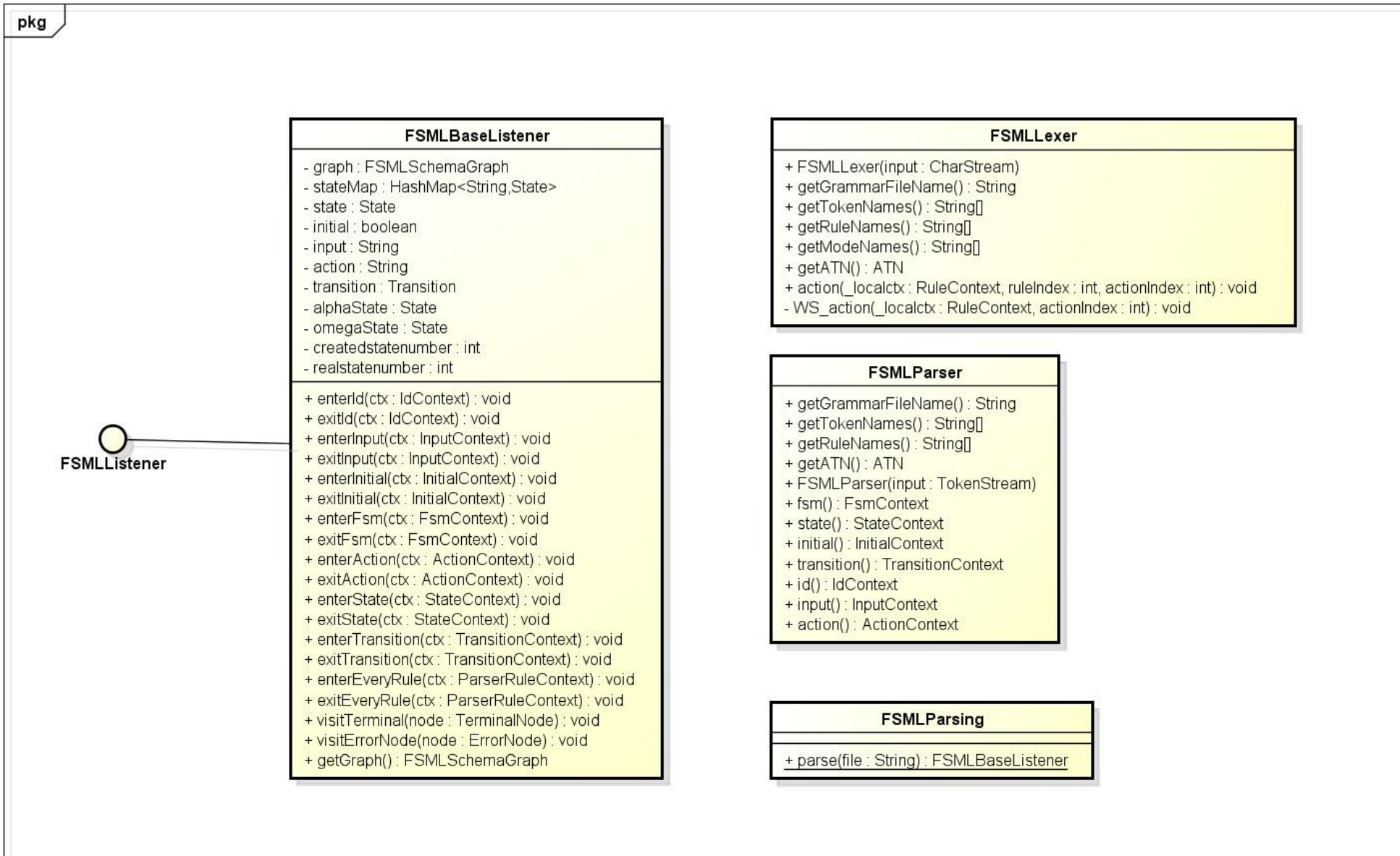
FSML's grUML-Schema



ANTLR grammar of FSML

```
grammar FSML;

fsm : state* ;
state : initial? 'state' id '{' transition* '}' ;
initial : 'initial' ;
transition : input ( '/' action )? ( '->' id )? ';' ;
id : Name ;
input : Name ;
action : Name ;
Name : Letter LetterOrDigit*;
WS : [ \t\r\n\u000C]+ -> skip;

fragment
Letter : [a-zA-Z$_];

fragment
LetterOrDigit : [a-zA-Z0-9$_];
```

FSMLBaseListener



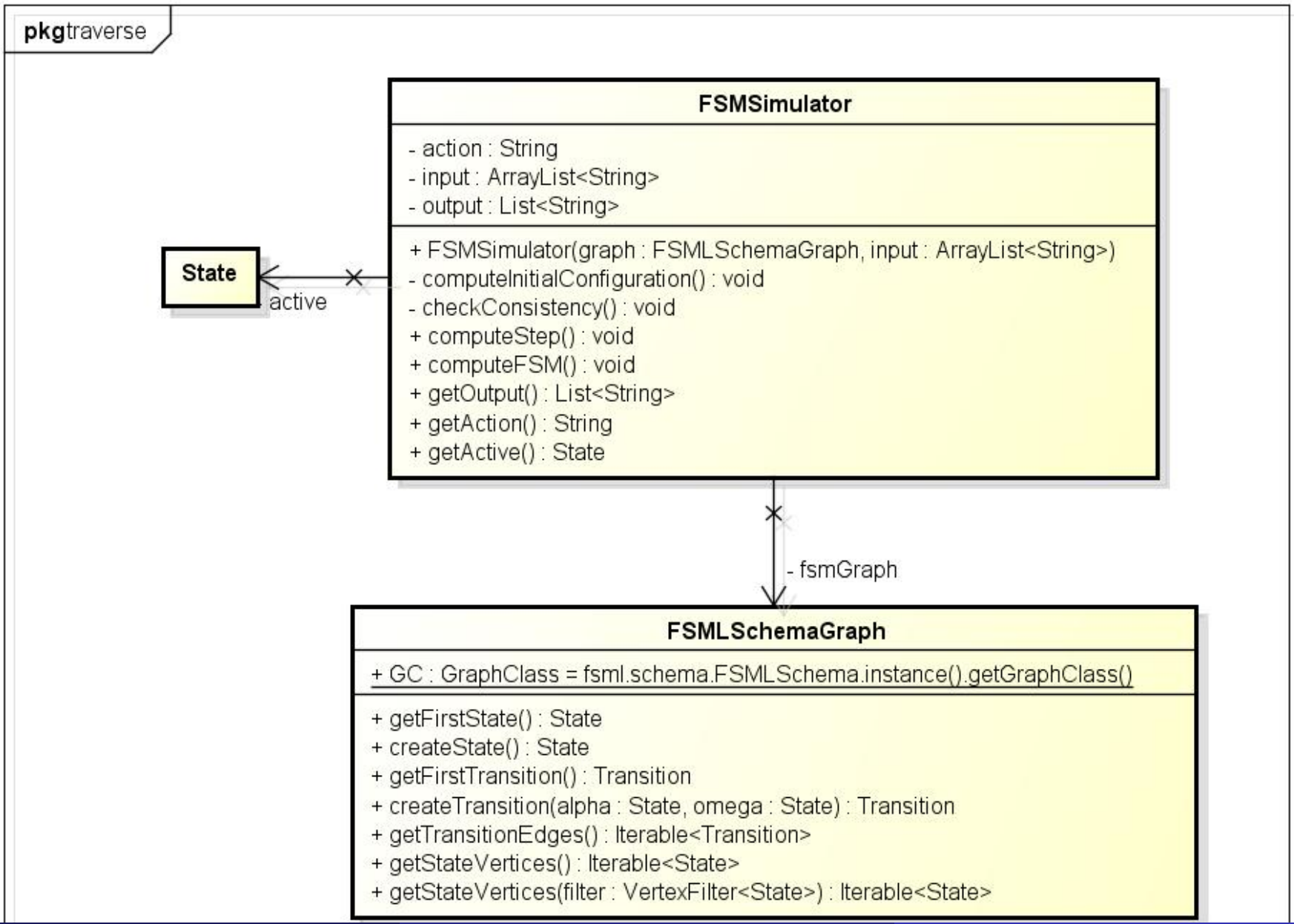
Wellformedness Check via GReQL (I)

```
//id resolvability
not(exists state : V{State} @ state.id = '' ) and
not(exists trans : E{Transition} @ trans.id = '');

//Reachability
forall state : V{State} @ exists statei : V{State}
@ statei.initial = true and statei -->{Transition}* state;

//Single Initial
exists statei : V{State}@ statei.initial = true and
not(exists state : V{State}@ state.initial = true and
state <> statei);
```

Wellformedness Check via GReQL (II)

```
//distinct IDs
forall state : V{State} @ not(exists state2 : V{State}
@ state2.id = state.id and state <> state2)

//Determinismus
forall trans : E{Transition} @ not(exists trans2 : E{Transition}
@ alpha(trans) = alpha(trans2) and trans.input = trans2.input
and trans <> trans2)
```

TravSimulator



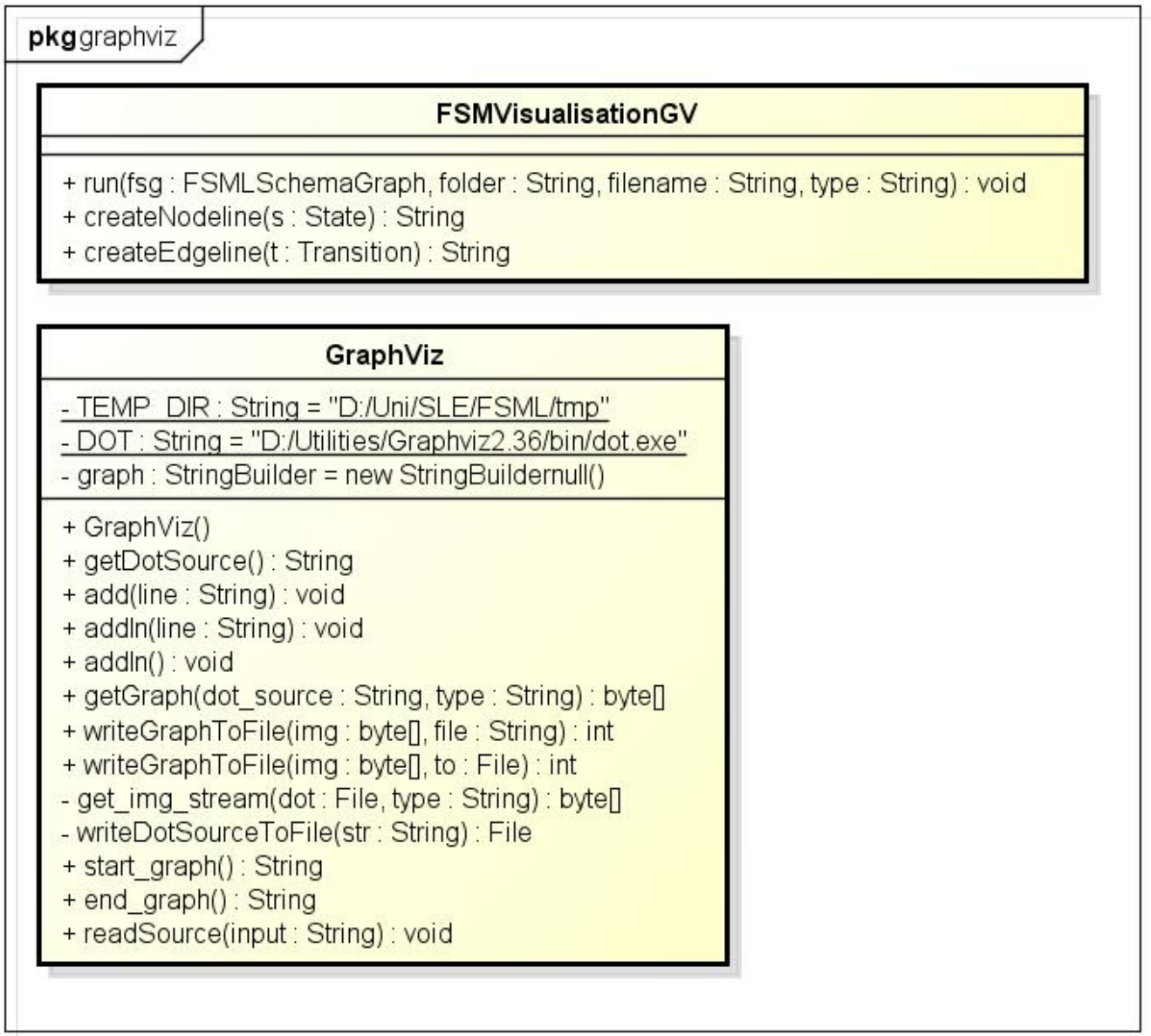
TravSimulator's step method

```
public void computeStep(){
  String trigger = input.get(0);
  Iterable<Edge> transitions = active.incidences(EdgeDirection.OUT);
  boolean transitionFound = false;
  for(Edge e : transitions){
    Transition t = (Transition) e;
    if(trigger.equals(t.get_input())){
      transitionFound = true;
      if(t.get_action().length()>0)
        action = t.get_action();
      else
        action = "";
      active = t.getOmega();
    }
  }
  if(!transitionFound)
    throw new InputMismatchException("The input contains " +
        "an invalid element:"+trigger);
  input.remove(0);
  output.add("Active = "+active.get_id() + "| Action = "+action);
}
```

SESimulator's AspectJ

```
public aspect FsmlSchemaExtensionGen {
  public Transition State.getNextTransition(String input){
    Iterable<Edge> transitions = this.incidences(EdgeDirection.OUT);
    boolean transitionFound = false;
    Transition t = null;
    for(Edge e : transitions){
      t = (Transition) e;
      if(input.equals(t.get_input())){
        transitionFound = true;
        break;
      }
    }
    if(!transitionFound)
      throw new fsml.exception.FSMTriggerNotFoundException(
        "The input contains an invalid element:"+input);
    return t;
  }
    .
    .
}
```

GraphViz Visualization



Demo

DEMO !

Test Generation

- There is no testmodelgenerator at JGralab.
- Lots of papers exist concerning EMF
- EMF-Models can be transformed into TGraphs
- $\Rightarrow$ Implementing a testgenerator

Meta-model coverage

1. Class Coverage: each meta-class is instantiated at least once
2. Association End Multiplicities: for each association extremity, each representative multiplicity must be covered
3. Class Attribute: for each attribute, each representative value interesting for the tester must be covered

$\Rightarrow$ Goal: Creating valid models, stress testing, creating invalid models to test constraintchecks

Input and Flow Coverage Criteria

1. All-States
2. All-Transitions
3. All-Inputs
4. Depth-n : Each run of length n from the initial state is considered in a test case.
5. All-n-Transitions: Each run of length n from any state is considered in a test case.
6. All-Paths: All possible transition sequences have to be included. $\rightarrow$ infeasible.

$\Rightarrow$ Goal: stress testing, validating execution