Meta-modeling with OCL & KerMeta

Prof. Jean-Marc Jézéquel
(Univ. Rennes 1 & INRIA)
Triskell Team @ IRISA
Campus de Beaulieu
F-35042 Rennes Cedex
Tel : +33 299 847 192 Fax : +33 299 847 171
e-mail : jezequel@irisa.fr
http://www.irisa.fr/prive/jezequel

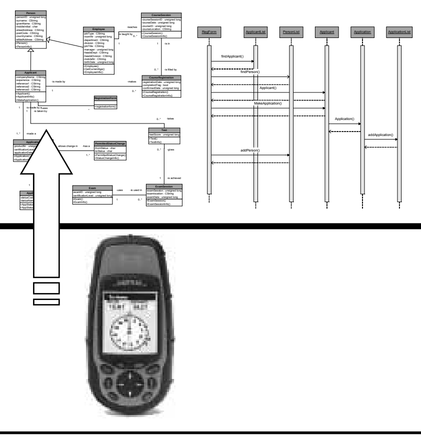
The World and the Model

- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*
  - Consider modeling both the machine & its environment (M. Jackson)
- UML paved the way from OOP to Model Based SE

Specificity of Engineering: Model something not yet existing(in order to build it)

M_1 (modeling space)

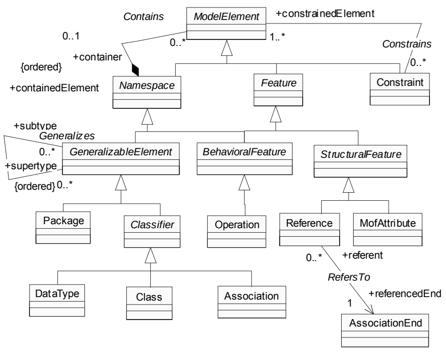
Is represented by



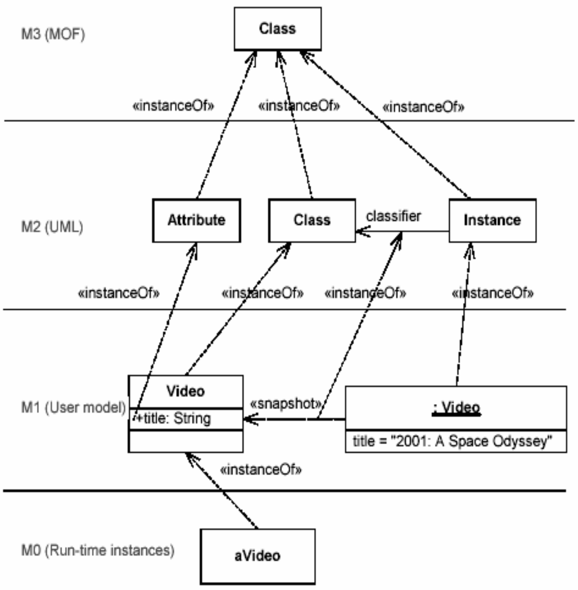
M_0 (the world)

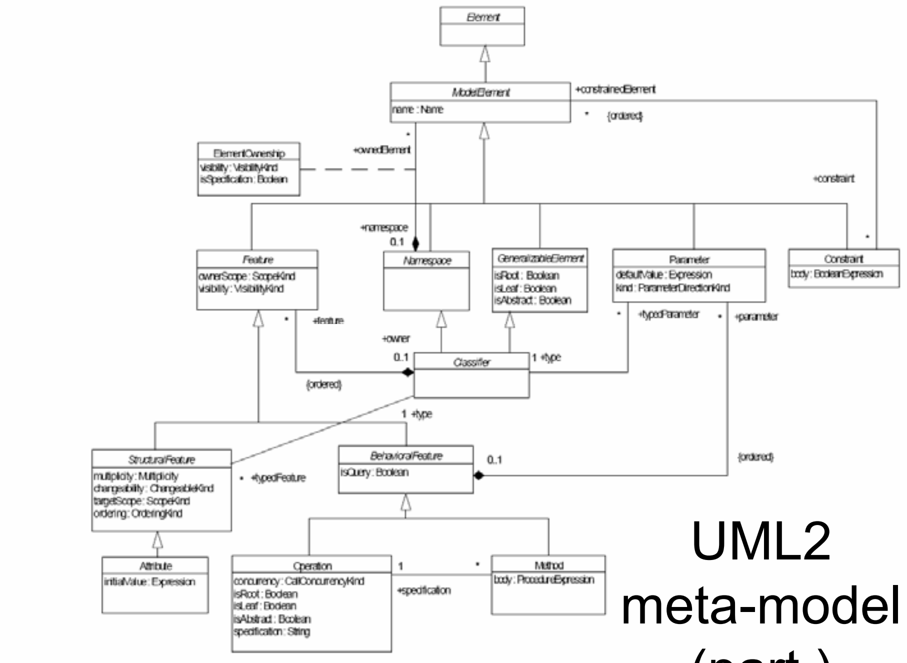
Assigning Meaning to Models

- If a UML model *is no longer* just
  - fancy pictures to decorate your room
  - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
  - Let's make a model of what a model is!
  - => *meta-modeling*
    - » & meta-meta-modeling..
    - » Use Meta-Object Facility (MOF) to avoid infinite Meta-recursion



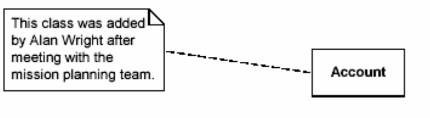
The 4 layers in practice

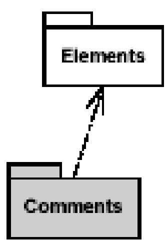


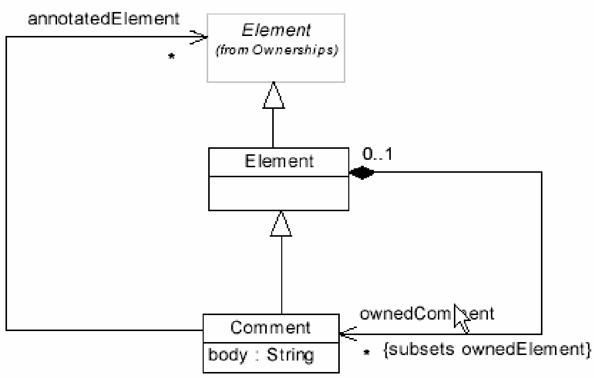


UML2 meta-model (part.)

Zoom: comments







Generalizations

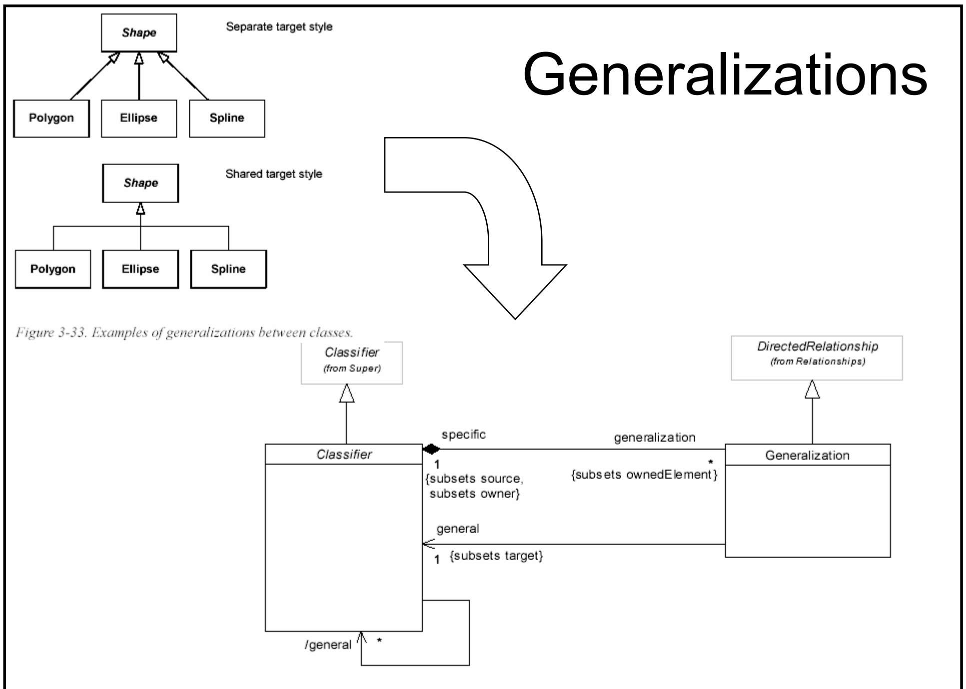


Figure 3-33. Examples of generalizations between classes.

Figure 3-32. The elements defined in the Generalizations package.

Static Semantics with OCL

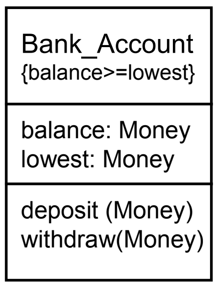
- Complementing a meta-model with Well-Formedness Rules, aka *Contracts* e.g.;
  - ModelElement has a unique name in a Namespace
  - no cycle in a UML2 inheritance graph…
- Expressed with the OCL (Object Constraint Language)
  - The OCL is a language of typed expressions.
  - A constraint is a valid OCL expression of type Boolean.
  - A constraint is a restriction on one or more values of (part of) an object-oriented model or system.

OCL

- Can be used at both
  - M1 level (constraints on Models)
    - » aka *Design-by-Contract* (Meyer)
  - M2 level (constraints on Meta-Models)
    - » aka Static semantics
- Let’s overview it with M1 level exemples

Class invariants

- Contraints can be added to models
  - notation: between { }
- Invariant = Boolean expression
  - True for all instances of a class in stable states...
  - Expressed with the OCL (Object Constraint Language)
    - » e.g. {balance >= lowest}
    - » Can also navigate the associations



Precondition:
Burden on the client

- Specification on what must be true for a client to be allowed to call a method
  - example: amount > 0
- Notation in UML
  - {«precondition» *OCL boolean expression*}
  - Abbreviation: {pre: *OCL boolean expression*}

Postcondition:
Burden on the implementor

- Specification on what must be true at completion of any successful call to a method
  - example: balance = balance @pre + amount
- Notation in UML
  - {«postcondition» *OCL boolean expression*}
  - Abbreviation: {post: *OCL boolean expression*}
  - Operator for previous value (idem old Eiffel):
    - » *OCL expression* @pre

To be Abstract and Precise

Bank_Account
{balance>=lowest}

balance: Money
lowest: Money

deposit (amount: Money)
 {pre: amount> 0}
 {post: balance = balance @pre + amount}
withdraw(amount: Money)
 {pre: amount> 0 and montant<=balance-lowest}
 {post: balance = balance @pre - amount}

- In memory implementation
    - straightforward
    - list of transactions
- Data base implementation
- etc.

Types in OCL

The types in OCL are as follows:

- Predefined types
    - Basic types - Integer, Real, String and Boolean
    - Collection types - Collection, Set, Bag, Sequence
- Meta types
    - OclAny, OclExpression, OclType
- User-defined model types
    - Enumeration and all classes, types and interfaces

Boolean

Operation	Notation	Result type
or	a or b	Boolean
and	a and b	Boolean
exclusive or	a xor b	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implication	a implies b	Boolean
if then else	if a then b1 else b2 endif	type of b

Real and Integer

Operation	**Notation**	**Result type**
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiply	a * b	Integer or Real
divide	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs	Integer or Real
maximum	a.max(b)	Integer or Real
minimum	a.min(b)	Integer or Real
round	a.round	Integer
floor	a.floor	Integer

String

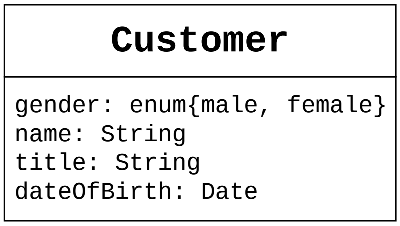
Operation	Expression	Result type
concatenation	s.concat(string)	String
size	s.size	Integer
to lower case	s.toLower	String
to upper case	s.toUpper	String
substring	s.substring(int, int)	String
equals	s1 = s2	Boolean
not equals	s1 <> s2	Boolean

Simple constraints

```
Customer
--------
name: String
title: String
age: Integer
isMale: Boolean
```

```
title = if isMale then ‘Mr.’ else ‘Ms.’ endif
age >= 18 and age < 66
name.size < 100
```

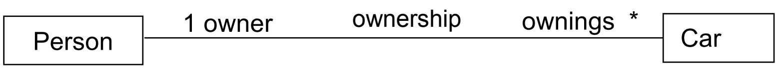
Using Enumerations




```
gender = #male implies title = 'Mr. '
```


Non-local contracts: navigating associations

- Each association is a navigation path
  - The context of an OCL expression is the starting point
  - Role names are used to select which association is to be traversed (or target class name if only one)



Context Car inv:
self.owner.age >= 18

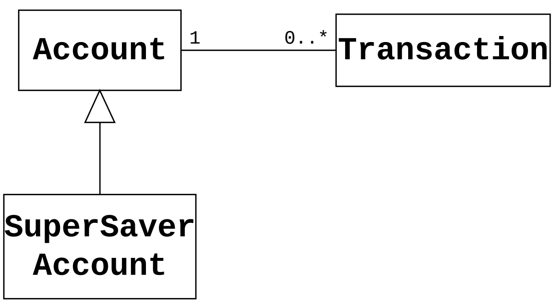
Navigation of 0..* associations

- Through navigation, we no longer get a scalar but a *collection* of objects
- OCL defines 3 sub-types of collection
  - **Set** : when navigation of a 0..* association
    - » *Context Person inv: ownings* return a Set[Car]
    - » Each element is in the Set at most once
  - **Bag** : if more than one navigation step
    - » An element can be present more than once in the Bag
  - **Sequence** : navigation of an association {ordered}
    - » It is an ordered Bag
- Many predefined operations on type *collection*

Syntax::
Collection->operation

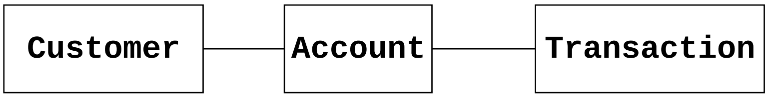
Constraint examples

```
self.transaction -> forAll(t:Transaction | t.value > 100)
```




```
self.balance > 0
```

Navigating to collections



Customer

`self.account` produces a set of Accounts

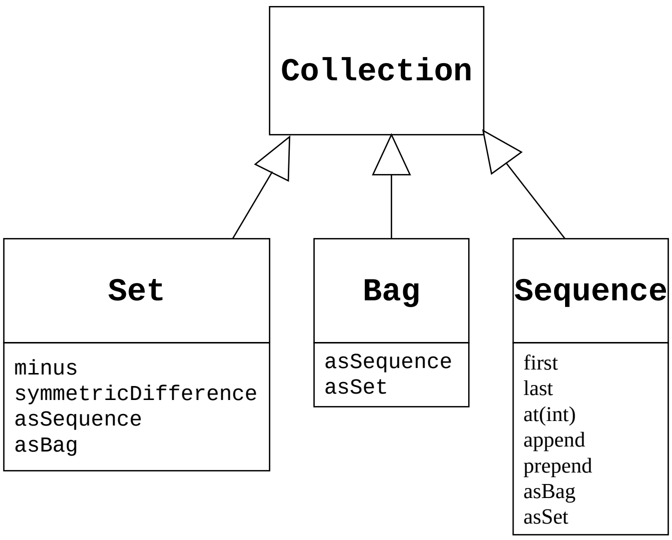
Customer

`self.account.transaction` produces a bag of transactions

If we want to use this as a set we have to do the following

```
self.account.transaction -> asSet
```


Collection hierarchy



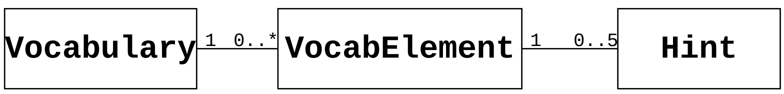
Basic operations on collections

- *isEmpty*
  - *true* if collection has no element

Context Person inv:
age<18 implies ownings->isEmpty

- *notEmpty*
  - *true* if collection has at least one element
- *size*
  - Number of elements in the collection
- *count (elem)*
  - Number of occurrences of element *elem* in the collection

Multiplicity constraints



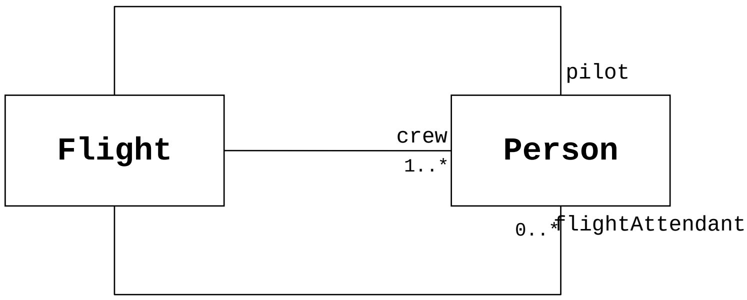
Equivalent constraints expressed on the classes

```
VocabElement
self.hint -> size >= 0 and self.hint -> size <= 5

VocabElement
self.vocabulary -> size = 1

Hint
self.vocabElement -> size = 1
```

The subset constraint




```
Flight
self.crew -> includes( self.pilot )

Flight
self.crew -> includesAll(self.flightAttendants)
```


select Operation

- possible syntax
  - collection->select(elem:T | expr)
  - collection->select(elem | expr)
  - collection->select(expr)
- Selects the subset of *collection* for which property *expr* holds
- e.g.

```
context Person inv:
ownings->select(v: Car | v.mileage<100000)->notEmpty
```

- shortcut:

```
context Person inv:
ownings->select(mileage<100000)->notEmpty
```

forAll Operation

- possible syntax
  - collection->forall(elem:T | expr)
  - collection->forall(elem | expr)
  - collection->forall(expr)
- True iff *expr* holds for each element of the *collection*
- e.g.

```
context Person inv:
ownings->forall(v: Car | v.mileage<100000)
```

- shortcut:

```
context Person inv:
ownings->forall(mileage<100000)
```


Operations on Collections

Operation	Description
size	The number of elements in the collection
count(object)	The number of occurences of object in the collection.
includes(object)	True if the object is an element of the collection.
includesAll(collection)	True if all elements of the parameter collection are present in the current collection.
isEmpty	True if the collection contains no elements.
notEmpty	True if the collection contains one or more elements.
iterate(expression)	Expression is evaluated for every element in the collection.
sum(collection)	The addition of all elements in the collection.
exists(expression)	True if expression is true for at least one element in the collection.
forAll(expression)	True if expression is true for all elements.

OCL for M2: Examples of WFR

- ModelElement has a unique name in a Namespace

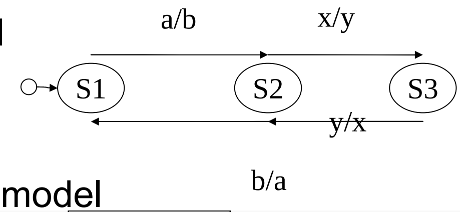
  Context ModelElement inv :

  namespace.ownedElement->collect(name)->count(self.name)=1
- …

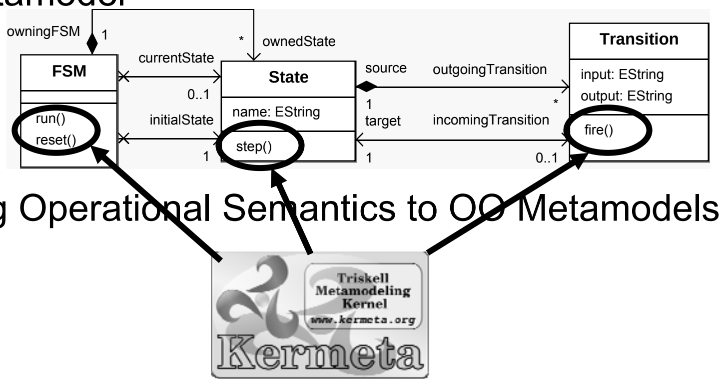
Dynamic Semantic with Kermeta

Example

- A model



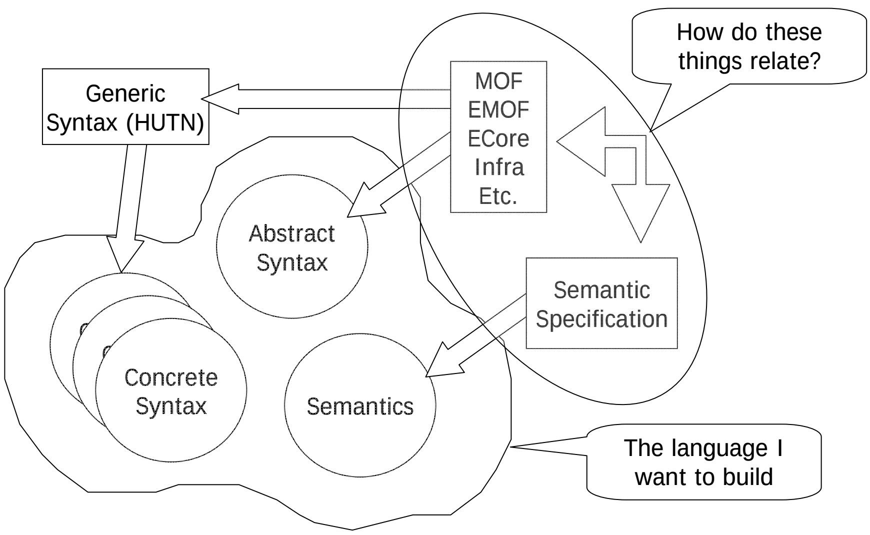
- Its metamodel



- Adding Operational Semantics to OO Metamodels

 34

From Metamodels to Languages



 35

Metadata languages

- (E)MOF => Only data structures
  - classes, properties, associations, ...
  - operations : only signatures
- Not sufficient to operate on models
  - Constraints
  - Actions
  - Transformations
  - ...

 36

Typical example (excerpted from MOF spec)

- *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*”.

A natural language specification

```
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
```

An operational specification

 37

What is "meta"-executability?

- Basic CRUD Operations
- Merge, Composition…

M	Definition
M-1	Execution

►Simply an (object-oriented) program that manipulates model elements

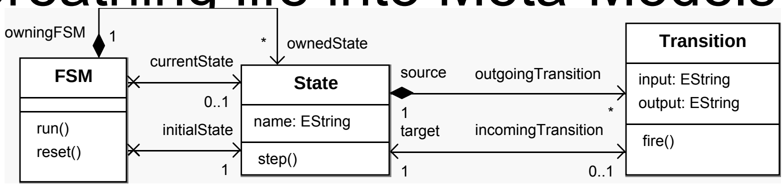
"***Program = Data Structure + Algorithm***", **Niklaus Wirth**

Meta-Executability = Meta-Data + Meta-Actions

Kermeta Rationale

- Model, meta-model, meta-metamodel, DSLs…
  - Meta-bla-bla too complex for the normal engineer
- On the other hand, engineers are familiars with
  - OO programming languages (Java,C#,C++,..)
  - UML (at least class diagram)
  - May have heard of *Design-by-Contract*
- Kermeta leverages this familiarity to make Meta-modeling easy for the masses

Breathing life into Meta-Models



// MyKermetaProgram.kmt
// An E-MOF metamodel is an OO program that does nothing
require "StateMachine.ecore" *// to import it in Kermeta*
// Kermeta lets you weave in **aspects**
// Contracts (OCL WFR)
require “StaticSemantics.ocl”
// Method bodies (Dynamic semantics)
require “DynamicSemantics.kmt”
// Transformations

```
Context FSM
inv: ownedState->forAll(s1,s2|
s1.name=s2.name implies s1=s2)
```

```
aspect class FSM {
  operation reset() : Void {
        currentState := initialState
```

```
class Minimizer {
   operation minimize (source: FSM):FSM {…}
}
```

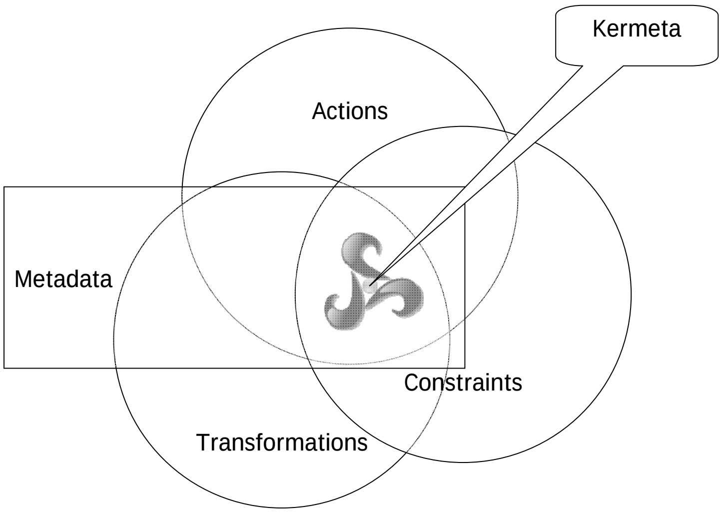
 40

Kermeta:
a **Ker**nel **meta**modeling language

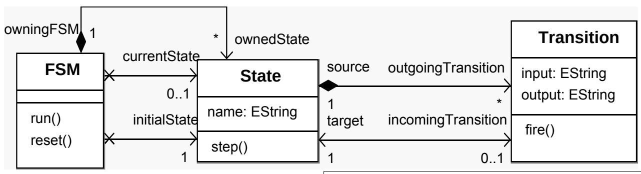
- Strict EMOF extension
- Statically Typed
  - Generics, Function types (for OCL-like iterators)
- Object-Oriented
  - Multiple inheritance / dynamic binding / reflection
- Model-Oriented
  - Associations / Compositions
  - Model are first class citizens, notion of model type
- Aspect-Oriented
  - Simple syntax for static introduction
  - Arbitrary complex aspect weaving as a framework
- Still “kernel” language
  - Seamless import of Java classes in Kermeta for GUI/IO etc.

 41

Kermeta, a Kernel to Meta



EMOF ⇔Kermeta



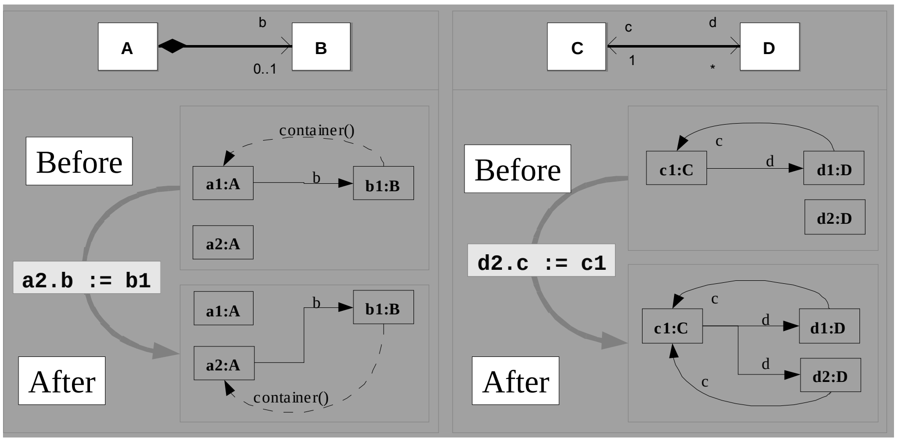
class FSM
{
attribute ownedState : State[0..*]#owningFSM
reference initialState : State[1..1]
reference currentState : State
operation run() : kermeta::standard::~Void **is do**
end
operation reset() : kermeta::standard::~Void **is d**
end
}

class State{
  reference owningFSM : FSM[1..1]#ownedState
  attribute name : String
  attribute outgoingTransition : Transition[0..*]#source
  reference incomingTransition : Transition#target
  operation step(c : String) : kermeta::standard::~Void **is d**
  end
}
class Transition{
  reference source : State[1..1]#outgoingTransition
  reference target : State[1..1]#incomingTransition
  attribute input : String
  attribute output : String
  operation fire() : String **is do**
  end
}

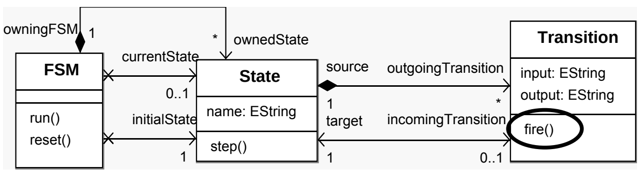
Assignment semantics

Composition

Association

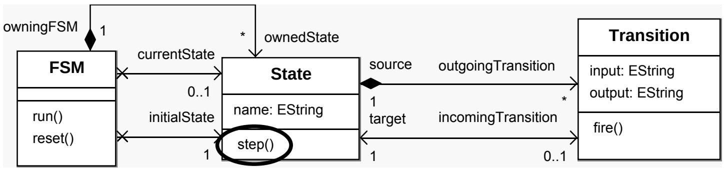


Example



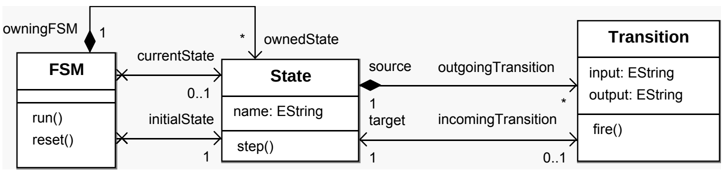
operation fire() : String

source.owningFSM.currentState := target
result := output



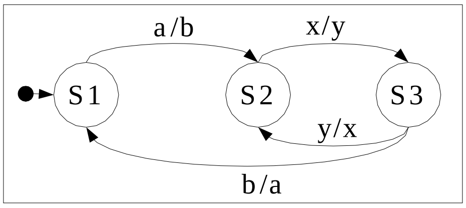
operation step(c : String) : String

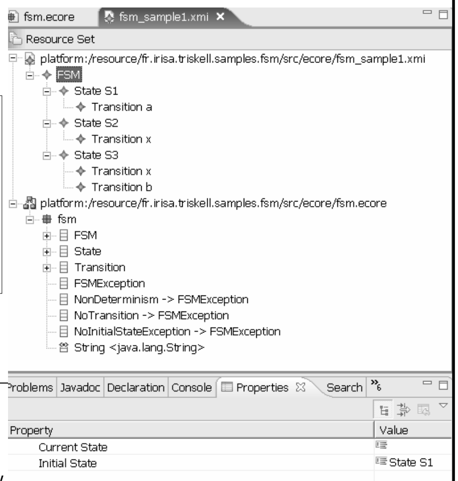
```
// Get the valid transitions
var validTransitions : Collection<Transition>
validTransitions := outgoingTransition.select { t |
        t.input.equals(c)
}
// Check if there is one and only one valid transition
if validTransitions.empty then raise NoTransition.new end
if validTransitions.size > 1 then
        raise NonDeterminism.new
end
// fire the transition
result := validTransitions.one.fire
```

operation run() : **Void**

```
from var str : String
until str == "exit"
loop
        stdio.writeln("current state is " + currentState.name)
        str := stdio.read("Enter an input string or 'exit'
                                        to exit simulation : ")
        stdio.writeln(str)
        if str != "exit" then
                do
                        stdio.writeln("Output string : " + currentState.step(str))
                rescue (ex : FSMException)
                        stdio.writeln("ERROR : " + ex.toString)
                end
        end
end
stdio.writeln("* END OF SIMULATION *")
```

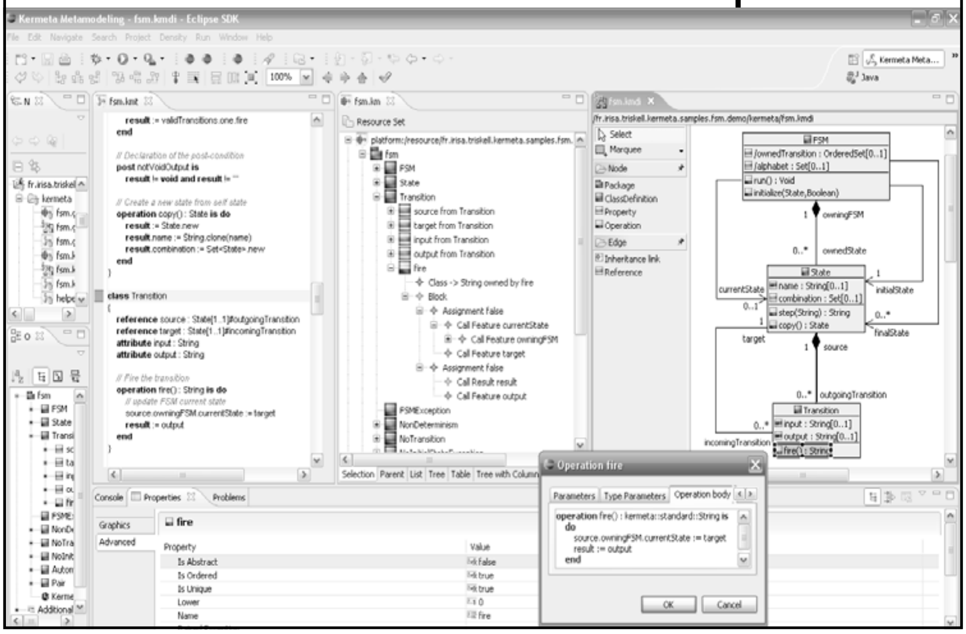




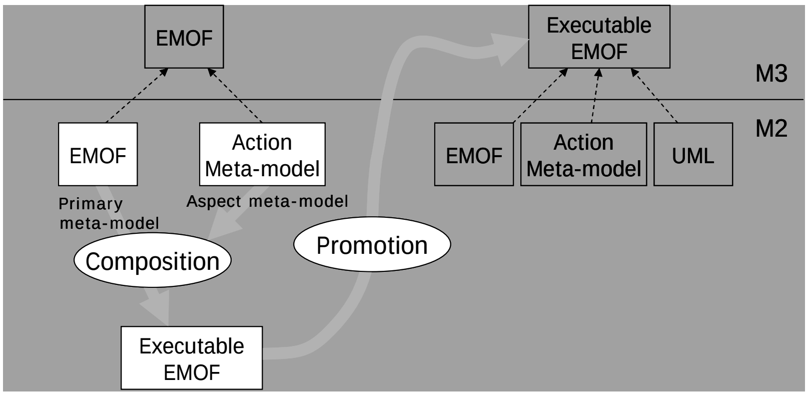

```
/**
 * Load a sample FSM from a xmi2 file
 */
operation loadFSM() : FSM is do
  var repository : EMFRepository init EMFRepository.new
  var resource : EMFResource
  resource ?= repository.createResource("../models/fsm_sample1.xmi", "../metamodels/fsm.ecore")
  resource.load

  // Load the fsm (we get the main instance)
  result ?= resource.instances.one
end
```


Kermeta workbench snapshot



Using aspect-composition to reflectively build Kermeta



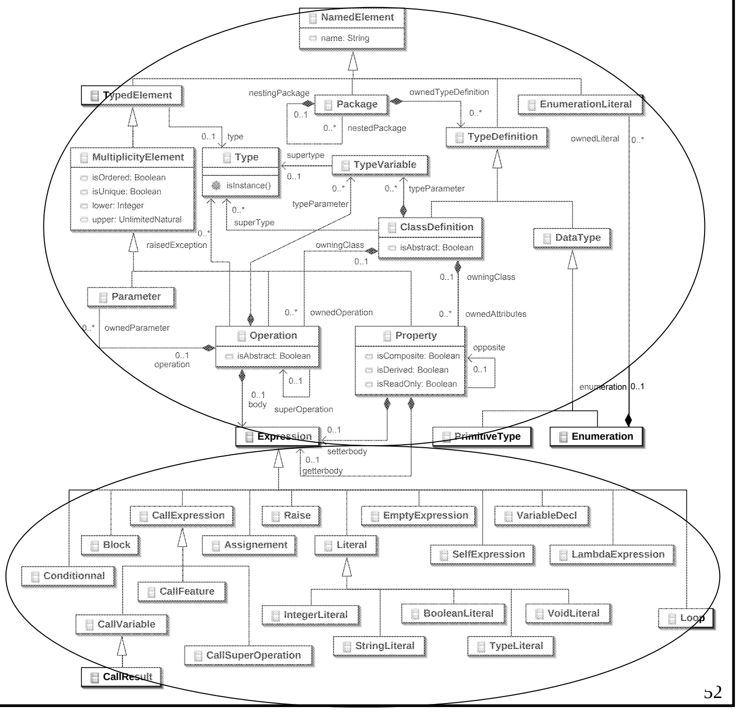
The action metamodel

Kermeta design

Close to the OCL

- CRUD operation
- Control structures
- Operation call
- Variables and assignment
- Exceptions handling
- Functions (OCL-like iterators)

KerMeta Metamodel



 52

Current Status

- Latest version (1.1.0)
  - Parser, type checker, interpreter, debugger
  - Eclipse plug-in: Textual Editor, Browser, Launcher
  - EMF Ecore metamodel Import / Export
  - EMF model Import / Export
  - Constraints (Kermeta or OCL)
  - Graphical Editor (generated with Topcased)
  - Documentation and Examples
- Under development / test
  - Seamless import of Java classes in Kermeta
  - Compiler

 53

Smoothly interoperates with Eclipse/EMF
Open Source
►Download it *now!*

A statically typed object-oriented executable meta-language

- Home page
  - http://www.kermeta.org
- Development page
  - http://kermeta.gforge.inria.fr/