Transforming a Graph Grammar Specification of FSM to Kermeta

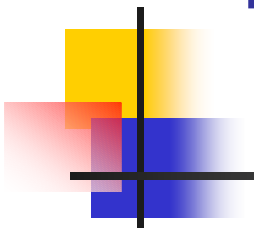
Jun Li

McGill University

References

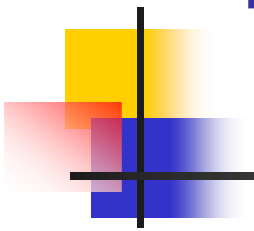
- Hans Vangheluwe and Juan de Lara. **Meta-models are models too.** In *Winter Simulation Conference*, pages 597 - 605. IEEE Computer Society Press, December 2002
- Eugene Syriani and Hans Vangheluwe. **Programmed Graph Rewriting with Time for Simulation-Based Design**. In: *International Conference on Model Transformation* (ICMT 2008). LNCS, Springer-Verlag, 2008.
- Eugene Syriani and Hans Vangheluwe. **Programmed Graph Rewriting with DEVS**. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007)*, LNCS, Springer-Verlag, October 2007.
- http://www.kermeta.org/
- Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel, Weaving executability into object-oriented meta-languages. Proceedings of MODELS/UML'2005, volume 3713 of LNCS,

Kermeta – Kernel Meta-modelling



- Executable meta-modelling language
- Meta-models structural specifications
  - Using class diagram
- Specify operational semantics / action specifications
  - Kermeta program (coding by hand)
  - Static semantic (similar to OCL)
    - Constraints: pre & post conditions, invariants
    - Define algorithmic specifications
  - Dynamic semantic: defining behaviours of the meta-model,
    - Data manipulation, simulation (in text mode)

Kermeta – Kernel Meta-modelling



- Kermeta is an extension to the Essential Meta-Object Facilities (EMOF) 2.0
- Built on the top of Eclipse Modelling Framework (EMF); transform from/to Ecore
  - Heavily rely on the EMF functionalities
- Object-Oriented (Eiffel, Java)
- Aspect-Oriented Modelling
- Imperative language
- Statically typed (100% type safe)
- Fully integrated with Eclipse; distributed as Eclipse plug-in, open-source (EPL)

Operations in Kermeta

- EMOF operation's definition

  Operation ***isInstance(element:Element):Boolean***

  "Returns true if the element is an instance of this type or a subclass of this type. Returns false if the element is null" [4]

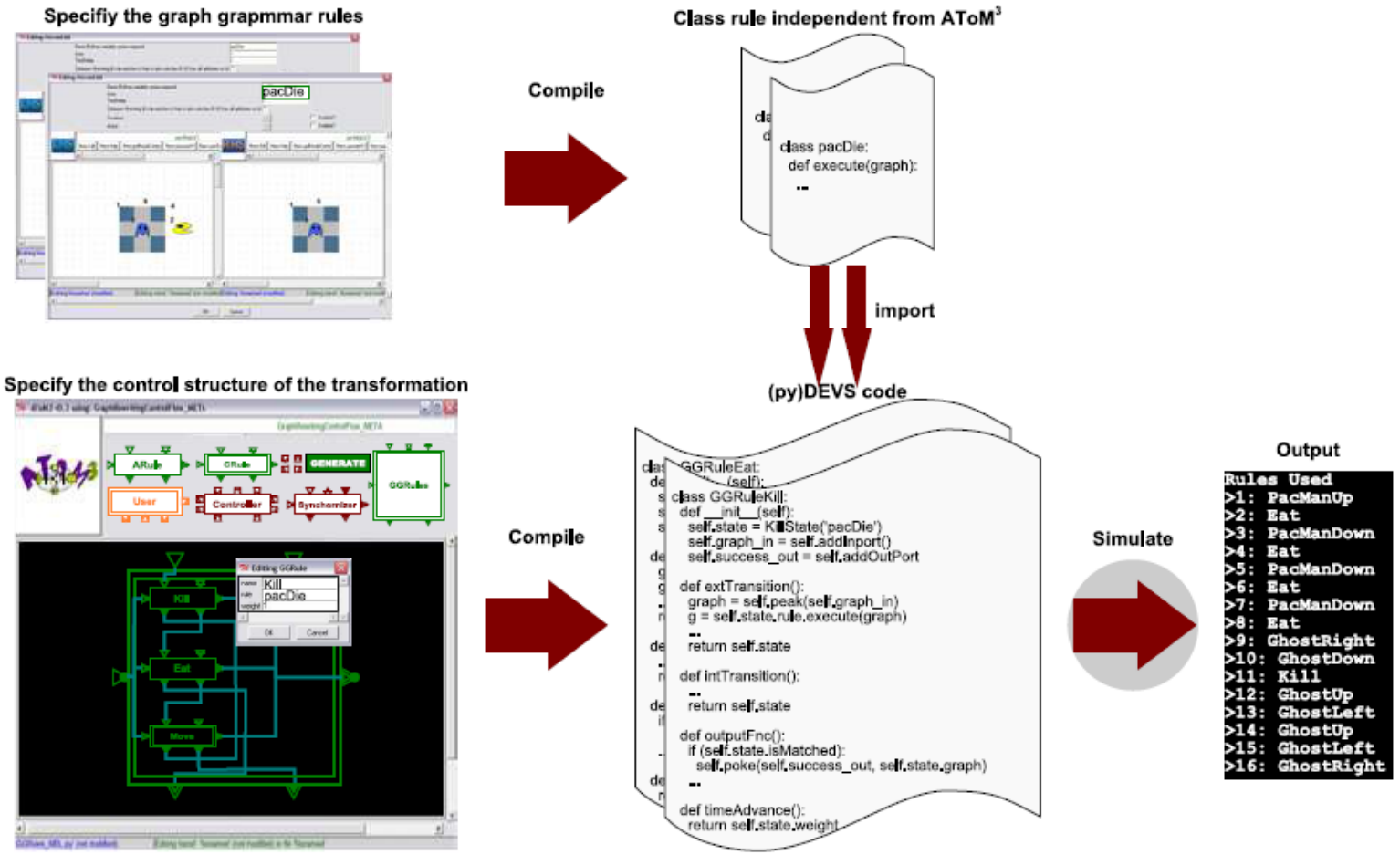
```
operation isInstance(element : Element) : Boolean is do
    // false if the element is null
    if element == void then result := false
    else
        // true if the element is an instance of this type
        // or a subclass of this type
        result := element.getMetaClass == self or
            element.getMetaClass.allSuperClasses.contains(self)
    end
end
```

Fig. 1. Executable specification of the *isInstance* operation of the EMOF *Type* class

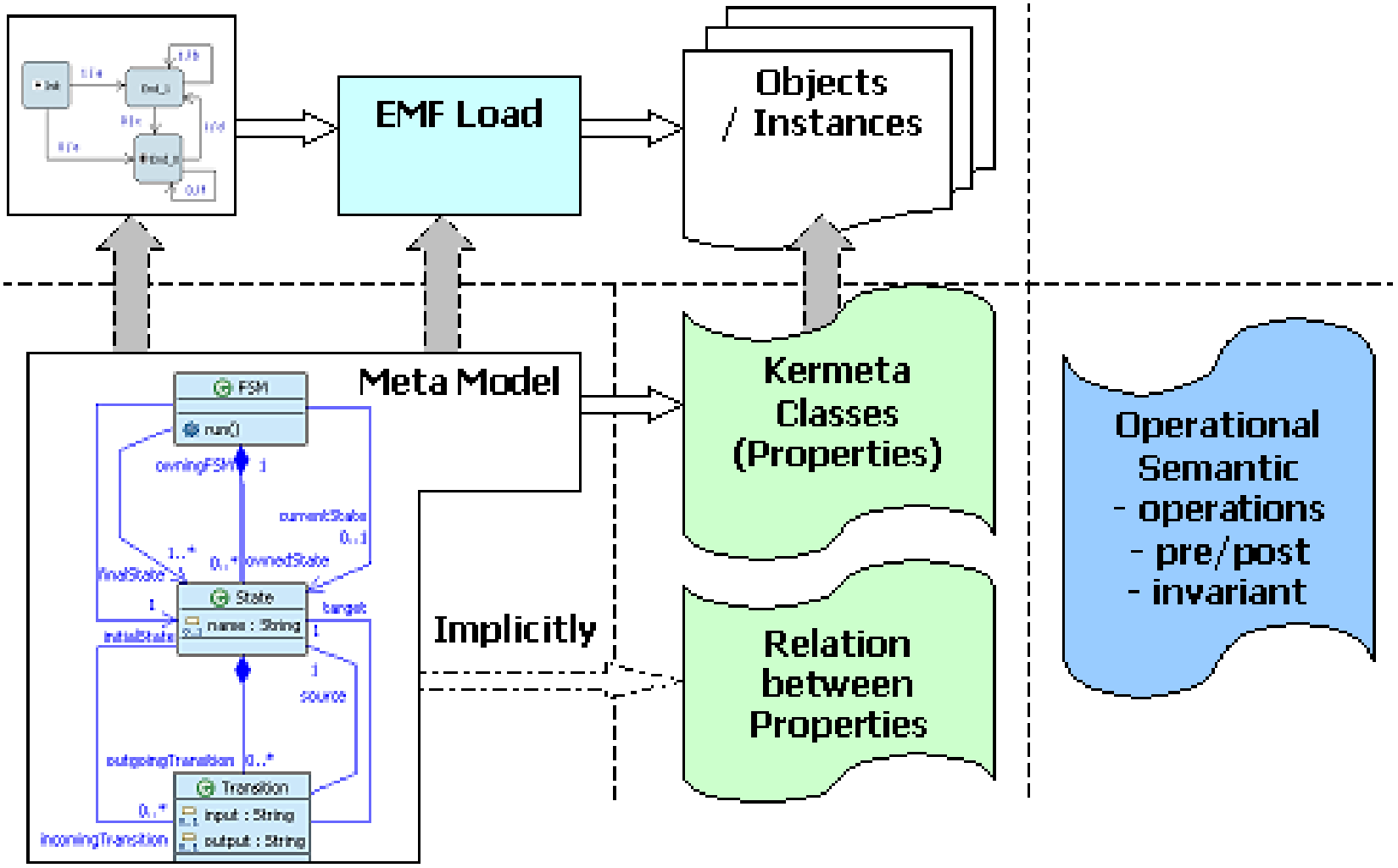
Graph Grammar Compiler

- AToM³ uses Graph Grammar to define operation semantic of meta-model
  - Each rule corresponds to an EMOF operation
- Goal: GG Compiler for Kermeta
  - Using AToM³ Graph Grammar to define the behaviour of Kermeta meta-model.
  - Then compile them into Kermeta program.
- Eugene and Hans Solution
  - Compile GG into Python classes
  - Specify the control structure of the transformation rules in MoTif
  - Adv: Models and GG are represented as ASG (Abstract Syntax Graphs), graph rewriting

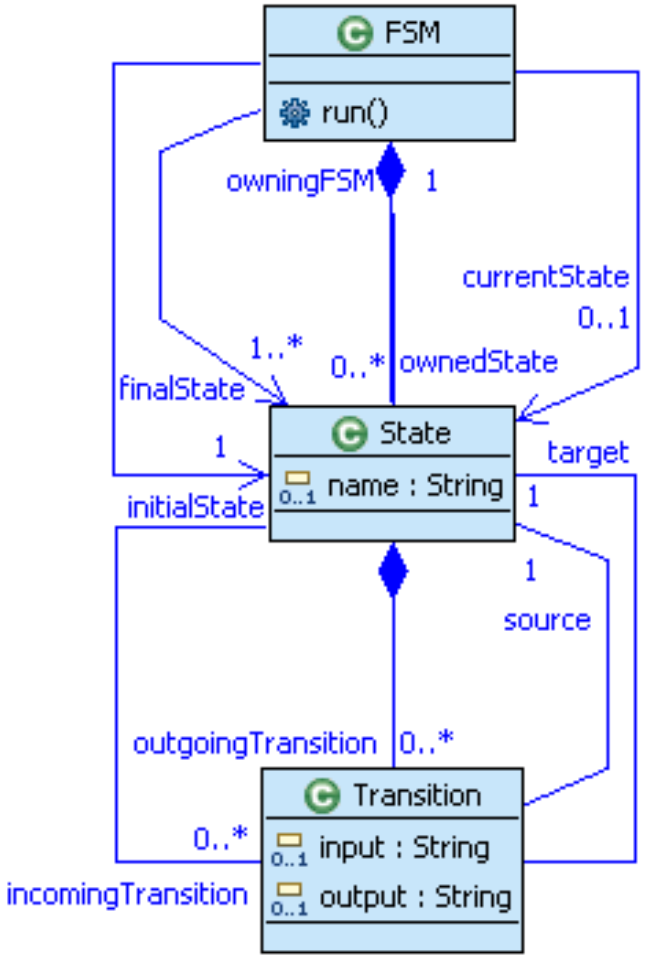
DEVS-based Programmed Graph Rewriting Architecture [2]

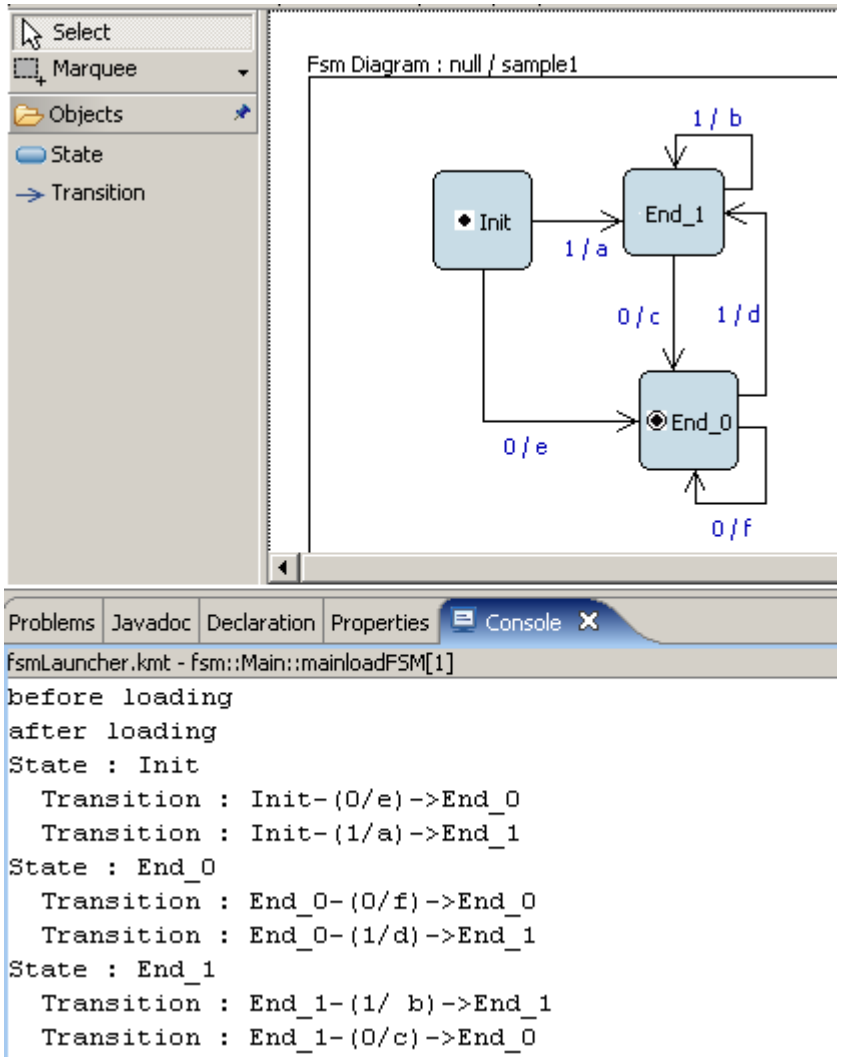


Kermate Meta-modelling

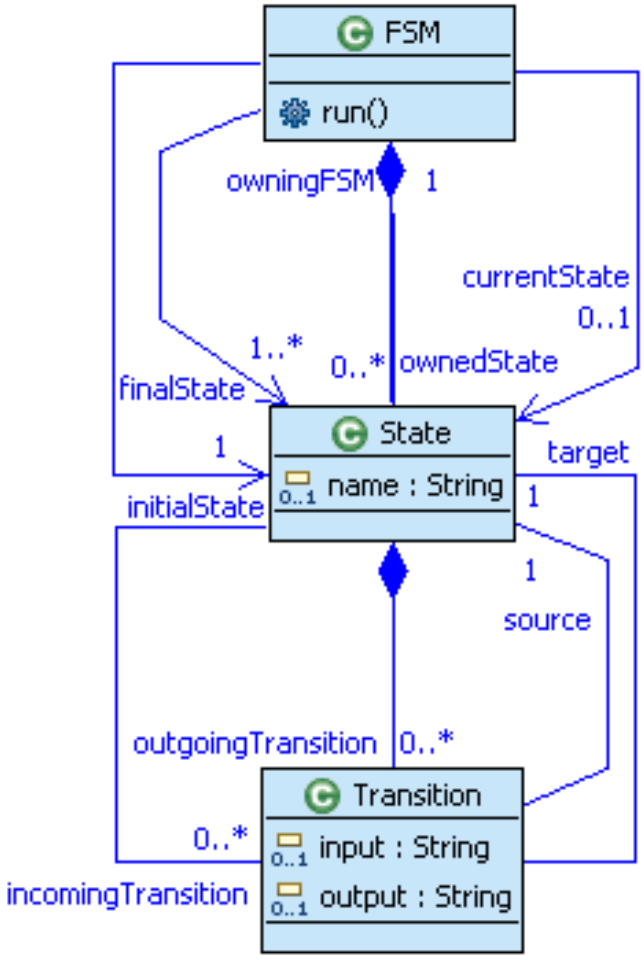


Kermeta FSM-Model



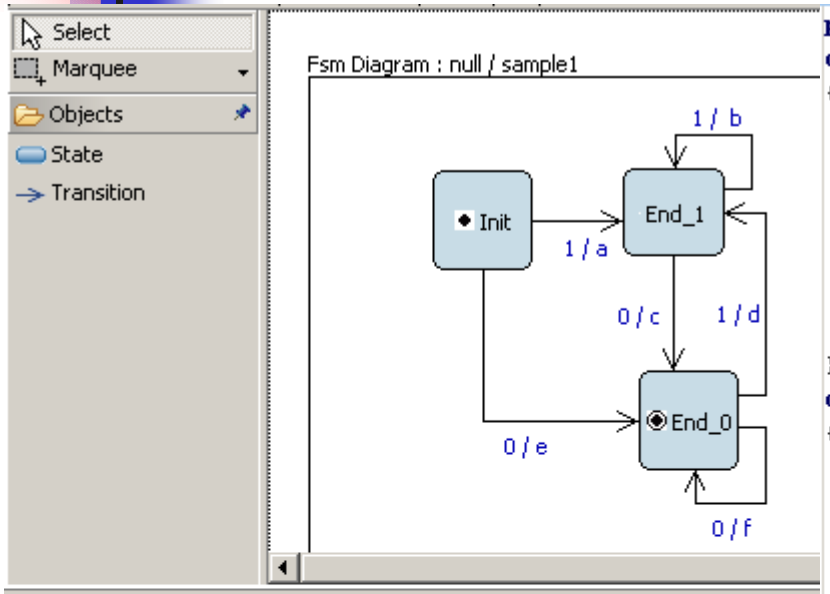


Internal Data Structure




```
package fsm;
class FSM
{
    attribute ownedState : State[0..*]#owningFSM
    reference initialState : State[1..1]
    reference finalState : State[1..*]
    reference currentState : State
    @overloadable "true"
    operation run() : kermeta::standard::~Void is
        raise kermeta::exceptions::NotImplementedException.new
}
class State
{
    reference owningFSM : FSM[1..1]#ownedState
    attribute name : String
    attribute outgoingTransition : Transition[0..*]#source
    reference incomingTransition : Transition[0..*]#target
}
class Transition
{
    reference source : State[1..1]#outgoingTransition
    reference target : State[1..1]#incomingTransition
    attribute input : String
    attribute output : String
}
```

Implicit Relation



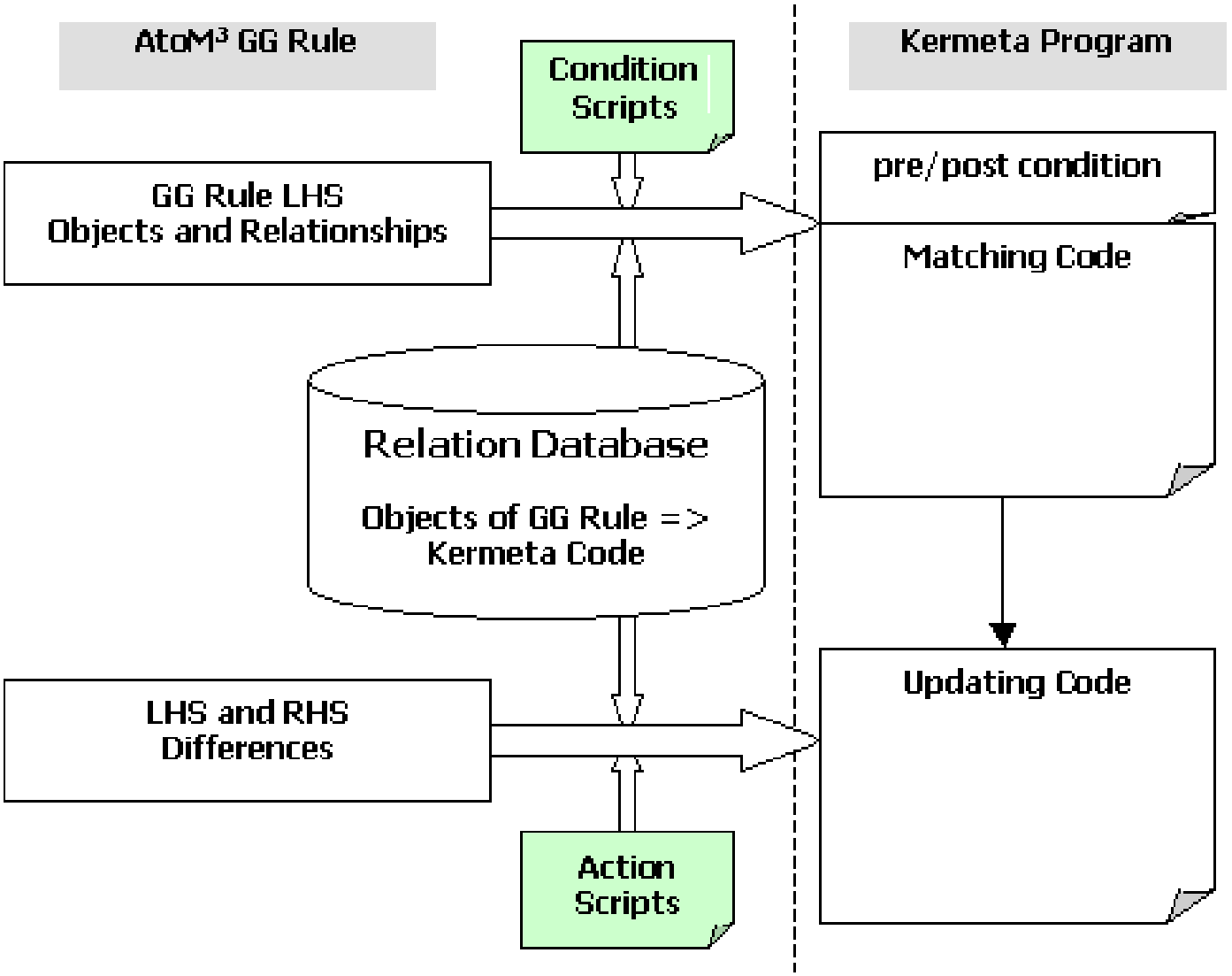
Problems | Javadoc | Declaration | Properties | Console

fsmLauncher.kmt - fsm::Main::mainloadFSM[1]

```
before loading
after loading
State : Init
  Transition : Init-(0/e)->End_0
  Transition : Init-(1/a)->End_1
State : End_0
  Transition : End_0-(0/f)->End_0
  Transition : End_0-(1/d)->End_1
State : End_1
  Transition : End_1-(1/ b)->End_1
  Transition : End_1-(0/c)->End_0
```

```
package fsm;
class FSM
{
    attribute ownedState : State[0..*]#owningFSM
    reference initialState : State[1..1]
    reference finalState : State[1..*]
    reference currentState : State
    @overloadable "true"
    operation run() : kermeta::standard::~Void is
        raise kermeta::exceptions::NotImplementedException.new
}
class State
{
    reference owningFSM : FSM[1..1]#ownedState
    attribute name : String
    attribute outgoingTransition : Transition[0..*]#source
    reference incomingTransition : Transition[0..*]#target
}
class Transition
{
    reference source : State[1..1]#outgoingTransition
    reference target : State[1..1]#incomingTransition
    attribute input : String
    attribute output : String
}
fsm.ownedState.each
{ s |
    stdio.writeln("State : " + s.name)
    s.outgoingTransition.each { t |
       stdio.writeln("  Transition : " + t.source.name
       + "-(" + t.input + "/" +  t.output   + ")->"
       + t.target.name)
}   }
```

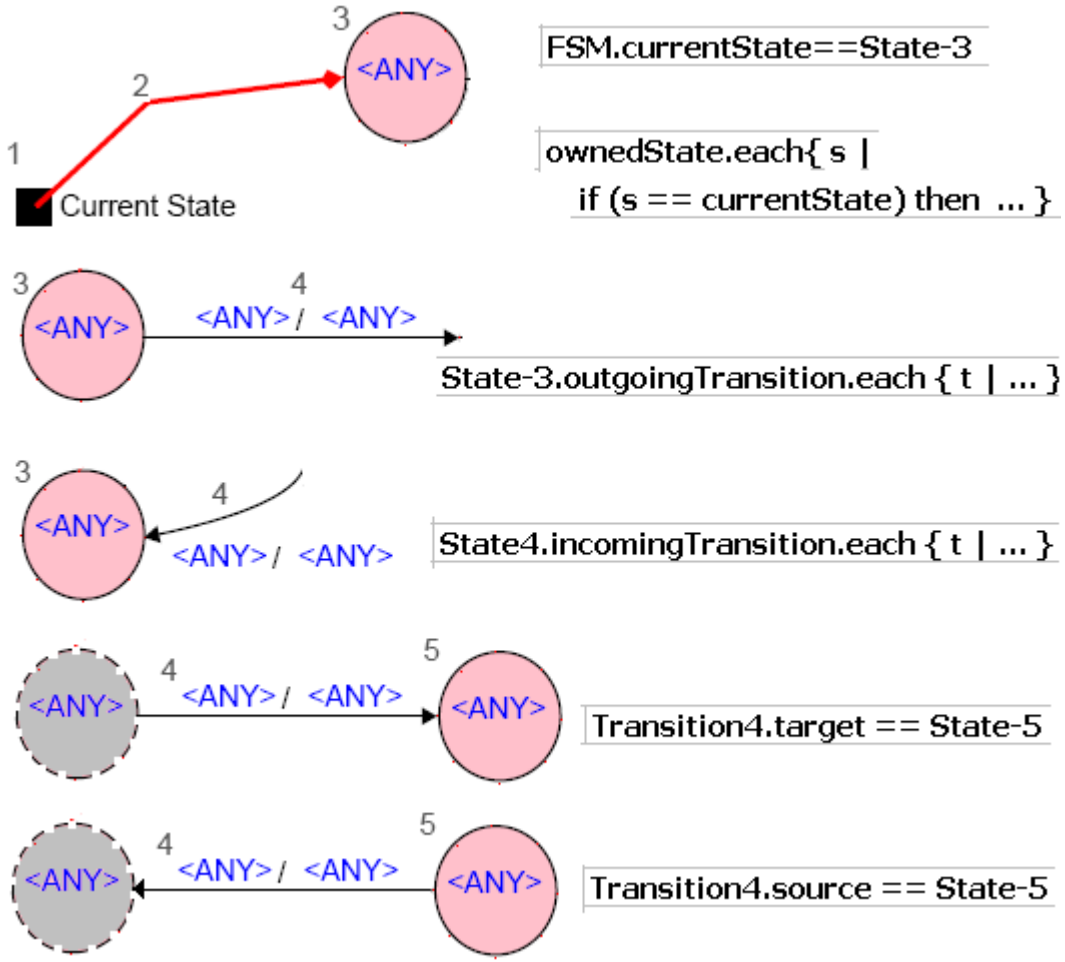
GG Compiler Mechanism

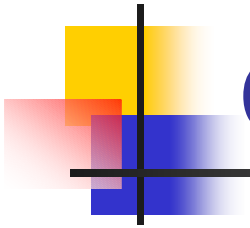


GG – Kermeta Relations

Class in GG	Class in Kermeta FSM
FSAState	State
FSATransition	Transition
current	FSM.currentState
points_to	--

Properties	
FSAState.	***State.***
out_connections_	outgoingTransaction[0..*]
in_connections_	incomingTransaction[0..*]
FSATransition	***Transition***
out_connections_	target
in_connections_	source

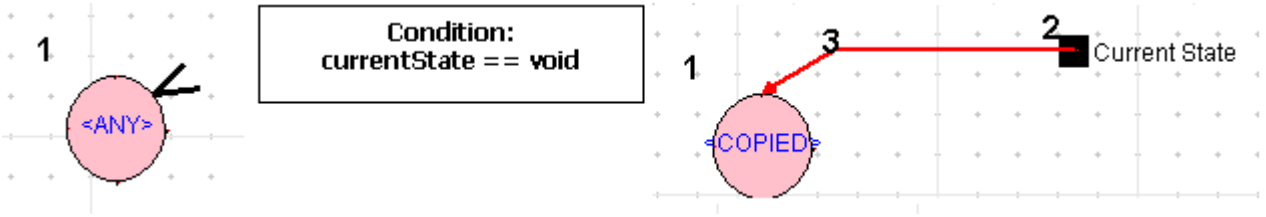




Compilation Principles

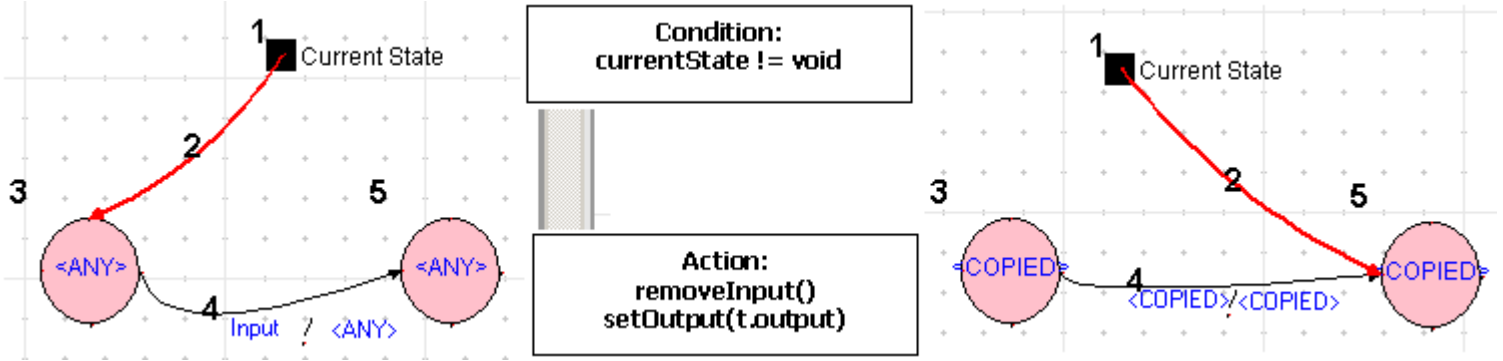
- Each Rule is translated into an operation,
  - All the operations are belong to FSM class
  - LHS is converted to conditional expressions
  - RHS is converted to assignment statements
- Condition of each rule
  - Converted to pre/pos conditions of the operation
  - Only allows local variable, FSM member variables, which can be known from the Class Diagram
  - Input[] are global variables
- Attributes with specified value will become logical expressions for matching code
- Action of each rule will copy into program
  - Assume syntax is correct, may use local variables
  - Assume exist helper methods

Compiling Rule-1



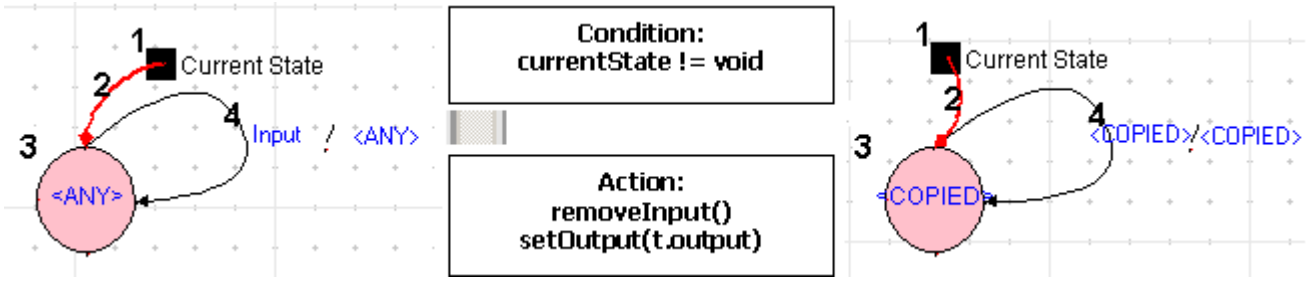

```
operation Rule01(str : String) : String raises FSMException
    pre preRule01 is
        currentState == void
is do
    ownedState.each
    { s |    // Matching LHS
        if(s == initialState) then
            do  // LHS->RHS changes
                currentState := s
                // Generel error handling
                rescue (err : Exception)
                    stdio.writeln(err.toString)
                    stdio.writeln(err.message)
            end
        end
    }
end
```

Compiling Rule-2




```
operation Rule02(str : String) : String raises FSMException
    pre preRule02 is currentState != void
is do
    ownedState.each
    { s | // Matching Code
        if (s == currentState) then
            s.outgoingTransition.each { t |
                if (t.input.equals(Input)) then
                    do  // Updating Code
                        setOutput(t.output) // 1. Produce output
                        removeInput()       // 2. consumed the input
                        currentState := t.target // 3. updating current-state
                        // Generel error handling
                        rescue (err : Exception)
                            stdio.writeln(err.toString)
                            stdio.writeln(err.message)
end     end     }   end    }     end
```

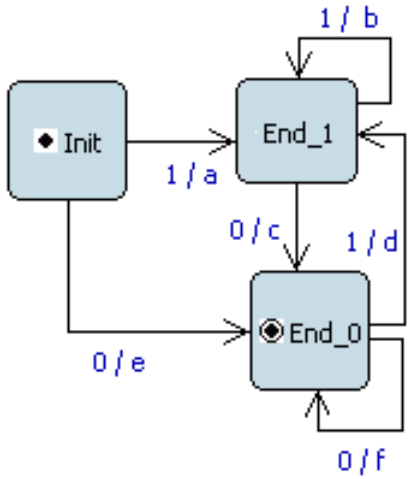
Compiling Rule-3




```
operation Rule03(str : String) : String raises FSMException
    pre preRule03 is currentState != void
is do
    ownedState.each
    { s | // Matching Code
        if (s == currentState) then
            s.outgoingTransition.each { t |
                if (t.input.equals(Input) and t.target==s) then
                    do  // Updating Code
                        setOutput(t.output) // 1. Produce output
                        removeInput()       // 2. consumed the input
                        // Generel error handling
                        rescue (err : Exception)
                            stdio.writeln(err.toString)
                            stdio.writeln(err.message)
end     end     }   end     }     end
```

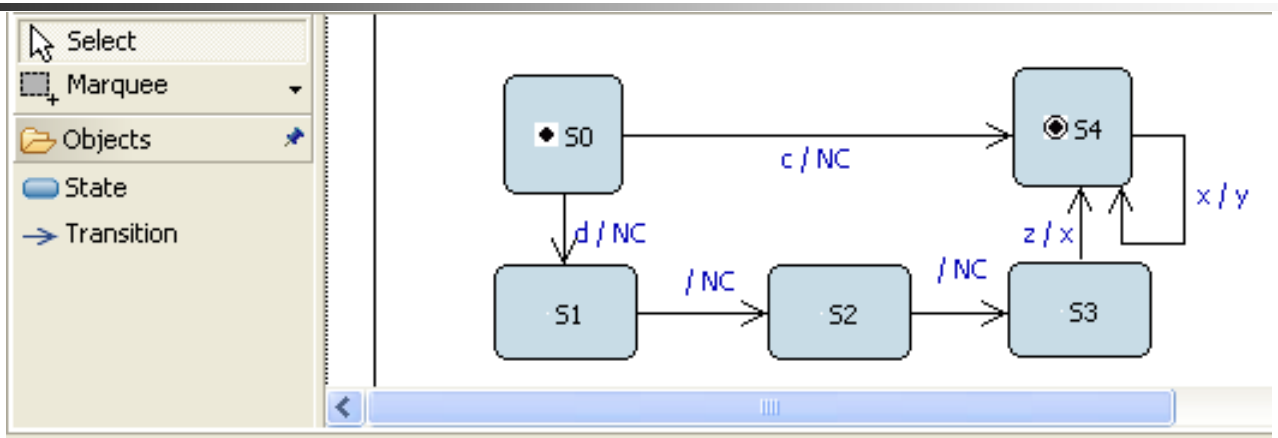
Simulation Example

```
fsmLauncher.kmt - fsm::Main::mainloadFSM (pre/post activated)[3]
Current state : <None>
>> give me a letter : 0
-- Rule01 is applied.
Current state : Init
-- Rule02 is applied.
string produced : e
Current state : End_0
[kermeta::exceptions::ConstraintViolatedPre:15497]
pre preRule01 of operation Rule01 of class FSM violated
Current state : End_0
>> give me a letter : 0
[kermeta::exceptions::ConstraintViolatedPre:15662]
pre preRule01 of operation Rule01 of class FSM violated
-- Rule03 is applied.
string produced : f
Current state : End_0
[kermeta::exceptions::ConstraintViolatedPre:15718]
pre preRule01 of operation Rule01 of class FSM violated
Current state : End_0
>> give me a letter : 1
[kermeta::exceptions::ConstraintViolatedPre:17439]
pre preRule01 of operation Rule01 of class FSM violated
-- Rule02 is applied.
string produced : d
Current state : End_1
[kermeta::exceptions::ConstraintViolatedPre:17500]
pre preRule01 of operation Rule01 of class FSM violated
Current state : End_1
```




```
>> give me a letter : 0
[kermeta::exceptions::ConstraintViolatedPre:17654]
pre preRule01 of operation Rule01 of class FSM violated
-- Rule02 is applied.
string produced : c
Current state : End_0
[kermeta::exceptions::ConstraintViolatedPre:17710]
pre preRule01 of operation Rule01 of class FSM violated
Current state : End_0
>> give me a letter :
```

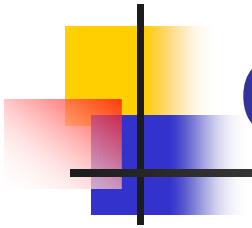
Simulation Example-2



Problems | Javadoc | Declaration | Console | Properties

fsmLauncher.kmt - fsm::Main::mainloadFSM (pre/post activated)[4]

```
Current state : <None>
>> give me a letter : d
-- Rule01 is applied.
Current state : S0
-- Rule02 is applied.
Current state : S1
-- Rule02 is applied.
Current state : S2
-- Rule02 is applied.
Current state : S3
[kermeta::exceptions::ConstraintViolatedPre:18943]
pre preRule01 of operation Rule01 of class FSM violated
Current state : S3
>> give me a letter :
```



Conclusion

- KerMeta is heavily rely on the EMF
  - Declaration references, attributes of class
  - Data internal relationship based on Class Diagram
  - Compiler needs to interpret those relations for each meta-model
  - Limitation on pre/post conditions, actions code
- Advantage:
  - Kermeta code is more readable, easy to debug in the comprehensive IDE, meet S.E. aspect, relatively efficient in non-graph calculation
- AToM³ uses data structure ASG
  - AToM³ GG Compiler can work on all ASG graph
  - Efficient in graph rewriting, simulation

Thank you!