Model-driven Development of Discrete Event Simulations

Cem Mergenci
Kaan Onarlıoğlu
Uğur Aksu

Agenda

- Discrete Event Simulation
- Problem Statement
- Metamodel by Ecore
- UML Profiling
- Example Models
- Grammar
- M2T and M2M
- Lessons Learned

Discrete Event Simulation

- Simulation
  - Formally describing a real life system
  - Experiment and observe the behavior
- Discrete Event Simulation
  - Sequence of events
  - Operational perspective
  - Events modify states

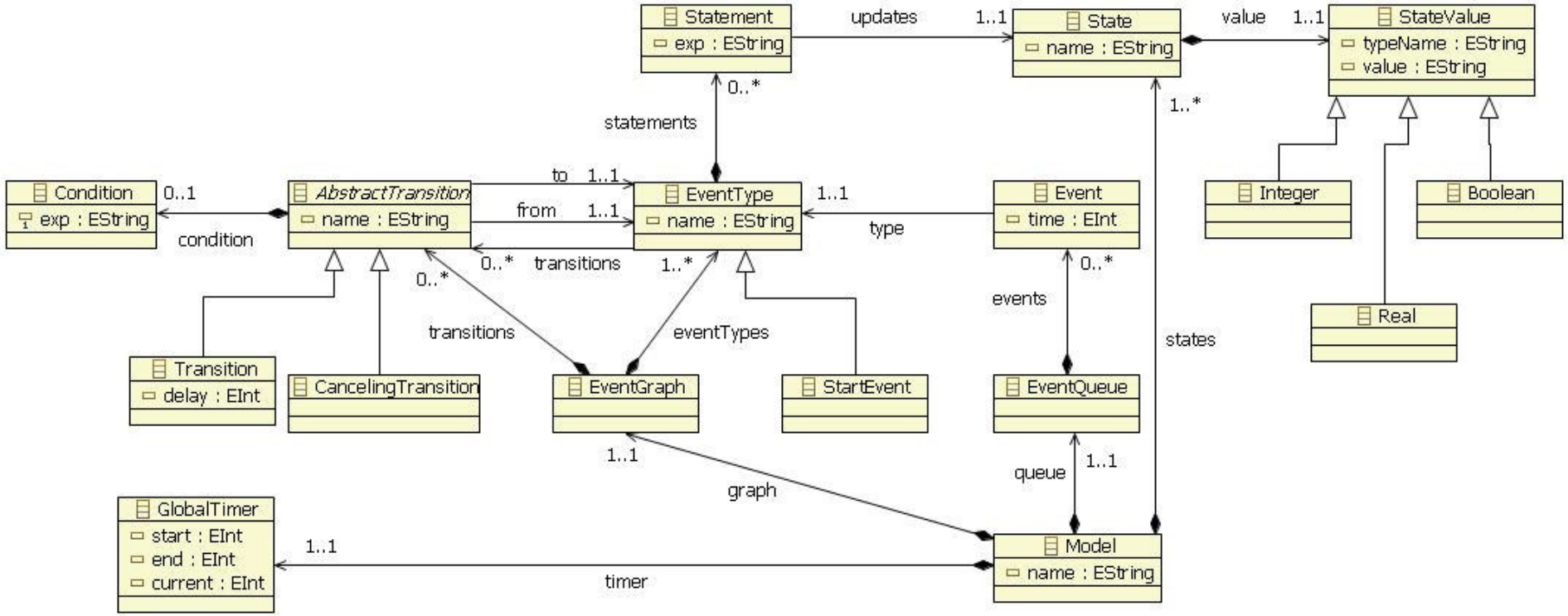
Problem Statement

- DES application frameworks
  - General programming paradigms and tools
  - Do not allow using domain concepts
  - Not interoperable
    - Have to stick to a single environment
- MDSD
  - Capture domain concepts
  - Transform to different models
  - Switch to different frameworks

Vocabulary

- Event
- State
- Event Scheduling & Canceling
- Global Timer
- Event Queue
- Ending Condition

Metamodel - Ecore

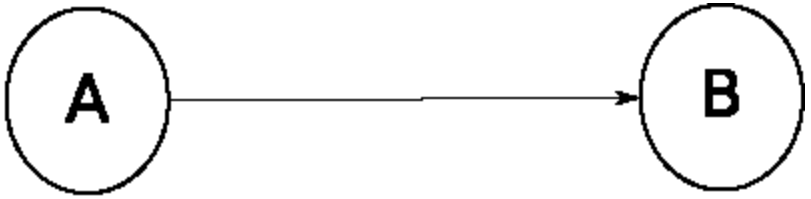


Static Semantics

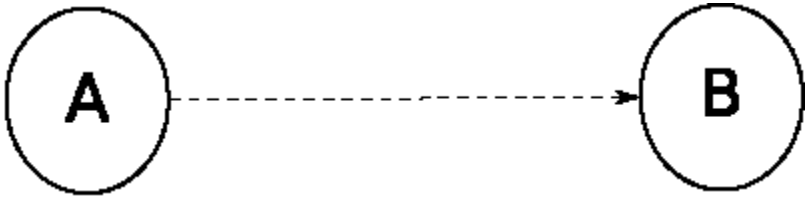
```
context StartEvent
inv: StartEvent::allInstances()->size()=1
inv: time = 0

context Simulation
inv: Event::allInstances()->forAll(e | e.time >= self.timer.current)

context Event
inv: Event::allInstances()->isUnique(name)

context Integer::value : Integer
init: value = 0
```

Concrete Syntax – Event Graph

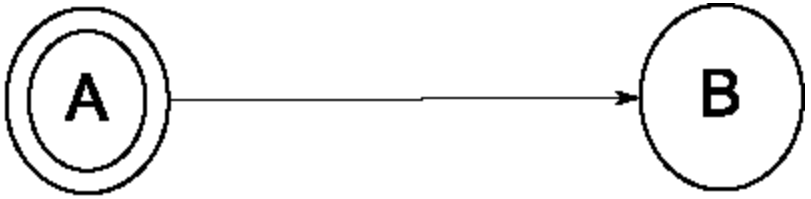
- Scheduling edge



- Canceling edge

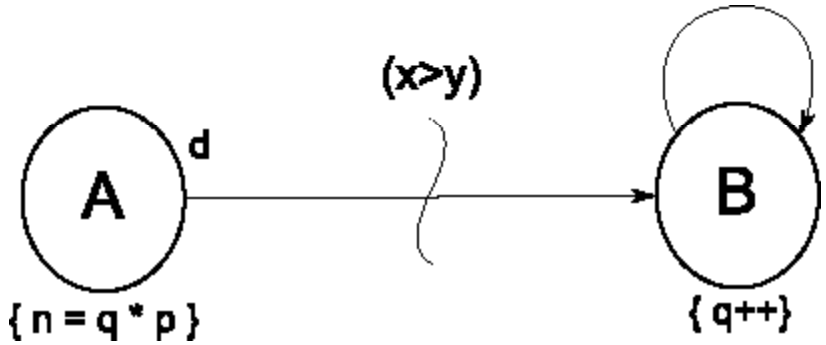


- Start event



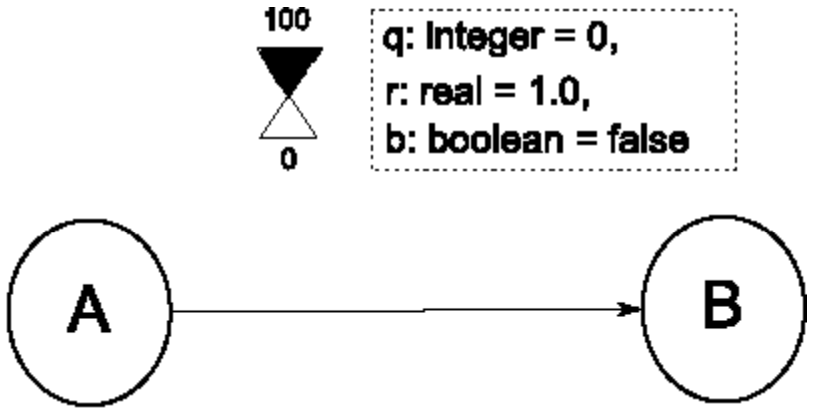
Concrete Syntax – Event Graph

- Conditional transition with delay
- Events with statements

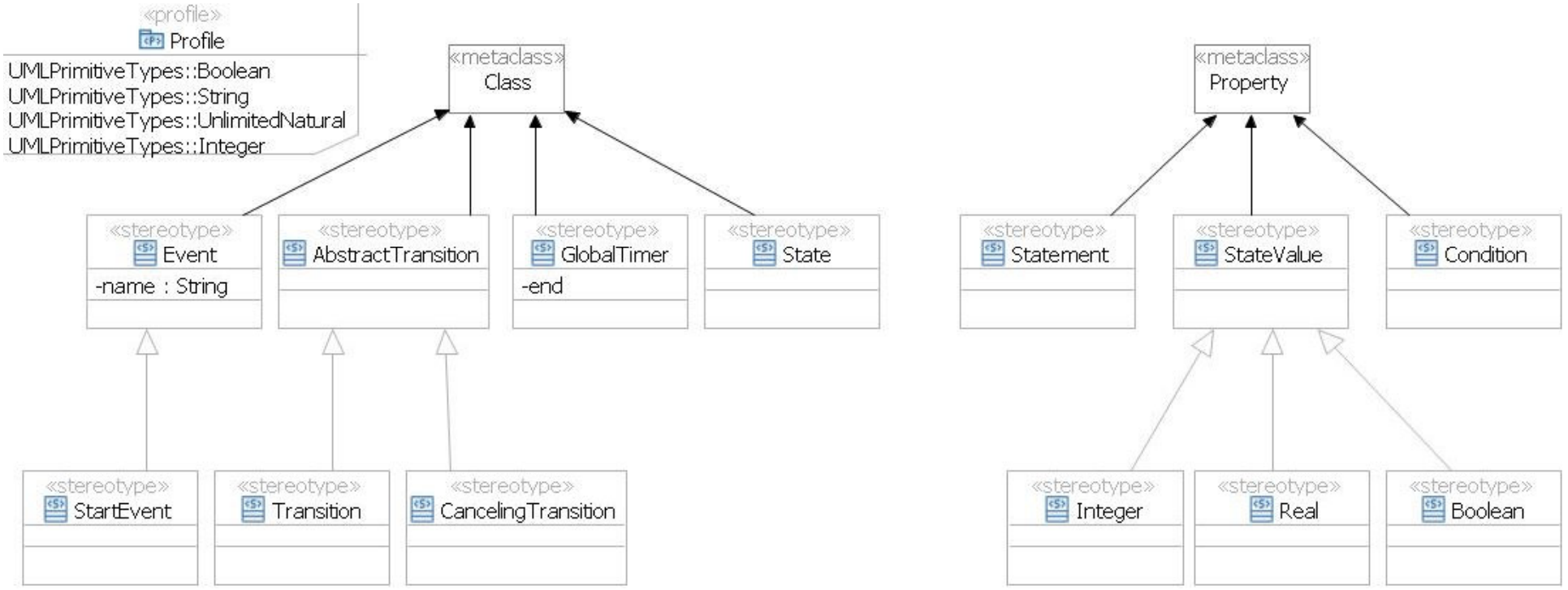


Concrete Syntax – Event Graph

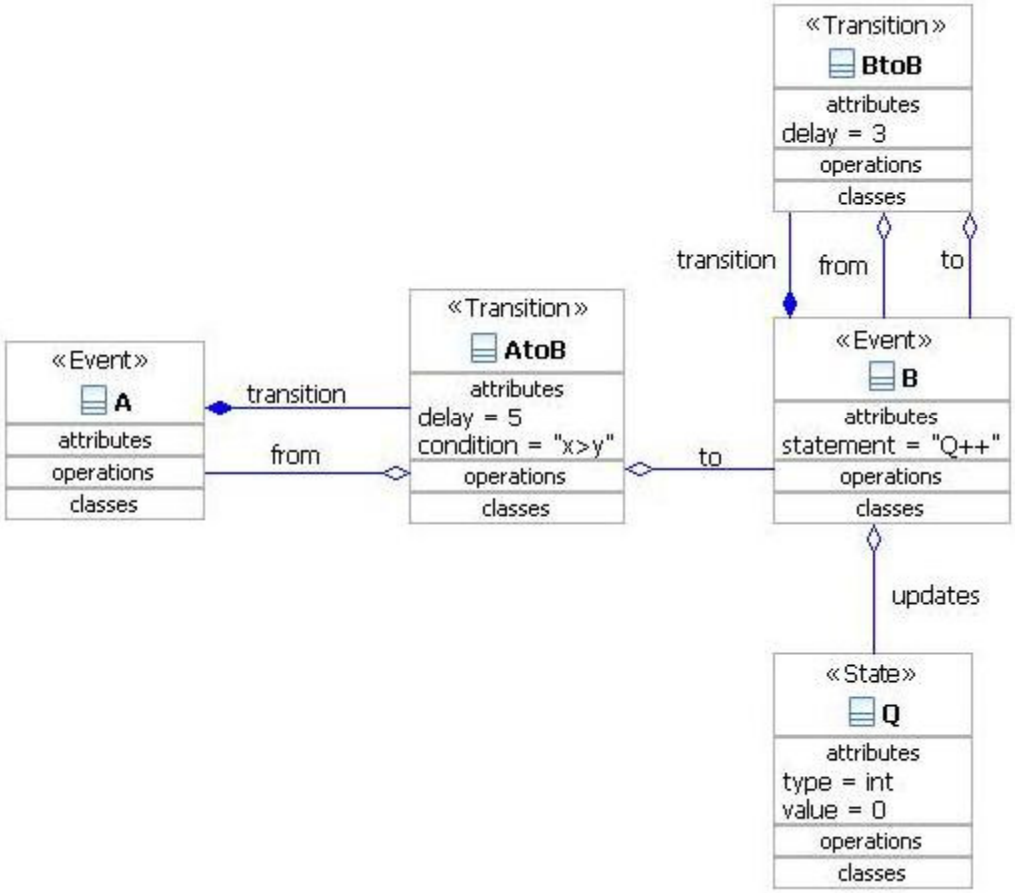
- Simulation time and state declaration



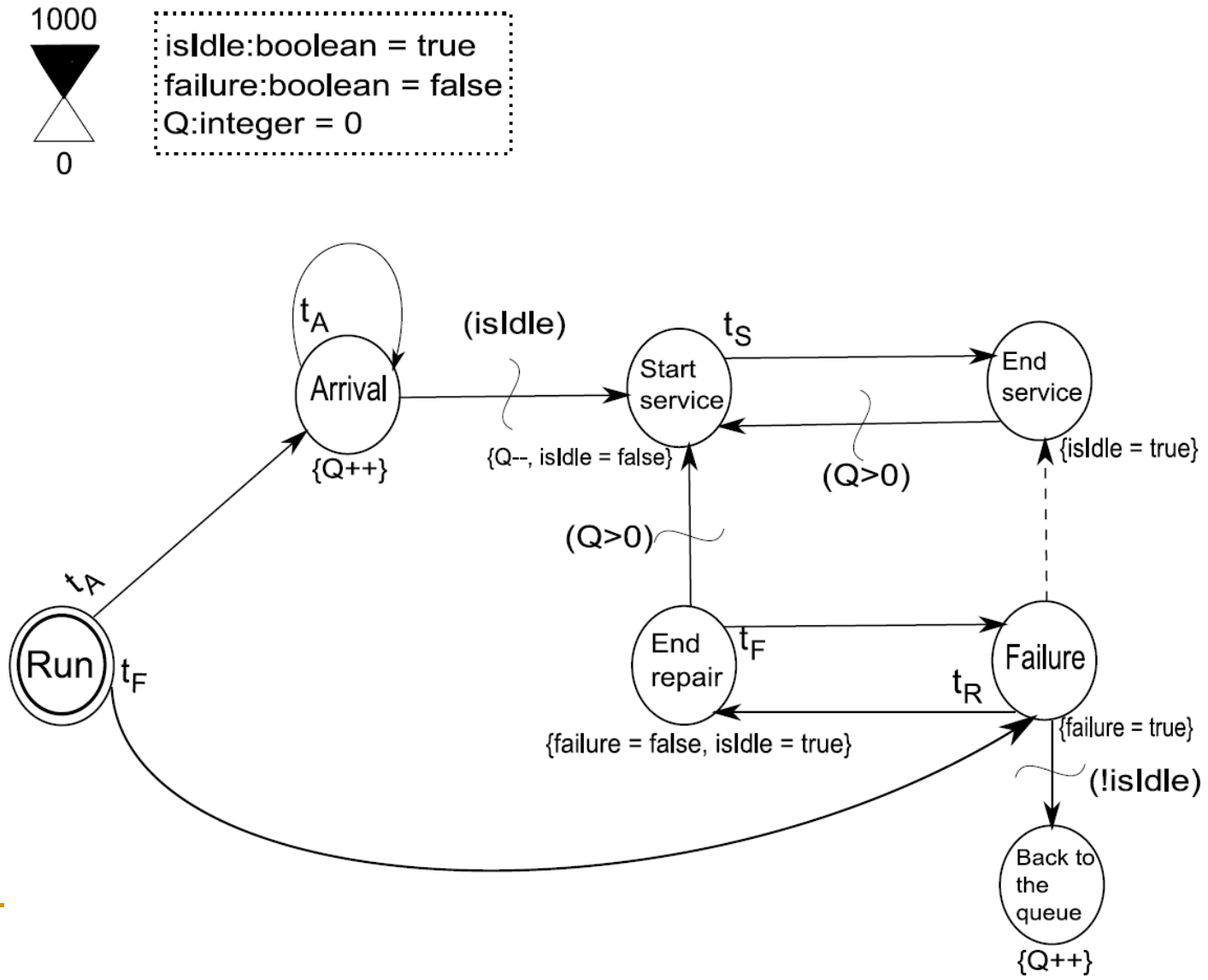
UML Profiling



UML Profiling Example



Car Washing Service Simulation



DSL Grammar

- Follow metamodel associations
  - Construct an EBNF grammar
- Also a textual concrete syntax

```
DESDeclaration = "DES", "varName", "(", "start", ":", integer, "end", ":", integer, ")", "{",
                 stateList, eventList, "}";

stateDeclaration = varName, ":", (("int", "=", integer) | ("real", "=", real) | ("bool", "=",
                 boolean));

stateList = "States", "{", stateDeclaration, {",", stateDeclaration}, "}";

eventList = startEvent, {event};

startEventDeclaration = "start", event;

eventDeclaration = "event", varName, "{" , [stateUpDate, ","], [eventScheduling,  ","],
                 [eventCancelling], "}";
```

Model-to-Text

- Transform to Java code using Xpand
  - SimKit framework
- *Model* → *SimEntityBase* Java class
- *State* → Protected property
- *Statement* → *firePropertyChanged* method
- *Transition* → *waitDelay* method
- *CancellingTransition* → *interrupt* method

Xpand

```
«IMPORT metamodel»

«EXTENSION template::GeneratorExtensions»

«DEFINE main FOR Model»
    «FILE name+".java"»
        import simkit.*;

        public class «name» extends SimEntityBase {
            «FOREACH states AS s»
                protected «s.value.typeName» «s.name»;

                public «s.value.typeName» «s.getter()»() {
                    return «s.name»;
                }
            «ENDFOREACH»

            «FOREACH graph.eventTypes AS e»
                public void do«e.name.toFirstUpper()»() {
                    «IF e.name == "Run"»
                        reset();
                    «ENDIF»
                    «FOREACH e.statements AS s»
                        firePropertyChange( "«s.updates.name»", «s.updates.name», «s.exp» );
                    «ENDFOREACH»

                    «EXPAND transitionClass FOREACH e.transitions»
                }
            «ENDFOREACH»

            public void reset() {
                super.reset();
                «FOREACH states AS s»
                    «s.name» = «s.value.value»;
                «ENDFOREACH»
            }
        }
    «ENDFILE»
«ENDDEFINE»
```

Model-to-Model

- Transform to DOT model using ATL
  - Graph visualization tool from GraphViz
- *EventType* → *Node*
- *Statement* → *NodeLabel*
- *Transition* → *DirectedArc*
- *Delay, Condition* → *TransitionLabel*

ATL

```
rule TransitionToDirectedArc
{
    from
        t: metamodel!Transition
    to
        out: DOT!DirectedArc (
            fromNode <- t.from,
            toNode <- t.to,
            arrowHead <- arrowHeadShape,
            arrowTail <- arrowTailShape,
            taillabel <- arcTailLabel,
            label <- conditionLabel
        ),

        arrowHeadShape: DOT!ArrowShape (
            name <- 'vee',
            isPlain <- false,
            clipping <- 'none'
        ),

        arrowTailShape: DOT!ArrowShape (
            name <- 'none',
            isPlain <- false,
            clipping <- 'none'
        ),

        arcTailLabel : DOT!SimpleLabel (
            content <- if t.delay > 0 then ''+t.delay else '' endif
        ),

        conditionLabel : DOT!SimpleLabel (
            content <- if t.condition.oclIsUndefined() then '' else t.condition.exp endif
        )
}
```

Lessons Learned

- Good domain analysis
  - Easy metamodeling
- Tools for metamodeling
  - Not many choices
  - EMF tools are complex and buggy
    - Get used to it…
- Grammar
  - Not so hard to construct
  - Also built a textual concrete syntax

Lessons Learned

- UML Profiling
  - Harder than metamodel from scratch
- M2T
  - Instant executable models!
- M2M
  - Need better tools

Conclusion

- DES domain suitable for MDSD
- Effective modeling with event graphs
- Easy to understand and visualize
- Platform independency with M2T
- Interoperability with M2M

Thanks

Q & A