

Implementing FSML using JGralab

Marcel Heinz

January 30, 2014

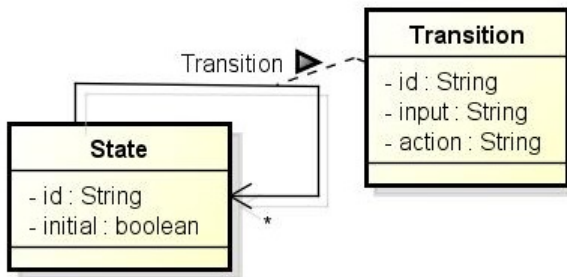
Parts of the implementation

- 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

<i>M3layer</i>	MOF Metametamodel	<i>Metaschema layer</i>	grUML Metaschema
<i>M2layer</i>	Metamodel	<i>Schema layer</i>	Graph Schema
<i>M1layer</i>	Model	<i>Instance layer</i>	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$_];
```

pkg

FSMLListener

FSMLBaseListener

```
- graph : FSMLSchemaGraph
- stateMap : HashMap<String,State>
- state : State
- initial : boolean
- input : String
- action : String
- transition : Transition
- alphaState : State
- omegaState : State
- createdstatenumber : int
- realstatenumber : int

+ enterId(cbx : IdContext) : void
+ exitId(cbx : IdContext) : void
+ enterInput(cbx : InputContext) : void
+ exitInput(cbx : InputContext) : void
+ enterInitial(cbx : InitialContext) : void
+ exitInitial(cbx : InitialContext) : void
+ enterFsm(cbx : FsmContext) : void
+ exitFsm(cbx : FsmContext) : void
+ enterAction(cbx : ActionContext) : void
+ exitAction(cbx : ActionContext) : void
+ enterState(cbx : StateContext) : void
+ exitState(cbx : StateContext) : void
+ enterTransition(cbx : TransitionContext) : void
+ exitTransition(cbx : TransitionContext) : void
+ enterEveryRule(cbx : ParserRuleContext) : void
+ exitEveryRule(cbx : ParserRuleContext) : void
+ visitTerminal(node : TerminalNode) : void
+ visitErrorNode(node : ErrorNode) : void
+ getGraph() : FSMLSchemaGraph
```

FSMLLexer

```
+ FSMLLexer(input : CharStream)
+ getGrammarFileName() : String
+ getTokenNames() : String[]
+ getRuleNames() : String[]
+ getModelNames() : String[]
+ getATN() : ATN
+ action(_localctx : RuleContext, ruleIndex : int, actionIndex : int) : void
- WS_action(_localctx : RuleContext, actionIndex : int) : void
```

FSMLParser

```
+ getGrammarFileName() : String
+ getTokenNames() : String[]
+ getRuleNames() : String[]
+ getATN() : ATN
+ FSMLParser(input : TokenStream)
+ fsm() : FsmContext
+ state() : StateContext
+ initial() : InitialContext
+ transition() : TransitionContext
+ id() : IdContext
+ input() : InputContext
+ action() : ActionContext
```

FSMLParsing

```
+ parse(file : String) : FSMLBaseListener
```

Wellformedness Check via GReQL (I)

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

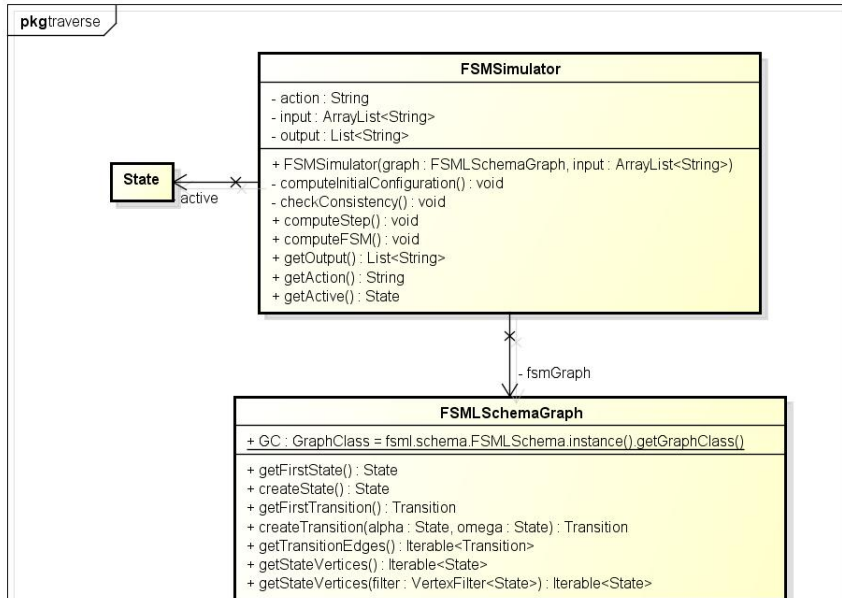
//Reachability
forall state : V{State} @ exists statei : V{State}
@ statei.initial = true and statei -->{Transition}* state;

//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'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

pkg graphviz

FSMVisualisationGV

```
+ run(fsg : FSMLSchemaGraph, folder : String, filename : String, type : String) : void  
+ createNodeline(s : State) : String  
+ createEdgeline(t : Transition) : String
```

GraphViz

```
- TEMP_DIR : String = "D:/Uni/SLE/FSML/tmp"  
- DOT : String = "D:/Utilities/Graphviz2.36/bin/dot.exe"  
- graph : StringBuilder = new StringBuildernull()  
  
+ GraphViz()  
+ getDotSource() : String  
+ add(line : String) : void  
+ addln(line : String) : void  
+ addln() : void  
+ getGraph(dot_source : String, type : String) : byte[]  
+ writeGraphToFile(img : byte[], file : String) : int  
+ writeGraphToFile(img : byte[], to : File) : int  
- get_img_stream(dot : File, type : String) : byte[]  
- writeDotSourceToFile(str : String) : File  
+ start_graph() : String  
+ end_graph() : String  
+ readSource(input : String) : void
```

DEMO !

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

- 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

⇒ 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