Kermeta tutorial

Reference manual

François Tanguy, Didier Vojtisek, Cyril Faucher

Abstract

This tutorial is part of a serie of tutorials that explain step by step one of the aspect of Kermeta. This one will show you how to use Eclipse to run a Kermeta program.

Published Build date: 3-November-2010
2006-09-25T10:56:58
19/07/2006

List of Figures

CHAPTER

Preface

Kermeta is a Domain Specific Language dedicated to metamodel engineering. It fills the gap let by MOF which defines only the structure of meta-models, by adding a way to specify static semantic (similar to OCL) and dynamic semantic (using operational semantic in the operation of the metamodel). Kermeta uses the object-oriented paradigm like Java or Eiffel. This document presents various aspects of the language, including the textual syntax, the metamodel (which can be viewed as the abstract syntax) and some more advanced features typically included in its framework.

Important

Kermeta is an evolving software and d espite that we put a lot of attention to this document, it may contain errors (more likely in the code samples). If you find any error or have some information that improves this document, please send it to us using the bug tracker in the forge: http://gforge.inria.fr/tracker/?group_id=32 or using the developer mailing list (kermeta-developers@lists.gforge.inria.fr) Last check: v0.3.1

Tip

The most update version of this document is available on line from http://www.kermeta.org .

CHAPTER 1

Prerequisites

The reader is supposed to know how to create KerMeta meta models and how to add behavior to a metamodel. If not, please read the corresponding tutorials. models.

CHAPTER 2

A view on the FSM meta model

The meta model used for this tutorial is contained in the file named "fsm.kmt" in the "metamodels" directory. This is a KerMeta meta model which operations have been filled in.

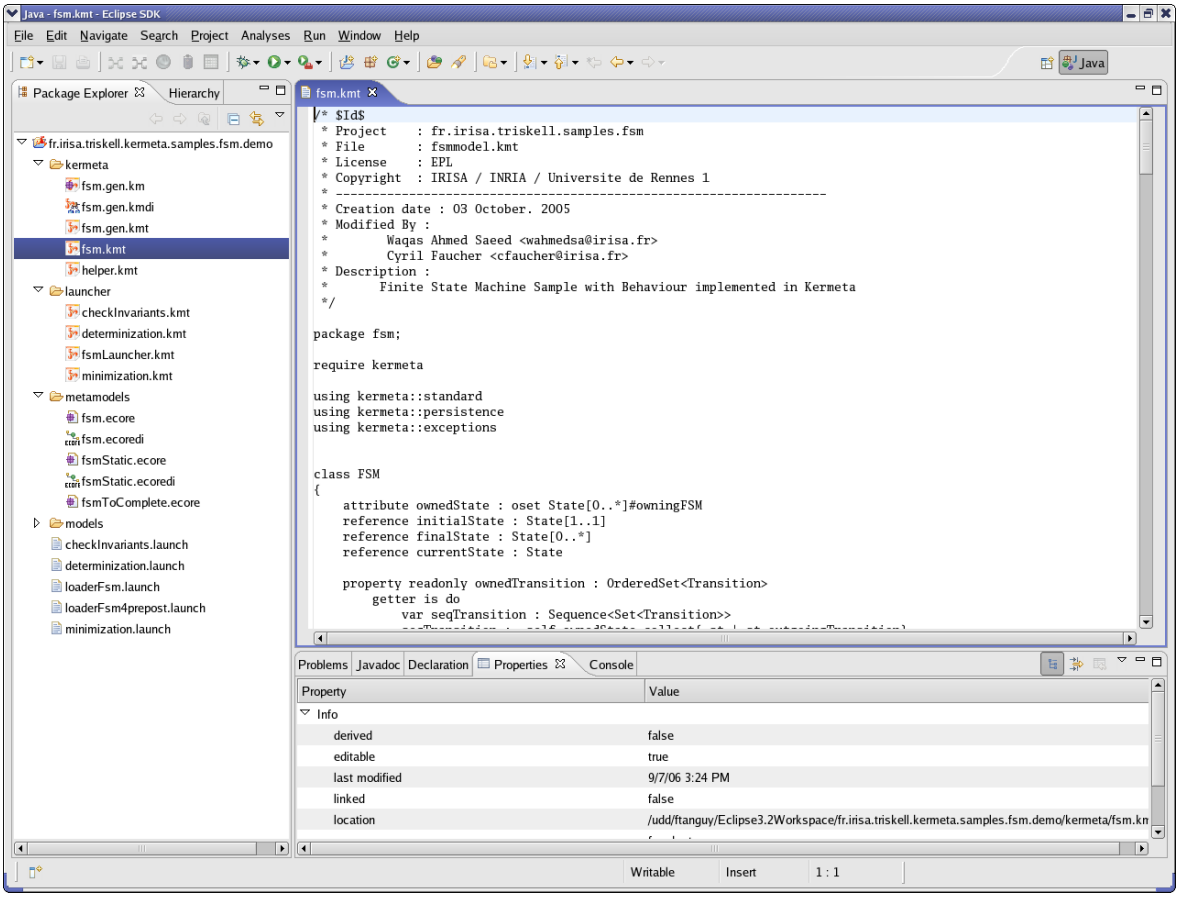

Figure 2.1.

CHAPTER 3

How to run an FSM model ?

3.1. An entry point for the program

We want to execute an FSM model. To do that we must call the "run" operation of the "FSM" class. We are going to do that thanks to a KerMeta script. This script will : load an instance of the FSM meta model stored in a file call the run operation of these instance. To launch a script, the interpreter must know the entry point of the program. That is the roles of these two statements :

- @mainClass which stands for the main class,

- @mainOperation which stands for the main operation of the main class.

In the FSM example, those scripts are in the "launcher" directory. Look at "minimization.kmt" script. Here the interpreter knows that entry point of the program is the operation "main" in the "Minimization" class.

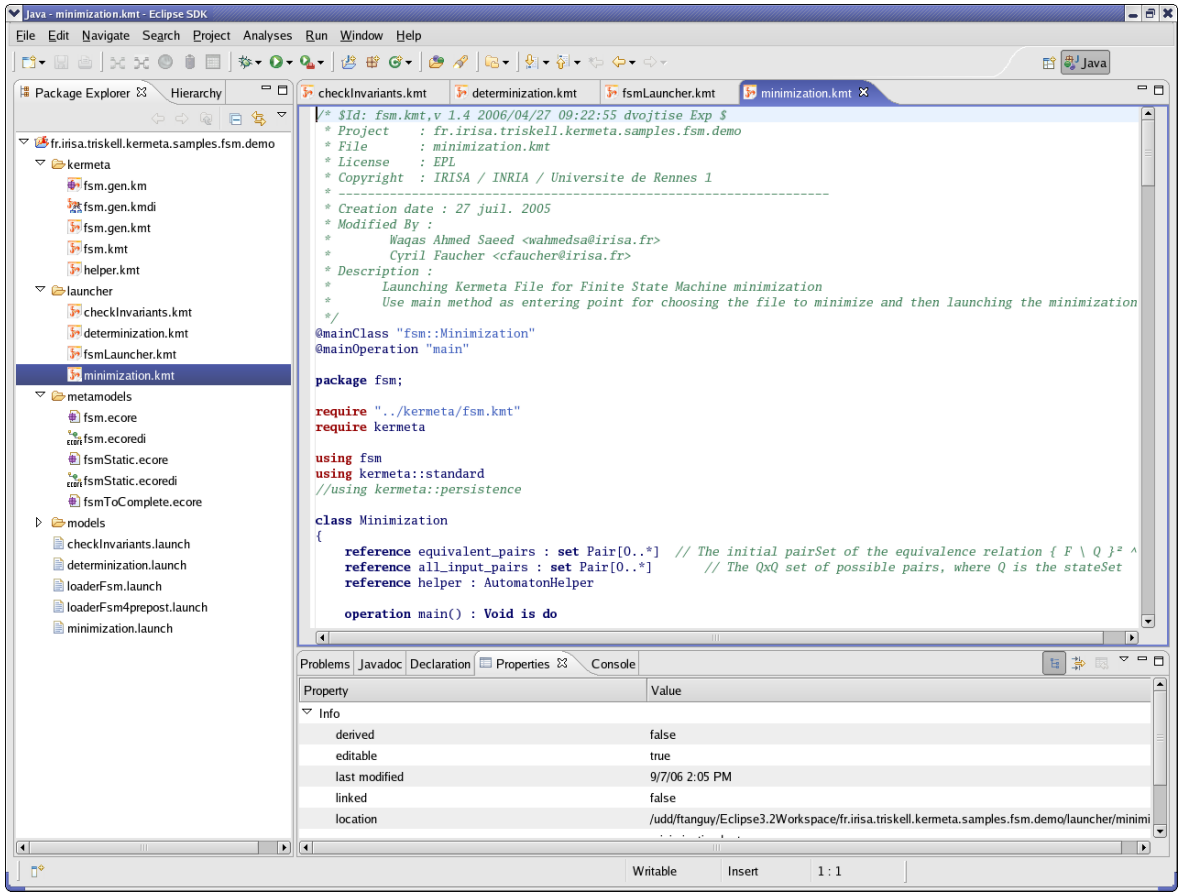

Figure 3.1.

3.2. Constraints checking execution or no constraints checking execution

Using KerMeta allows you to choose between two types of executions : one with constraints checking and one without. A constraints checking execution checks every pre/post condition statements of your KerMeta program. If one of this pre/post conditions are violated, an exception is raised and the program is aborted. A none constraints checking execution does not check the pre/post condition statements of your KerMeta program. For more details, go to section 4. One constraints checking execution sample is explained.

3.3. Execution without parameters

Let's have a look at the file named "minimization.kmt". Open it. Look at the code of the main operation. There is no parameter. To run this script with constraint checking, right click on "minimization.kmt" and select "Run As" and "Kermete App with constraints". To run this script without constraint checking, right click on "minimization.kmt" and select "Run As" and "Kermeta App".

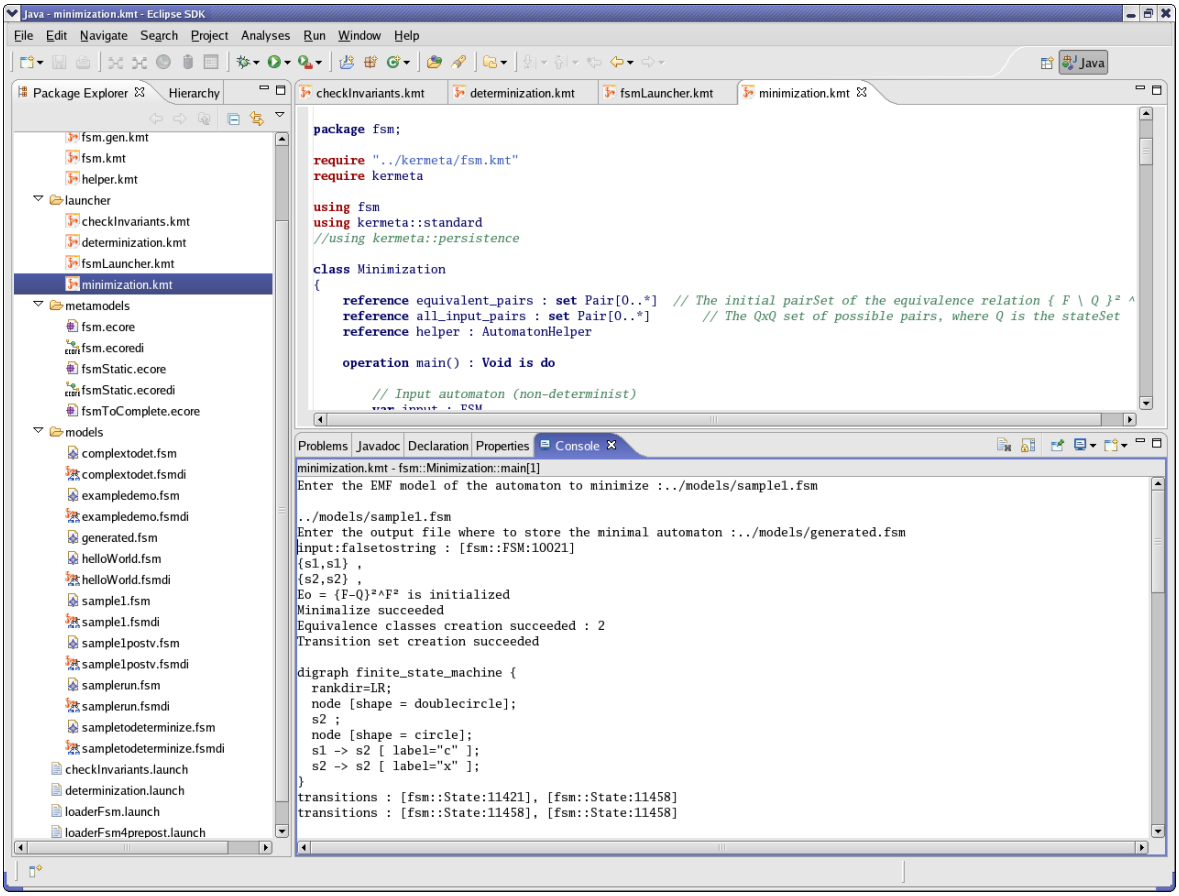

Figure 3.2.

The program asks you for a filename. Put in "../models/sample1.fsm" for example. You are lastly asked for a filename which will correspond to the file generated by the program. Put in "../generated.fsm" and see the execution.

3.4. Execution with parameter(s)

Now if you have a look at the three others scripts (checkInvariants, determinization and fsmLauncher) you will notice that the main operation of the main class takes one argument. Let's focus on "fsmLauncher.kmt" launcher. The main operation takes one parameter which is the name of the file containing the FSM model. It loads the model, prints it and runs it. If you try the running method above, an exception is raised because the parameterized file does not exist. Indeed we did not specify any filename to the program. So, you cannot use the method above to run those kind of script. That is the reason why we are going to use run configurations. Then right click on "fsmLauncher.kmt" file and select "Run As" and "Run...". A window appears like the one below. Select the run configuration named "loaderFSM" and look at the different options. Have a special look at the file parameters :

- "Location of your program file", here this is "fsmLauncher.kmt" filename relative to the project's root directory.

- "Class qualified name", that is to say the main class of the program.
- "Operation name", that is to say the main operation of the main class.
- "Operation arguments", the parameters you want to send to the main operation.

Here, we give the string "../models/sample1.fsm" as a parameter to mainLoadFSM operation to "fsm::Main" class. By clicking on "Run" button, it will start the execution. You can create yourself some new run configurations. Just by left clicking on "Kermeta Application " or "Kermeta Constraint Application" (depending on the constraint checking you want) and select "New" and fill in the required fields.

Caution

Eclipse is slash sensible. It only accepts front slash and no backslash. Then / fr.irisa.triskell.kermeta.samples.fsm.demo/launcher/fsmLauncher.kmt is a valid filename whereas \fr.irisa.triskell.kermeta.samples.fsm.demo\launcher\fsmLauncher.kmt is not.

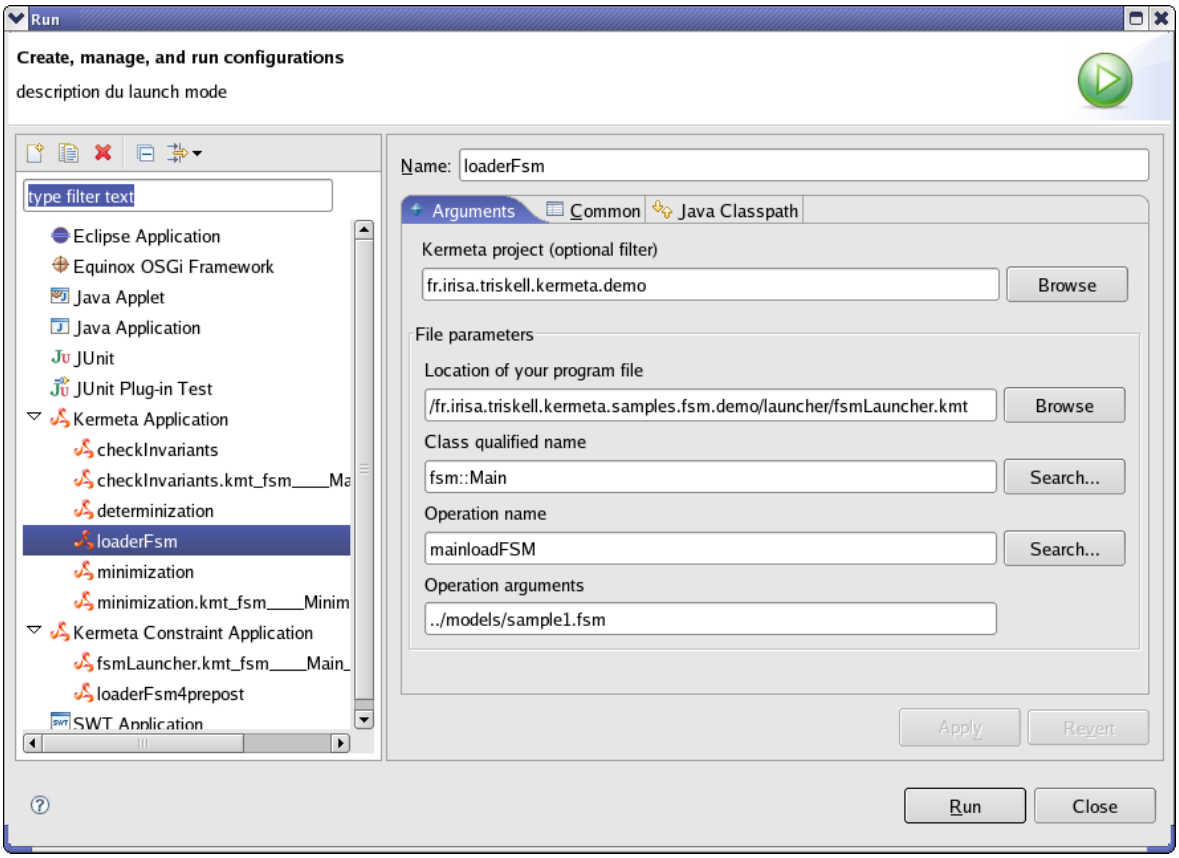

Figure 3.3.

CHAPTER 4

Constraints checking execution sample

Have a look at the "step" method in the "State" class in the "fsm.kmt" file.

```
// Go to the next state
operation step(c : String) : String raises FSMException is

// Declaration of the pre-condition
pre notVoidInput is
  c != void and c != ""

do
  // 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
end

// Declaration of the post-condition
post notVoidOutput is
  result != void and result != ""
```

There is a pre condition which says that the character given as a parameter must not be void or empty string. The post condition says that the result must not be void or empty string. For each "step" method call, the pre and post conditions will be checked. If there are evaluated as false, the program is aborted otherwise the program goes on. Look at the run configuration named "loaderFSM4prepost". Open the file (../models/sample1postv.fsm) used as parameter for this configuration. Observe the finite state diagram.

4.1. Pre condition violation

Execute "loaderFSM4prepost" configuration. When you are asked for a letter , just press enter to send an empty string. Normally, it should provoke the violation of the pre condition. before loading after loading State : s1 Transition : s1-(c/NC)->s2 State : s2 Transition : s2-(x/y)->s2 Current state : s1 give me a letter : stepping... [kermeta::exceptions::ConstraintViolatedPre:8670] pre notVoidInput of operation step of class State violated

4.2. Post condition violation

Execute "loaderFSM4prepost" configuration. When you are asked for a letter , press c and then press enter. Normally, the post condition will be violated because the result will be an empty string. before loading after loading State : s1 Transition : s1-(c/NC)->s2 State : s2 Transition : s2-(x/y)->s2 Current state : s1 give me a letter : c c stepping... [kermeta::exceptions::ConstraintViolatedPost:9684] post notVoidOutput of operation step of class State violated