How the workbench is able to help you in your metamodelling tasks

Didier Vojtisek, Francois Tanguy, Cyril Faucher

Published Build date: 3-November-2010
$LastChangedDate:: 2010-06-02 10:42:20#$

List of Figures

CHAPTER 1

Introduction

Kermeta is distributed with a workbench that provides many features.

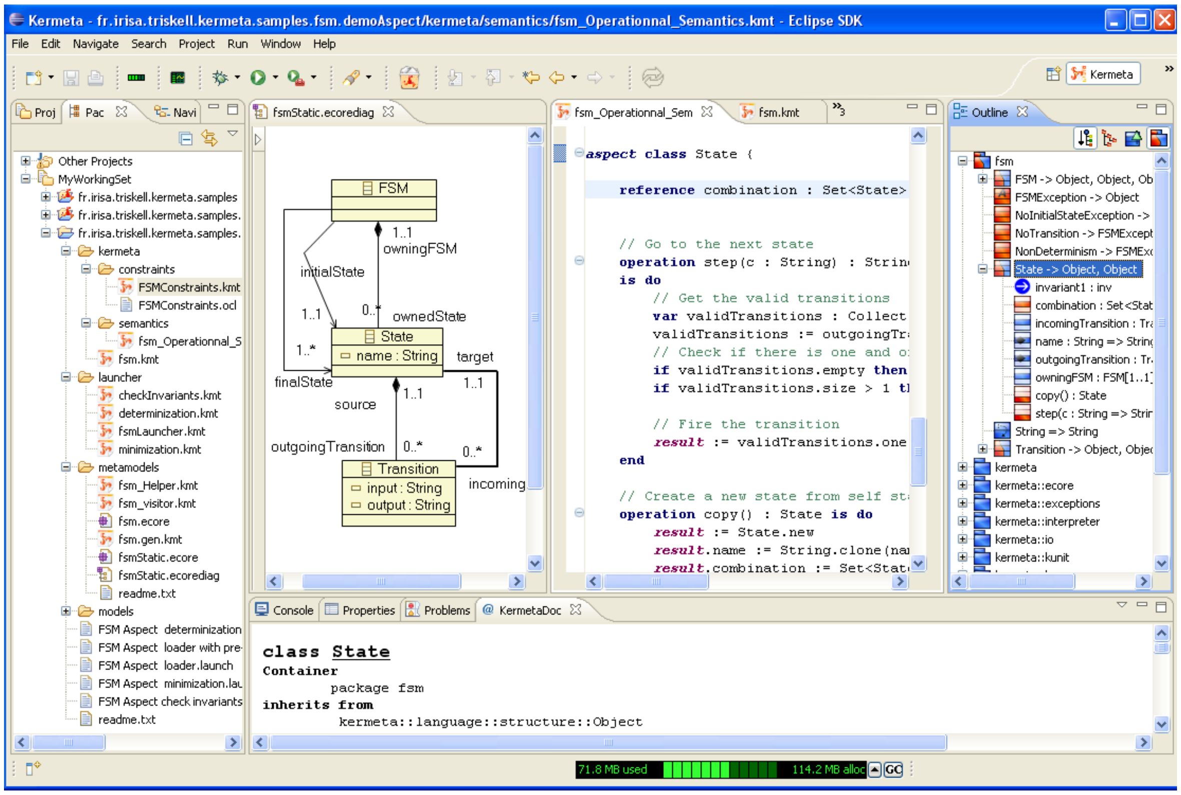
In the bundle version 1.3.0 you have:

- Text editors for kermeta textual syntax and KET (kermeta template emitter)
- an outline view
- a problems view
- an Ecore Model Editor (provided by Eclipse/EMF)
- an interpreter
- a debugger
- a compiler (compiles your kermeta code into an eclipse java plugin)
- Transformations to and from Kermeta. (especially to Ecore)
- Other helper transformations like Ecore merge
- Dedicated Graphic editor
- Object TouchNavigator
- KermetaDoc view (showing the documentation associated to your metamodel)
- Help system

Useful features needing the installation of an external tool:

- Class diagram Ecore Model editor. This feature is provided by EclipseUML (Omondo) (a community edition is available)

Here is a screenshot of a typical Kermeta workbench.



```
aspect class State {

    reference combination : Set<State>


    // Go to the next state
    operation step(c : String) : Strin
    is do
        // Get the valid transitions
        var validTransitions : Collect
        validTransitions := outgoingTra
        // Check if there is one and o
        if validTransitions.empty then
        if validTransitions.size > 1 tl

        // Fire the transition
        result := validTransitions.one
    end

    // Create a new state from self sta
    operation copy() : State is do
        result := State.new
        result.name := String.clone(na
        result.combination := Set<Stat
```

```
class State
Container
        package fsm
inherits from
         kermeta::language::structure::Object
```

Figure 1.1. Kermeta workbench screenshoot (v1.2.2)

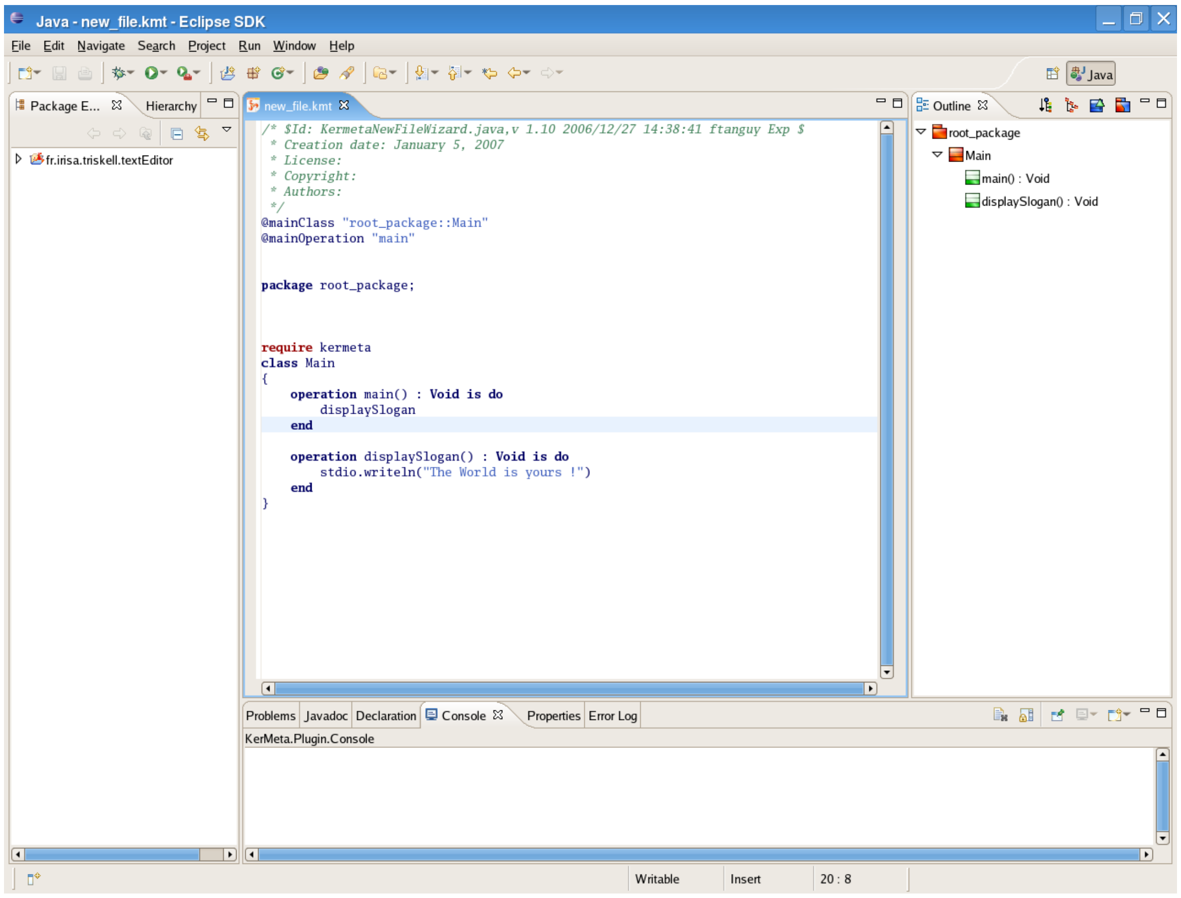
Warning

Kermeta is an evolving software and despite that we put a lot of attention to this document, it may contain errors or not conform to the latest version. If you find any error or have some information that improves this document, please send it to us using the bugtracker in the forge: http://gforge.inria.fr/tracker/?group_id=32 Last check: v1.2.2

CHAPTER 2 The Text Editor

2.1. The document

Nowadays a textual editor is not just an editable document. Many informations have been provided to the user in order to get improved editing performances. It is quite common to find textual editor with coloring syntax, auto-completion and outline. These three items working together usually makes a good textual editor. That is the reason why the Kermeta platform provides such an editor. Below, you can find the global view of the Kermeta editor.


Java - new_file.kmt - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Package E... Hierarchy
fr.irisa.triskell.textEditor
new_file.kmt
Outline
root_package
Main
main() : Void
displaySlogan() : Void
Problems Javadoc Declaration Console Properties Error Log
KerMeta.Plugin.Console
Writable Insert 20 : 8

```
/* $Id: KermetaNewFileWizard.java,v 1.10 2006/12/27 14:38:41 ftanguy Exp $
 * Creation date: January 5, 2007
 * License:
 * Copyright:
 * Authors:
 */
@mainClass "root_package::Main"
@mainOperation "main"


package root_package;



require kermeta
class Main
{
    operation main() : Void is do
        displaySlogan
    end

    operation displaySlogan() : Void is do
        stdio.writeln("The World is yours !")
    end
}
```

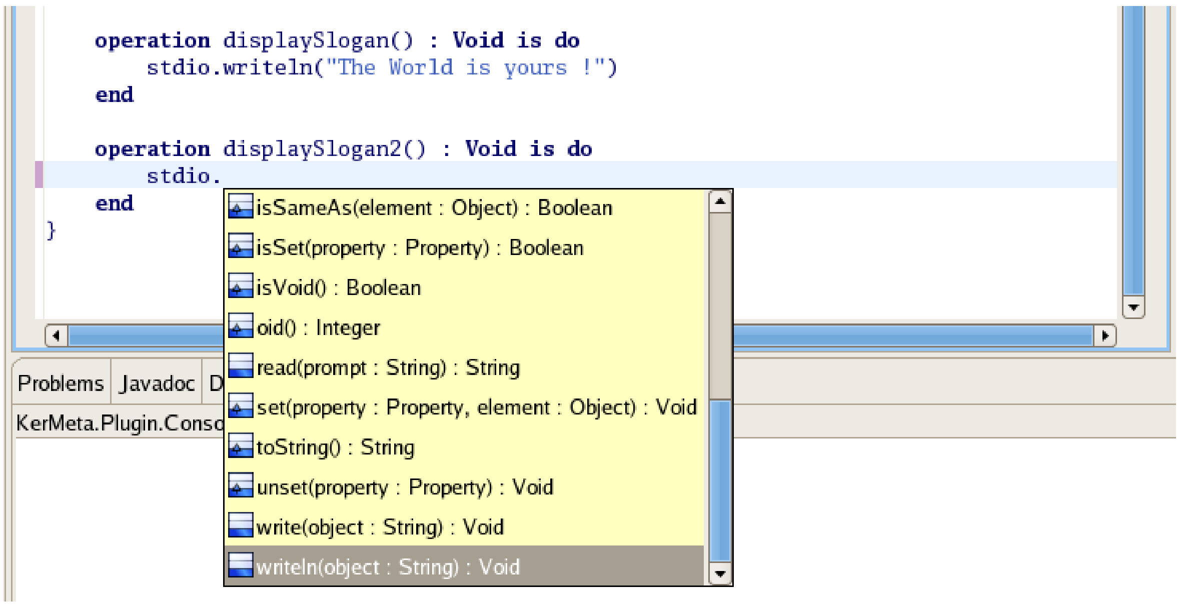
Figure 2.1. The Kermeta Textual Editor

2.2. Auto-completion

To develop faster and faster, some nice helpful tools now are usual. One of them is the auto-completion. According to a context, a list of item will be proposed. Using the arrows key and the carriage return key, you can select quickly and easily the item you want. The important thing is that, the list of item is calculated in order to only propose some relevant items. The figure below shows the auto-completion for "stdio" which is a special class in Kermeta (as System for Java). The typed point makes the system think that you want to call a method on "stdio". Then all the callable methods are proposed and you can choose one of them.

Note

The usual <ctrl>+<space> keyboard shortcut is used to access the code completion



```
    operation displaySlogan() : Void is do
        stdio.writeln("The World is yours !")
    end

    operation displaySlogan2() : Void is do
        stdio.
    end
}
```

```
isSameAs(element : Object) : Boolean
isSet(property : Property) : Boolean
isVoid() : Boolean
oid() : Integer
read(prompt : String) : String
set(property : Property, element : Object) : Void
toString() : String
unset(property : Property) : Void
write(object : String) : Void
writeln(object : String) : Void
```

Figure 2.2. Auto-Completion in the Kermeta Textual Editor

2.3. Outline

The outline is a structural view of your program. Code is not visible in this view, but the structure (packages, classes, attributes, methods and constraints) are visible in a tree hierarchy. Usually the outline position is on the top right side of the Eclipse's window. You can get advantages of the outline, when your program reaches a big size and when it is difficult to get a good visibility on the program's structure. Browsing the outline is much more faster than browsing the file. Moreover, clicking on an item (packages, classes, attributes or methods) brings you to that item in the file.

The following buttons are available on this view :

- sort features : The content of the outline is sorted alphabetically.

- Flatten inheritance : Show package hierarchy
- Display all the features of your classes. The inherited features are displayed using a specific icon set.
- Show imported types : Show all the classes that are declared using the require statement in this file.

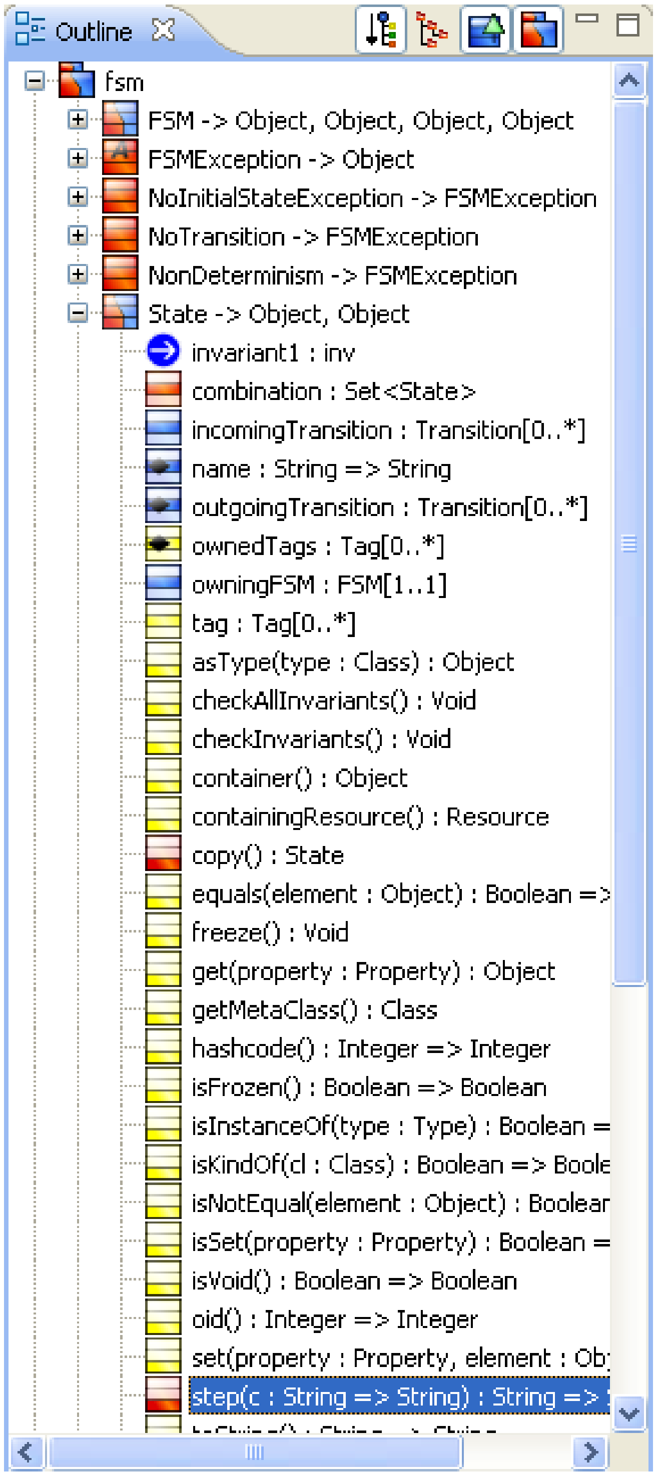
Note

The Flatten inheritance also display the features of the Object class as all the Kermeta classes inherit from it.

Warning

The "Show imported types" feature works on a per file basis and then it only shows the features declared using the require statement in this file and NOT all the files in your project. This option is really useful when you work with kermeta aspects since it display the elements in your faile and the elements imported by aspect.

In addition, the elements are colored in order to help to identify where and how an element is defined.



```
fsm
  FSM -> Object, Object, Object, Object
  FSMException -> Object
  NoInitialStateException -> FSMException
  NoTransition -> FSMException
  NonDeterminism -> FSMException
  State -> Object, Object
    invariant1 : inv
    combination : Set<State>
    incomingTransition : Transition[0..*]
    name : String => String
    outgoingTransition : Transition[0..*]
    ownedTags : Tag[0..*]
    owningFSM : FSM[1..1]
    tag : Tag[0..*]
    asType(type : Class) : Object
    checkAllInvariants() : Void
    checkInvariants() : Void
    container() : Object
    containingResource() : Resource
    copy() : State
    equals(element : Object) : Boolean =>
    freeze() : Void
    get(property : Property) : Object
    getMetaClass() : Class
    hashcode() : Integer => Integer
    isFrozen() : Boolean => Boolean
    isInstanceOf(type : Type) : Boolean =
    isKindOf(cl : Class) : Boolean => Boole
    isNotEqual(element : Object) : Boolear
    isSet(property : Property) : Boolean =
    isVoid() : Boolean => Boolean
    oid() : Integer => Integer
    set(property : Property, element : Ob
    step(c : String => String) : String =>
```

Figure 2.3. The outline in Kermeta Text Editor

The following meaning are assigned to the colors in this view :

- Red : The element is defined in the current file. It applies to : Packages, Classes, Attributes, References, Derived Properties and Operations.
- Blue : The element is defined in an imported (required) file. It applies to : Packages, Classes, Attributes, References, Derived Properties and Operations. On classes features, this typically represent features imported via an aspect.
- Yellow : The element is inherited from another class (regardless of the containment in the file). It applies to Attribute, Reference, Derived properties and Operations
- Green : The operation is imported via an aspect and is still abstract.
- Grey : The package doesn't define any class.

Tip

A double click will open the editor in order to show you the selected element. This feature works only if the element to display is in a kmt file.

Tip

If you have activated the KermetaDoc view, it will show the documentation associated to the selected element.

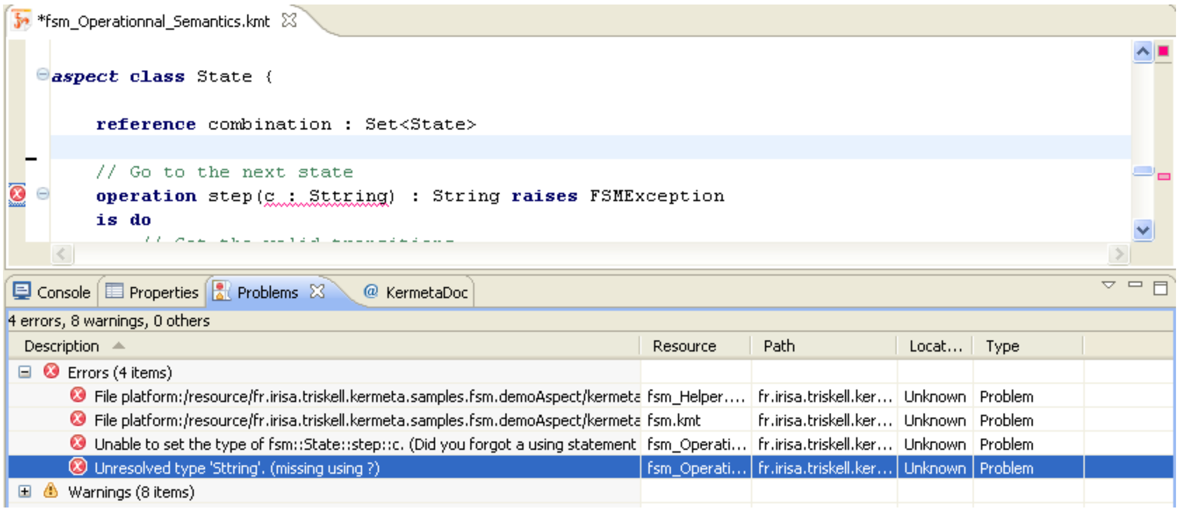
Note

In version 1.2.2, the constrains don't have a color differentiation.

2.4. Errors reporting

The textual editor is used to display extra pieces of information than simple text. In the figure below, a parsing error is reporting because of the semicolon which is not allowed in the language. The nice is thing is that in the Eclipse's problems view, just by double clicking on an error opens the erroneous file and make the cursor focusing on the correct line.

In addition, a list of all current errors is available in the general "Problems view".



```
aspect class State {

    reference combination : Set<State>

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

4 errors, 8 warnings, 0 others

Description	Resource	Path	Locat...	Type
Errors (4 items)				
File platform:/resource/fr.irisa.triskell.kermeta.samples.fsm.demoAspect/kermeta	fsm_Helper....	fr.irisa.triskell.ker...	Unknown	Problem
File platform:/resource/fr.irisa.triskell.kermeta.samples.fsm.demoAspect/kermeta	fsm.kmt	fr.irisa.triskell.ker...	Unknown	Problem
Unable to set the type of fsm::State::step::c. (Did you forgot a using statement	fsm_Operati...	fr.irisa.triskell.ker...	Unknown	Problem
Unresolved type 'Sttring'. (missing using ?)	fsm_Operati...	fr.irisa.triskell.ker...	Unknown	Problem
Warnings (8 items)				

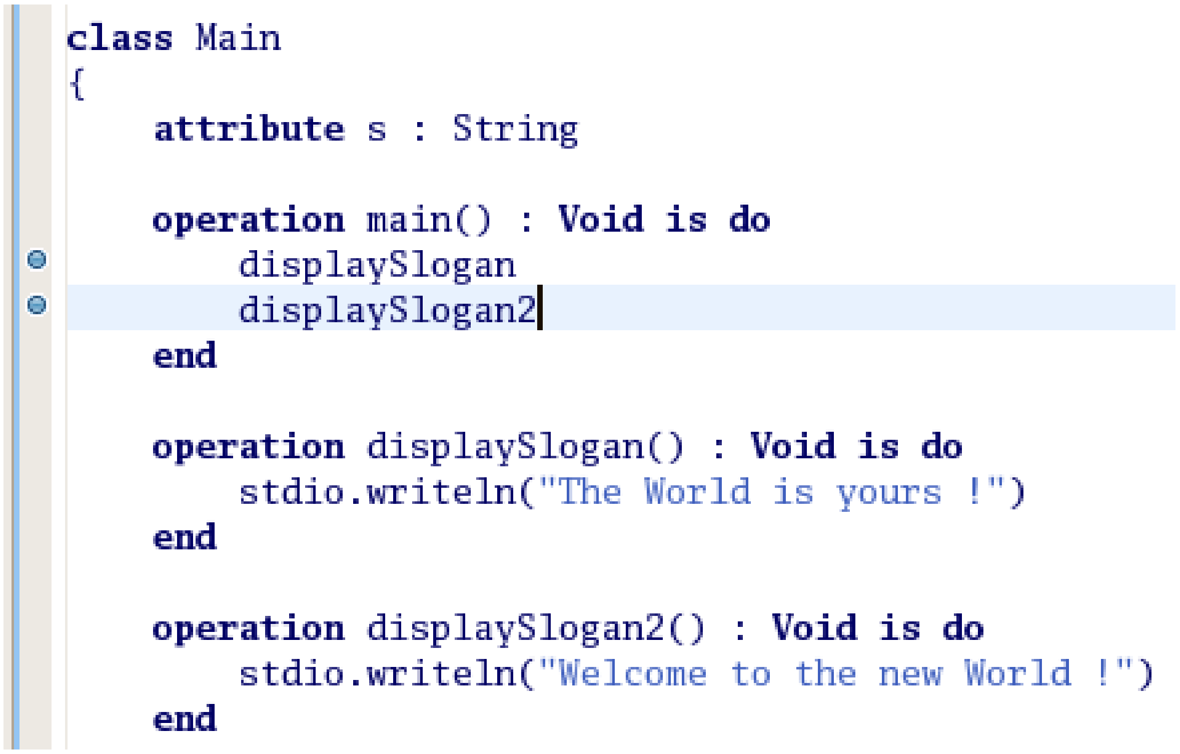
Figure 2.4. Error reporting

2.5. Debugging information

Kermeta is meant an executable language and provides a debugger. So, the textual editor is used to interact with the debugger.

Like in a Java style, some breakpoints can be added to the program by right clicking on the grey line at the left of the editor (as in the figure below). The breakpoint will be added in front of the selected line. Afterwards, when running the debugger, the program will halt at every breakpoint it goes through. At this moment, you will be able to have a look at the variables' value and run step by step or step into some methods.

See Chapter 5, *Debugging* for more details on the debugger features.

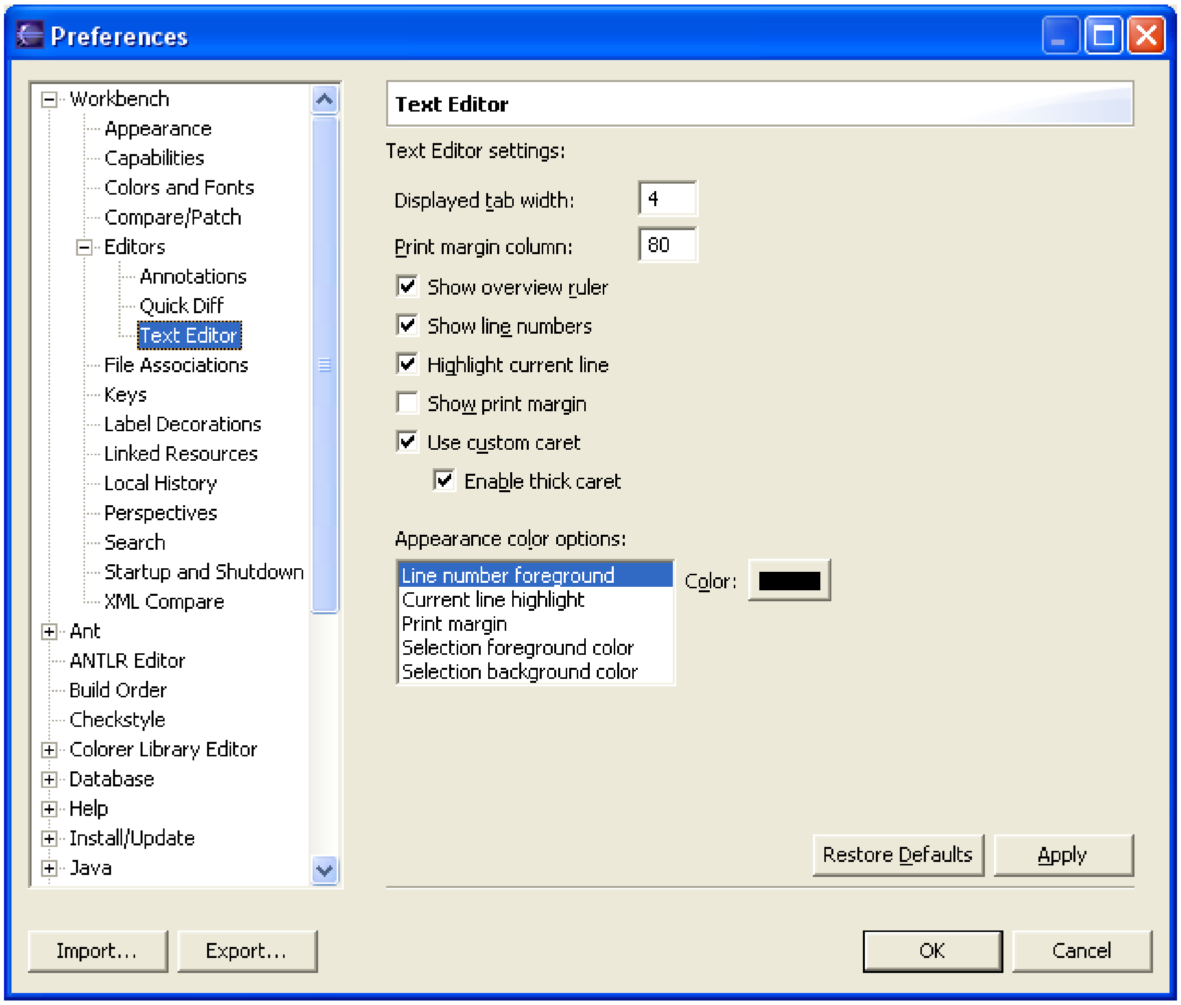


```
class Main
{
    attribute s : String

    operation main() : Void is do
        displaySlogan
        displaySlogan2
    end

    operation displaySlogan() : Void is do
        stdio.writeln("The World is yours !")
    end

    operation displaySlogan2() : Void is do
        stdio.writeln("Welcome to the new World !")
    end
```

Figure 2.5. Debugging Information in the Kermeta Text Editor

2.6. Editor preferences

You can customize your text editor by selecting Window from the Eclipse main menu and then Preferences. The image below shows the page configurator.



Text Editor

Text Editor settings:

Displayed tab width: 4

Print margin column: 80

- [✓] Show overview ruler
- [✓] Show line numbers
- [✓] Highlight current line
- [ ] Show print margin
- [✓] Use custom caret
  - [✓] Enable thick caret

Appearance color options:

- Line number foreground
- Current line highlight
- Print margin
- Selection foreground color
- Selection background color

Color: ■

Figure 2.6. TextEditor preferences

CHAPTER 3

Interpreting a Kermeta code

Kermeta workbench allows to interpret and run your Kermeta code in several ways.

3.1. Using a Run configuration

For those who already know how to run Java applications, this is a similar approach but for Kermeta files.

You'll need to create a new run configuration from the "`Run/Run...`" menu.

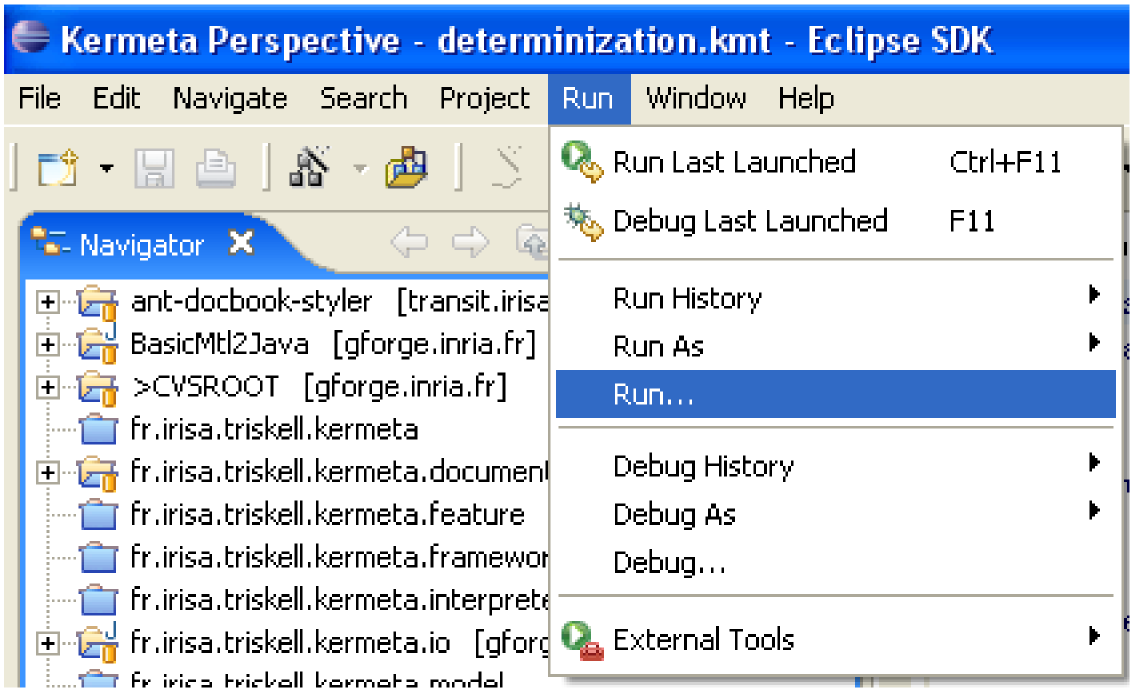

Kermeta Perspective - determinization.kmt - Eclipse SDK
File
Edit
Navigate
Search
Project
Run
Window
Help
Navigator
ant-docbook-styler [transit.irisa
BasicMtl2Java [gforge.inria.fr]
>CVSROOT [gforge.inria.fr]
fr.irisa.triskell.kermeta
fr.irisa.triskell.kermeta.document
fr.irisa.triskell.kermeta.feature
fr.irisa.triskell.kermeta.framewor
fr.irisa.triskell.kermeta.interprete
fr.irisa.triskell.kermeta.io [gforg
Run Last Launched Ctrl+F11
Debug Last Launched F11
Run History
Run As
Run...
Debug History
Debug As
Debug...
External Tools

Figure 3.1. Run menu

Caution

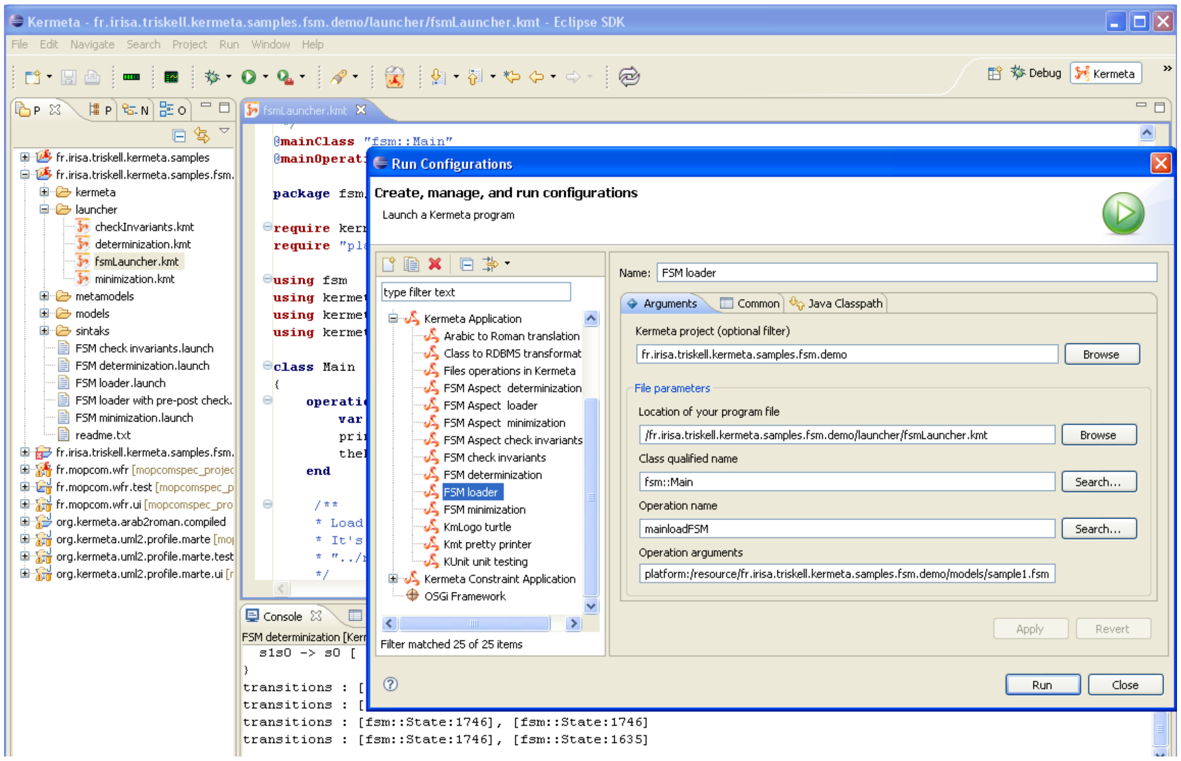
The run menu is available only with some perspectives. If you don't see the run menu, switch to the Kermeta perspective.

In the dialog, you selects the file, the main class and the main operation that must be run.

Search buttons will help you to find these elements in your files.

Caution

The search button may not work properly if your file contains errors.


Kermeta - fr.irisa.triskell.kermeta.samples.fsm.demo/launcher/fsmLauncher.kmt - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Debug
Kermeta
fsmLauncher.kmt
Run Configurations
Create, manage, and run configurations
Launch a Kermeta program
type filter text
Arguments
Common
Java Classpath
Browse
Search...
Apply
Revert
Run
Close
Console

fr.irisa.triskell.kermeta.samples
fr.irisa.triskell.kermeta.samples.fsm.
kermeta
launcher
checkInvariants.kmt
determinization.kmt
fsmLauncher.kmt
minimization.kmt
metamodels
models
sintaks
FSM check invariants.launch
FSM determinization.launch
FSM loader.launch
FSM loader with pre-post check.
FSM minimization.launch
readme.txt
fr.irisa.triskell.kermeta.samples.fsm.
fr.mopcom.wfr [mopcomspec_projec
fr.mopcom.wfr.test [mopcomspec_p
fr.mopcom.wfr.ui [mopcomspec_pro
org.kermeta.arab2roman.compiled
org.kermeta.uml2.profile.marte [mo
org.kermeta.uml2.profile.marte.test
org.kermeta.uml2.profile.marte.ui [r

```
@mainClass "fsm::Main"
@mainOperati
package fsm;
require ker
require "pl
using fsm
using kermet
using kermet
using kermet
class Main
{
    operati
        var
        pri
        the
    end
    /**
     * Load
     * It's
     * "../
     */
```

Kermeta Application
Arabic to Roman translation
Class to RDBMS transformat
Files operations in Kermeta
FSM Aspect determinization
FSM Aspect loader
FSM Aspect minimization
FSM Aspect check invariants
FSM check invariants
FSM determinization
FSM loader
FSM minimization
KmLogo turtle
Kmt pretty printer
KUnit unit testing
Kermeta Constraint Application
OSGi Framework

Filter matched 25 of 25 items

Name: FSM loader

Kermeta project (optional filter): fr.irisa.triskell.kermeta.samples.fsm.demo

File parameters

Location of your program file: /fr.irisa.triskell.kermeta.samples.fsm.demo/launcher/fsmLauncher.kmt

Class qualified name: fsm::Main

Operation name: mainloadFSM

Operation arguments: platform:/resource/fr.irisa.triskell.kermeta.samples.fsm.demo/models/sample1.fsm

```
FSM determinization [Kern
  s1s0 -> s0 [
}
transitions : [
transitions : [
transitions : [fsm::State:1746], [fsm::State:1746]
transitions : [fsm::State:1746], [fsm::State:1635]
```

Figure 3.2. Kermeta run configuration - main tab

Two optional annotations in your Kermeta code allow you to save time while creating run configurations. The @mainClass and @mainOperation are used to fill field when creating a new configuration.

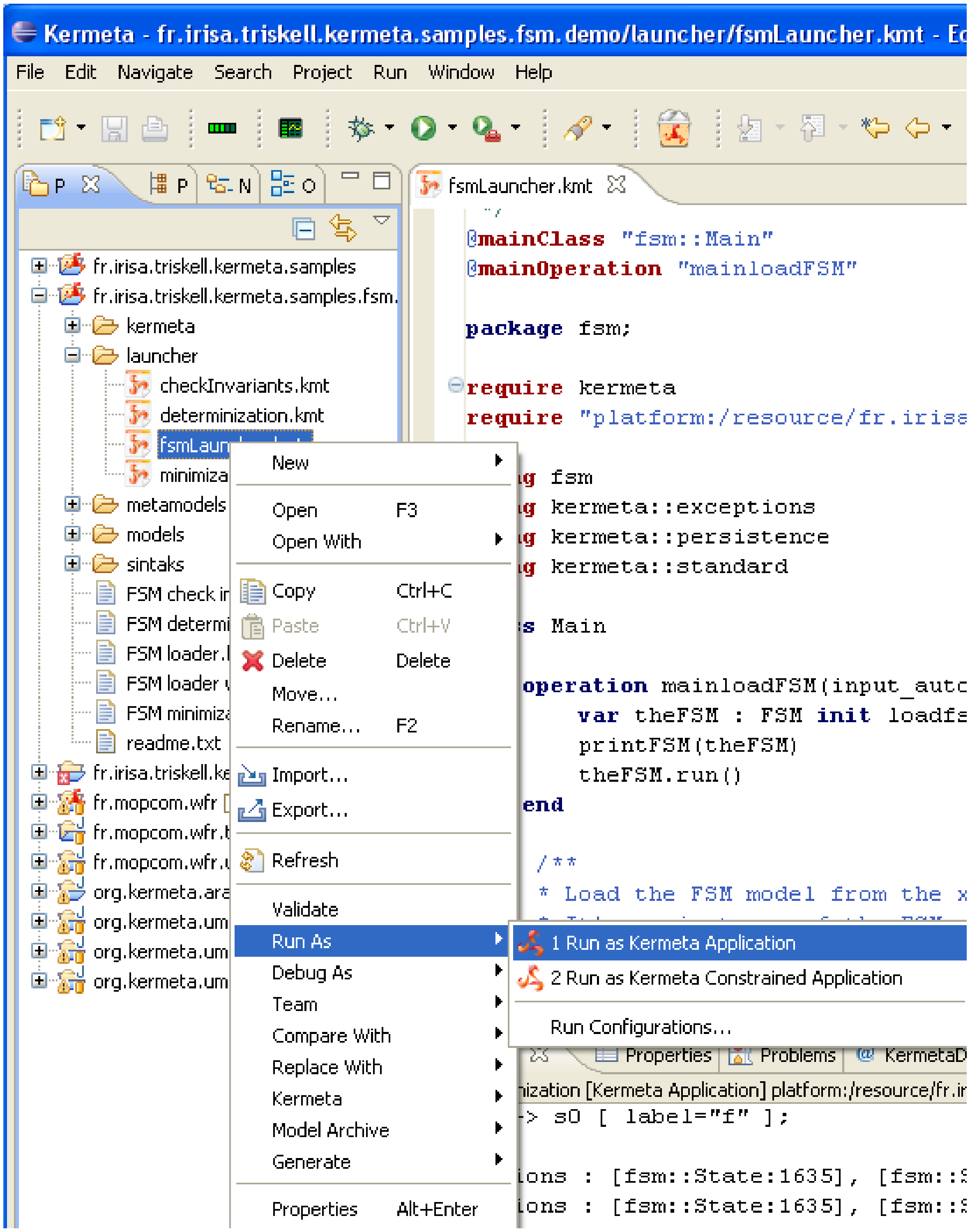
Ex:

```
@mainClass "fsm::Main"
@mainOperation "mainloadFSM"
```

Then, once you have your configuration, you can run it by all the means provided by Eclipse. Ie. "`Run Last Launched`", "`Run History`", ... available in the menus or in the toolbars.

Tip

A shortcut allows you to quickly start a Kermeta run configuration. It is available in the popup menu of *.kmt files. The "`Run As/Kermeta App`" button runs an existing configuration and the "`Run As/Run...`" button opens the configuration dialog.



```
@mainClass "fsm::Main"
@mainOperation "mainloadFSM"

package fsm;

require kermeta
require "platform:/resource/fr.irisa

g fsm
g kermeta::exceptions
g kermeta::persistence
g kermeta::standard

s Main

operation mainloadFSM(input_autc
    var theFSM : FSM init loadfs
    printFSM(theFSM)
    theFSM.run()
end

 /**
 * Load the FSM model from the x
```

```
nization [Kermeta Application] platform:/resource/fr.ir
-> s0 [ label="f" ];

ions : [fsm::State:1635], [fsm::S
ions : [fsm::State:1635], [fsm::S
```

Figure 3.3. Run shortcut to run configuration on the kmt files

The operation arguments field allow you to pass some strings as parameters of the operation you are launching.

In this sample screenshot, the FSM take a string as teh path to the file that will be loaded and simulated.

In addtion to this simple scenario, in the common Tab of the Kermeta run configuration, you can:

- save your configuration in a file, so you can share it with other users: in "Common" tab button "Shared file"
- specify where the stdio outputs must be sent. Typically, this can be in the Eclipse console or in a file.
- specify additional entries in the interpreter classpath that must be taken into while running this kermeta code. This is needed if you use "extern" call to your own jars.

Tip

If you specify the kermeta project u=in the run configuration and if this project is an eclipse plugin, then the interpreter will automatically take this plugin and all its dependencies into account in tha classpath.

3.2. Run the interpreter with a Java command line

You can run Kermeta interpreter from the command line. For this you need to run the java class `org.kermeta.interpreter.api.RunCommandLine` which is available in `fr.irisa.triskell.kermeta.interpreter` plugin. The RunCommandLine class allows to pass the name of: the Kermeta source file, and optionally the name of the class and the operation to run.

Note

The package org.kermeta.interpreter.api provides different facades in order to run a Kermeta program with the interpreter from a java program. You can either have a look to the document Deploying Kermeta code manual or you may also have a look into class `RunCommandLine` to see how it was implemented.

The interpreter command line is available in two main forms : as a set of eclipse plugin (it is available with all kermeta version deployed in Eclipse, or as a standalone jar (see Kermeta download page for the various distributions).

The standalone bundle is available in two version : all in one jar (typically `kermeta_standalone_full_1.3.0.jar`), or only kermeta in the main jar (typically `kermeta_standalone_no_thirdparty_in_main_jar.1.3.0.zip`. In this second version, the thirdparty jar aren't included.

Depending wether you use the standalone jars or the jar embedded in eclipse, the process will be more or less simple. The simpliest is to use the standalone version because you don't need to deal with the classpath.

```
java -jar kermeta_standalone.jar  -U ecore_mdk/add_observer.kmt -C pattern::AddObserver -O main❶
        input_models/needs_observer.ecore output_models/ Compteur Afficheur❷
```

❶ This sample launches the file `add_observer.kmt` and will start the operation `main` of the class `pattern::AddObserver`.

❷ This sample kermeta programs takes 4 strings as parameters.

Example 3.1. Simpliest use of the standalone version on the command line

3.2.1. Command line options

The command line needs several options to run properly a kermeta program.

.

- -C : (takes 1 argument) Qualified name of the class to run

  ex: myPackage::MyClass

  Note that it will not instanciate the class (ie. self will return void)

- -H : (takes 0 argument) Display the help

- -T : (takes 0 argument) Run as a test suite

- -P : (takes 0 argument) Display Profiling info

- -K : (takes 1 argument) Location for kermeta standard unit(framerwork.km) (KermetaUnit.STD_LIB_URI variable)

- -O : (takes 1 argument) Name of the operation to run

- -U : (takes 1 argument) URI of the unit file that must be loaded

- -M : (takes 1 argument) Indicate a file containing URI_MAP (This is useful to solve some model loading troubles)

Other values on the command line will be passed as string to the operation

```
java -jar kermeta_standalone.jar  -P -U ../ModelEngineeringCourse/transfo/add_observer.kmt -C pattern_macros::Main -O main❶
      -K lib/framework.km❷ ../models/needs_observer.ecore Compteur Afficheur❸
```

❶ This sample launches the file `add_observer.kmt` and will start the operation `main` of the class `pattern_macros::Main`.
❷ forces the interpreter to use a specific framework.km instead of the one inside the jar.
❸ This sample kermeta programs takes 3 strings as parameters.

Example 3.2. Example of a kermeta program called with the standalone version on the command line

3.2.2. About the URI map

When you run a kermeta program outside of Eclipse, the uri of type platform:// are not resolved since this is the role of Eclipse to resolve them. So, when running outiside of Eclipse, you need to help this resolution.

The resolution can be done using a map, whenever it find the key, it will replace it with the associated value. Like that you can transform platform:/plugin/ into a physical location in your disk.

With RunCommandLine.java, this is the option `-U` which allow to specify a file that contains the mapping.

This uri map file uses the syntax of java properties files.

Example of uri.map file which maps some platform:// urls into file:// urls

```
#Typical uri.map file
"platform:/plugin/"   "file:/C:/eclipse3.4/eclipse/workspace_head/"
"platform:/resource/" "file:/C:/eclipse3.4/eclipse/workspace_head/"
"kconf:/loader/"      "file:/C:/eclipse3.4/eclipse/workspace_head/org.kermeta.io.loader/instances/" ❶
```

❶ This special uri is used by kermeta to retrieve the folder `instances` in the plugin `org.kermeta.io.loader`. Kermeta intepreter cannot work correctly without it.

Another easy way (not detailed here) would be to retrieve one of the standalone versions and run it from eclipse.

Tip

With this map mechanism, you can also translate the platform uris into jar protocol. This protocol allows to search within a jar. for example, `jar:file:/my.jar!/src/kermeta/framework.km` opens the file `my.jar` and pick the file named `framework.km` into the subfolder `src/kermeta` in the jar.

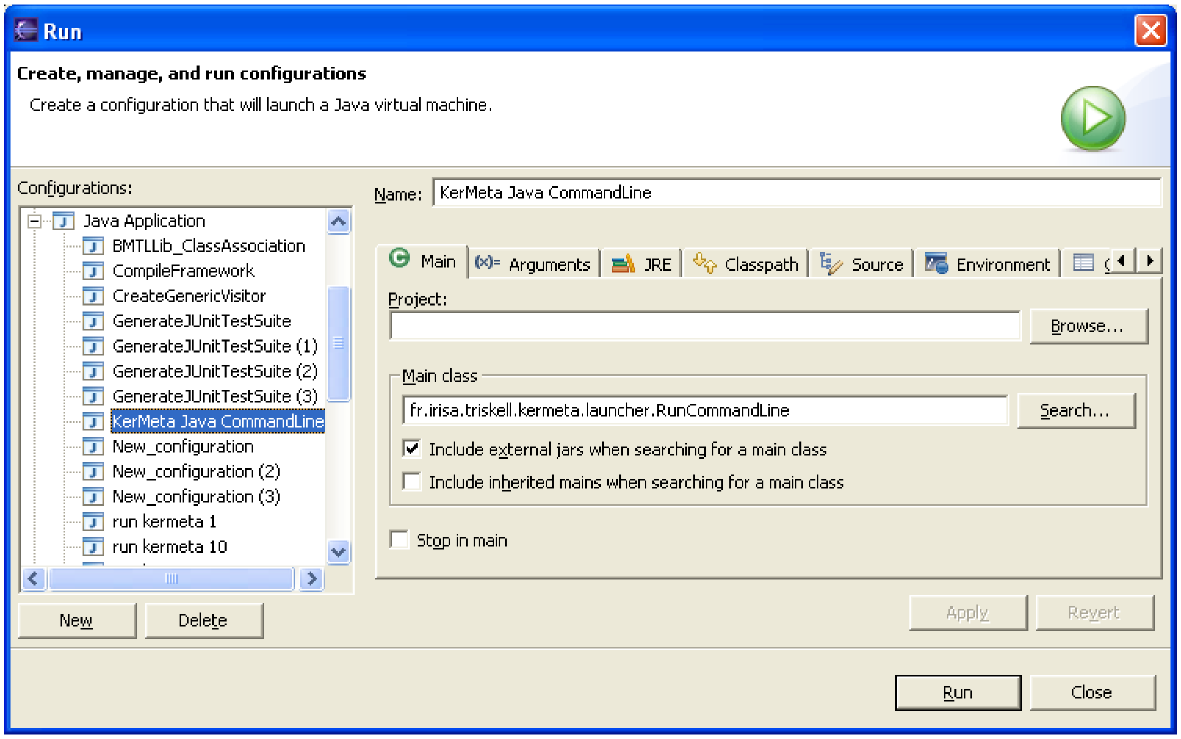
3.2.3. Run the eclipse version of the interpreter in eclipse

In this section we focus on calling the interpreter embedded in an eclipse installation.

Note

This is quite an advanced use case recommanded only for people who understand how eclipse files are organized and can manually create a classpath to retreive the various jar in eclipse installation.

From eclipse, you create a Java application LaunchConfiguration that will run the interpreter with your Kermeta file.



Name: KerMeta Java CommandLine

Project:

Main class: fr.irisa.triskell.kermeta.launcher.RunCommandLine

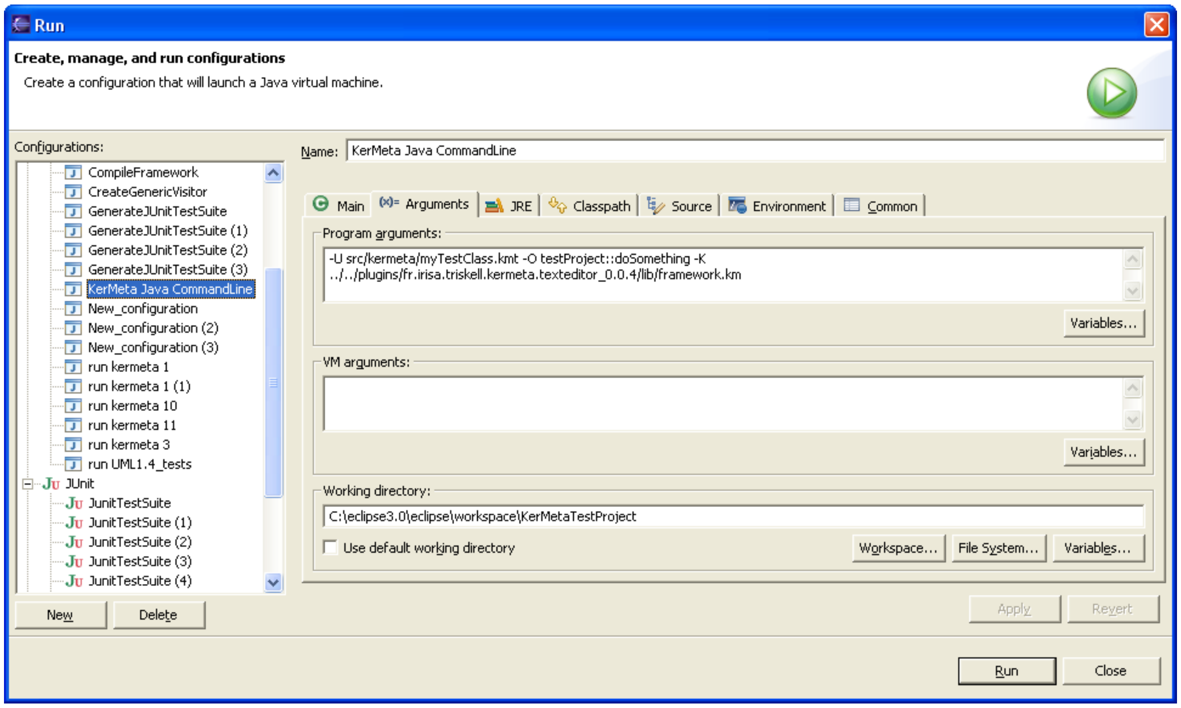
[✓] Include external jars when searching for a main class

[ ] Include inherited mains when searching for a main class

[ ] Stop in main

Figure 3.4. Java run configuration main tab

Be careful, you must select "Include external jars when searching for a main class".



Name: KerMeta Java CommandLine

Program arguments:

```
-U src/kermeta/myTestClass.kmt -O testProject::doSomething -K
../../plugins/fr.irisa.triskell.kermeta.texteditor_0.0.4/lib/framework.km
```

VM arguments:

Working directory: C:\eclipse3.0\eclipse\workspace\KerMetaTestProject

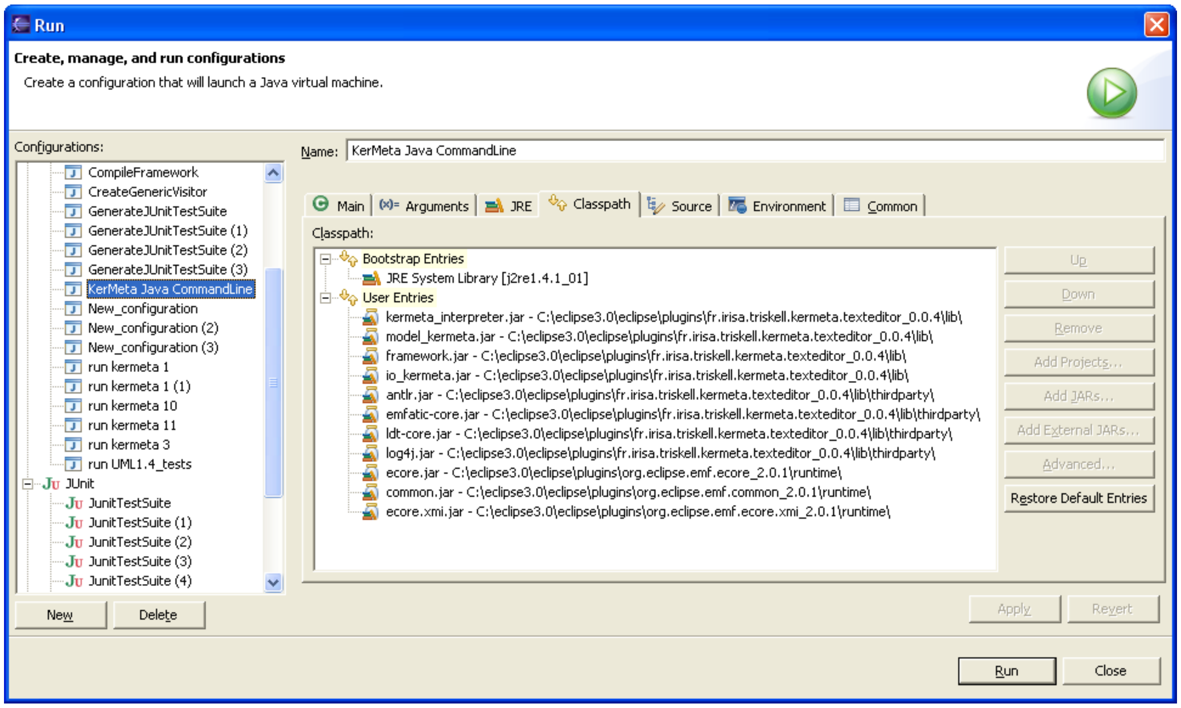
[ ] Use default working directory

Figure 3.5. Java run configuration arguments

The output level of interpreter traces can be set using a log4j configuration file. The concrete file to use is set by the java property "kermeta.log4j.configurationfile.name". A sample of log4j configuration file is located in the `fr.irisa.triskell.kermeta.model` plugin.

Exemple add the following in the VM arguments box: -Dkermeta.log4j.configurationfile.name= "../../plugins/fr.irisa.triskell.kermeta.model_1.3.0/kermeta_log4j_configuration.xml"

The classpath must be carefully set using the jar of Kermeta, the jar of Ecore and of emf.



Name: KerMeta Java CommandLine

Classpath:

```
Bootstrap Entries
    JRE System Library [j2re1.4.1_01]
User Entries
    kermeta_interpreter.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\
    model_kermeta.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\
    framework.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\
    io_kermeta.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\
    antlr.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\thirdparty\
    emfatic-core.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\thirdparty\
    ldt-core.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\thirdparty\
    log4j.jar - C:\eclipse3.0\eclipse\plugins\fr.irisa.triskell.kermeta.texteditor_0.0.4\lib\thirdparty\
    ecore.jar - C:\eclipse3.0\eclipse\plugins\org.eclipse.emf.ecore_2.0.1\runtime\
    common.jar - C:\eclipse3.0\eclipse\plugins\org.eclipse.emf.common_2.0.1\runtime\
    ecore.xmi.jar - C:\eclipse3.0\eclipse\plugins\org.eclipse.emf.ecore.xmi_2.0.1\runtime\
```

Figure 3.6. Java run configuration classpath (using the interpreter jars in the texteditor)

Tip

This RunCommandLine class or one of its sibling can also be used by any of your java programs.For more information please see the Javadoc reference of RunCommandLine.

Warning

This method need to know precisely which jar you use. If you update any of the required plugins (kermeta, emf, ecore , ...) the directory containing the jar will change because eclipse use the version number as part of the directory name. You'll have to manually change to ant file in order to reflect the update.

CHAPTER 4

Compiling a kermeta code into java

The workbench allows you to transform you kermeta code into Java-EMF code.

4.1. In a nutshell

The Kermeta compiler consists on a transformation from a kmt file to an Ecore file (step 1), finally the generation a plugin with the Java by using the EMF facilities (step 2). The resulting Java may be used both in Eclipse Application or in Java Application (standalone) and is fully compatible with EMF. A listing of dependencies is available below. The compiling process generates a set of files: *.km, *.traceability, *.ecore, *.simk, *.genmodel. A configuration file may be added: *.compiler.properties (more info. below). The 2nd step may be replayed by a right-click on the generated Ecore file.

4.2. Calling the complier

4.2.1. Launching a compilation process

Right click on a kmt file: "Compile Kmt to EMF plugin (Experimental)", then an Eclipse plugin will be generated, after that you can try to run one generated main contained in runner Java package.

4.2.2. Customizing the plugin generation

You are able to add a "properties file" for customizing the plugin generation. This file must be in the same folder than the main kmt file and respect the naming rule: {kmt_file_name_without_the_kmt_extension}.compiler.properties This configuration file will be taking into account, when you are compiling from a kmt file or the generated ecore file, for this last case you are able to re-generate the plugin without redo kmt2{ecore, simk} The available properties are listed below, every property is optional (the values are String without quote):

- plugin_id = {name and id of the generated plugin}
- copyright_header = {header include in each file}

- require_bundles = {plugin dependencies if your program depends on external plugins} | String separated by comma
- bundle_version = {version number of your plugin like 0.1.0 instead of the default 1.0.0}
- main_operations = {operation qualifiedNames} | String separated by comma, e.g.: mypackage::MyClass::myMainOperation, mypackage::MyClass::myMainOperation2
- unzip_externs = {zip file containing the externs required to the compiled source code} | String separated by comma, the String is composed of 2 parts separated by a ";", the first part is the location of the zip file, the second one is the folder in the generated plugin, ex.: platform:/resource/MyProject/util/externs.zip;src/java
- enable_emf_load_initialization = {enable EMF-reflection load initialization for large model} | String equals to true or false

4.3. Running the compiled code (RunConfiguration)

4.3.1. Running the compiled code inside Eclipse in development mode

Right-click on the generated class containing the Java Main method or you are able to create a "Java Run Configuration" pointing the Java Main method

4.3.2. Running the compiled code inside Eclipse in deployed mode

TODO

4.3.3. Running the compiled code outside Eclipse

Creating a standalone version of your compiled code, required libs:

- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.emf.common_2.4.0.v200808251517.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.emf.ecore.xmi_2.4.1.v200808251517.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.emf.ecore_2.4.1.v200808251517.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.core.resources_3.4.0.v20080604-1400.jar"
- classpathentry kind="lib"

path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.core.runtime_3.4.0.v20080512.jar"

- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.emf.codegen_2.4.0.v200808251517.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.emf.ecoretools.registration_1.0.0.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.equinox.registry_3.4.0.v20080516-0950.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/fr.irisa.triskell.eclipse.util_1.2.0.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.equinox.common_3.4.0.v20080421-2006.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.core.jobs_3.4.0.v20080512.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/fr.irisa.triskell.kermeta.model_1.2.0.jar"
- classpathentry kind="lib" path="C:/eclipse-ganymede/eclipse/plugins/org.eclipse.osgi_3.4.0.v20080605-1900.jar"

Note

This listing is in progress. Also the exact version of the plugins used may vary depending of your version of Keremta and Eclipse.

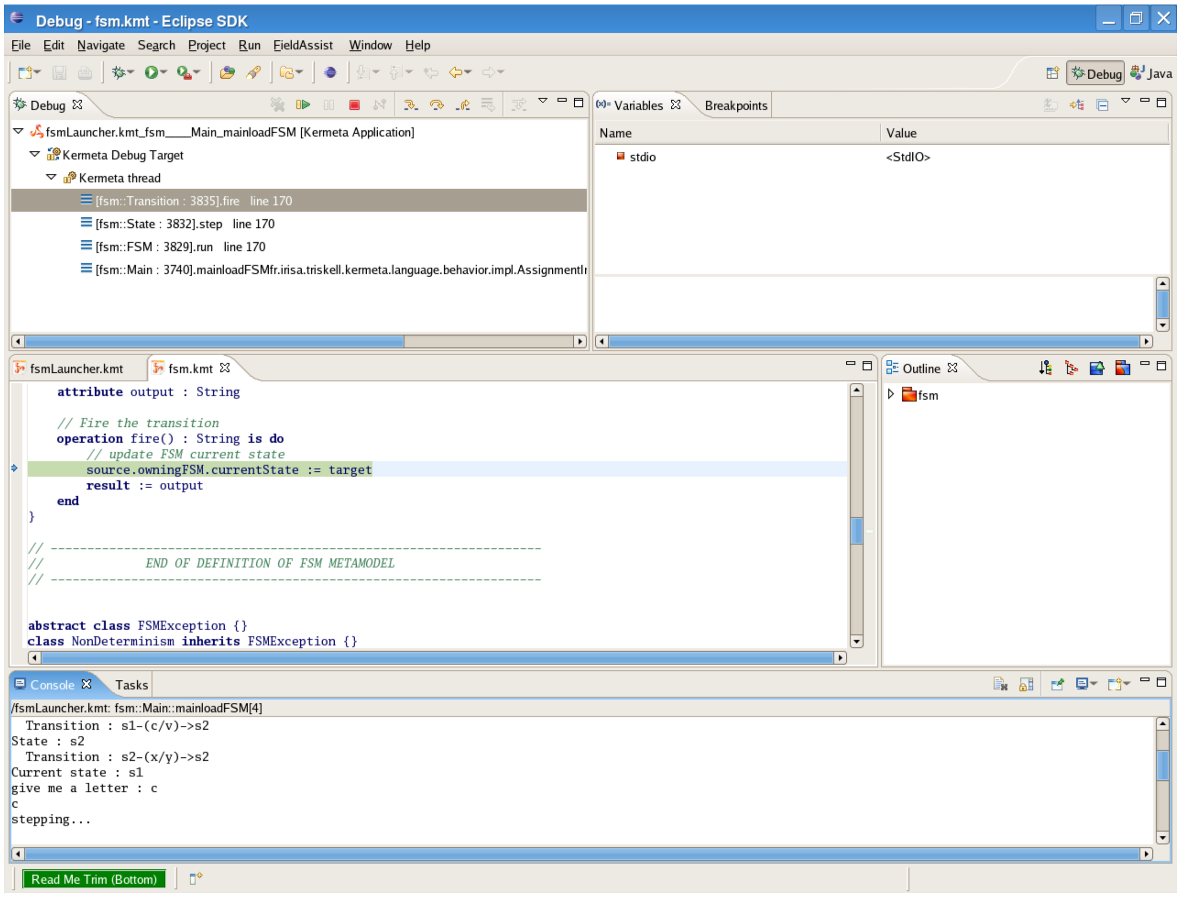
CHAPTER 5

Debugging

Since v0.1.0 Kermeta has a debugger. As any debugger, it proposes breakpoints step over, step into, run, resume, visualization of variables, etc

5.1. Global View

A debugger offers users the possibility to interpret Kermeta's programs step by step, or stopping programs at one moment to check the the programs' state. Above you can see the global view of the editor.



```
fsmLauncher.kmt_fsm____Main_mainloadFSM [Kermeta Application]
  Kermeta Debug Target
    Kermeta thread
      [fsm::Transition : 3835].fire   line 170
      [fsm::State : 3832].step   line 170
      [fsm::FSM : 3829].run   line 170
      [fsm::Main : 3740].mainloadFSMfr.irisa.triskell.kermeta.language.behavior.impl.AssignmentI
```

Name	Value
stdio	<StdIO>

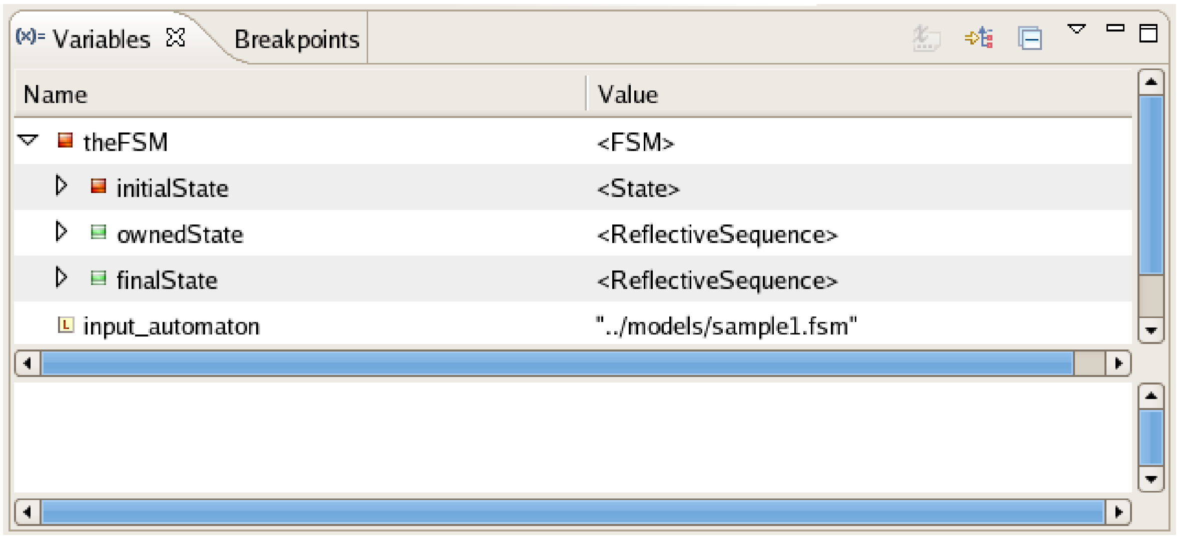
```
    attribute output : String

    // Fire the transition
    operation fire() : String is do
        // update FSM current state
        source.owningFSM.currentState := target
        result := output
    end
}

// -------------------------------------------------------------------
//                 END OF DEFINITION OF FSM METAMODEL
// -------------------------------------------------------------------


abstract class FSMException {}
class NonDeterminism inherits FSMException {}
```

```
/fsmLauncher.kmt: fsm::Main::mainloadFSM[4]
  Transition : s1-(c/v)->s2
State : s2
  Transition : s2-(x/y)->s2
Current state : s1
give me a letter : c
c
stepping...
```

Figure 5.1. The Debugger global view

5.2. The Watch View

During programs execution, you may want to know variables' values. For that purpose, you can use the watch view which displays all the variables accessible in the current context and their values. Moreover you can browse those variables to have a deeper look inside variables.

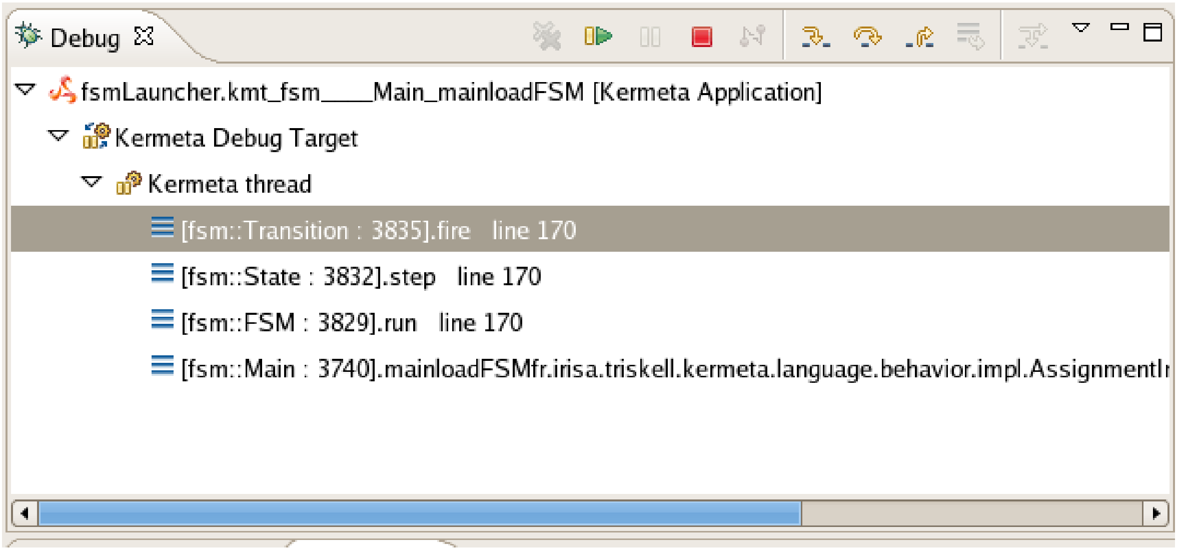


Name	Value
theFSM	<FSM>
initialState	<State>
ownedState	<ReflectiveSequence>
finalState	<ReflectiveSequence>
input_automaton	"../models/sample1.fsm"

Figure 5.2. The Watch View

5.3. The Frame View

A frame is a context environment for a call method. That means that for each method call, you have access to its context that is to say the variables values. Obviously, it is strongly related to the watch view. Clicking on one frame modifies the watch view. The frame view is also nice to follow the execution. For instance, in the figure above, we know that the run method has called the step method which has called the fire method.

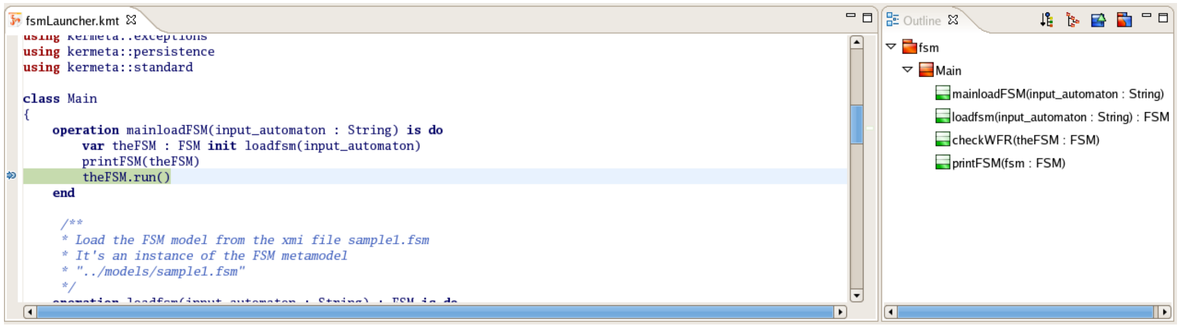


```
fsmLauncher.kmt_fsm____Main_mainloadFSM [Kermeta Application]
    Kermeta Debug Target
        Kermeta thread
            [fsm::Transition : 3835].fire   line 170
            [fsm::State : 3832].step   line 170
            [fsm::FSM : 3829].run   line 170
            [fsm::Main : 3740].mainloadFSMfr.irisa.triskell.kermeta.language.behavior.impl.AssignmentI
```

Figure 5.3. The Frame View

5.4. The Editor View

When debugging, you surely want to know which line of code is executed. That is the reason why the Kermeta text editor is displayed with the outline. The current line of code is colored in green as in the following figure. This green line is moving to the next instruction depending your action.



```
using kermeta::exceptions
using kermeta::persistence
using kermeta::standard

class Main
{
    operation mainloadFSM(input_automaton : String) is do
        var theFSM : FSM init loadfsm(input_automaton)
        printFSM(theFSM)
        theFSM.run()
    end

     /**
     * Load the FSM model from the xmi file sample1.fsm
     * It's an instance of the FSM metamodel
     * "../models/sample1.fsm"
     */
    operation loadfsm(input_automaton : String) : FSM is do
```

```
fsm
    Main
        mainloadFSM(input_automaton : String)
        loadfsm(input_automaton : String) : FSM
        checkWFR(theFSM : FSM)
        printFSM(fsm : FSM)
```

Figure 5.4. The Editor View

5.5. Debugging actions

There are three kind of actions you can execute:

- F5 : pressing the F5 key makes the debugger entering in the current operation.

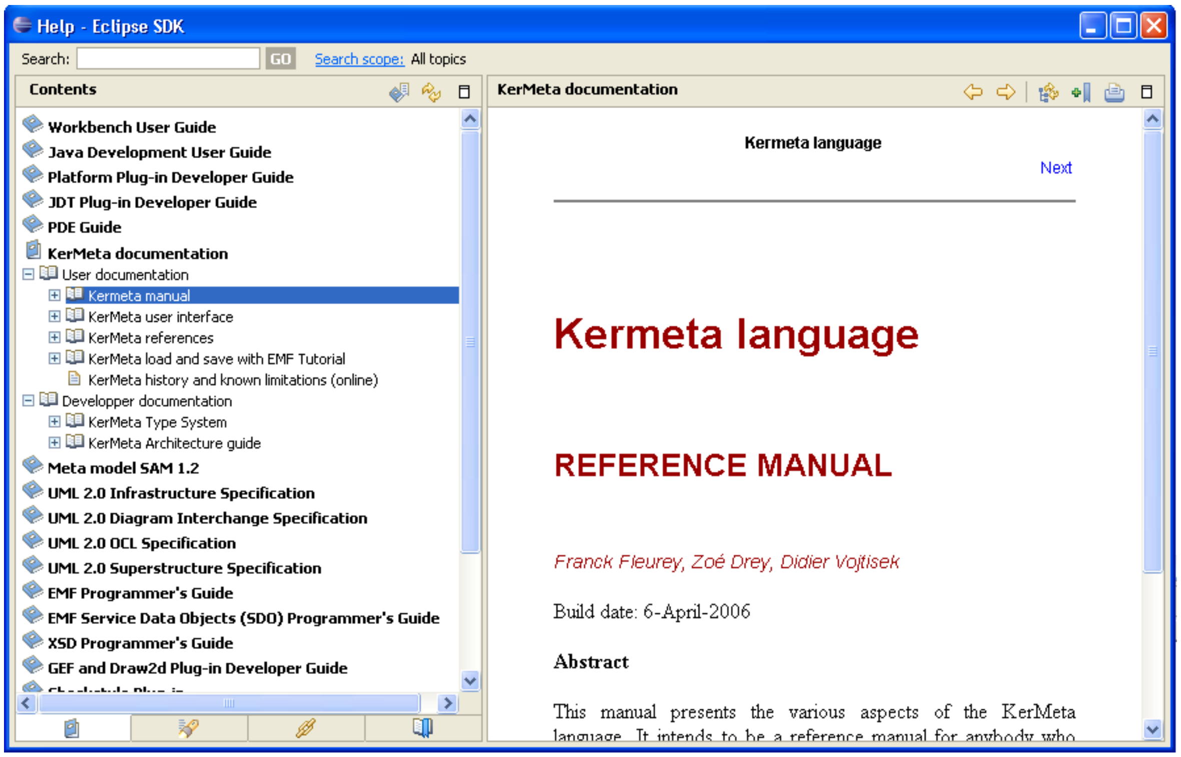
- F6 : pressing the F6 key makes the debugger stepping to the next operation.
- F8 : pressing the F8 key makes the debugger executing all the operations to the next breakpoint, if there is some one.

Note

A nice feature is that you can add or remove breakpoints to your Kermeta file during debugging.

CHAPTER 6
Help

The current help and many other documentation are available in the Eclipse Help system of your Eclipse installation: open Help -> Help Contents


Help - Eclipse SDK

Search: GO Search scope: All topics

Contents

- Workbench User Guide
- Java Development User Guide
- Platform Plug-in Developer Guide
- JDT Plug-in Developer Guide
- PDE Guide
- KerMeta documentation
  - User documentation
    - Kermeta manual
    - KerMeta user interface
    - KerMeta references
    - KerMeta load and save with EMF Tutorial
    - KerMeta history and known limitations (online)
  - Developper documentation
    - KerMeta Type System
    - KerMeta Architecture guide
- Meta model SAM 1.2
- UML 2.0 Infrastructure Specification
- UML 2.0 Diagram Interchange Specification
- UML 2.0 OCL Specification
- UML 2.0 Superstructure Specification
- EMF Programmer's Guide
- EMF Service Data Objects (SDO) Programmer's Guide
- XSD Programmer's Guide
- GEF and Draw2d Plug-in Developer Guide

KerMeta documentation

Kermeta language

Next

Kermeta language

REFERENCE MANUAL

Franck Fleurey, Zoé Drey, Didier Vojtisek

Build date: 6-April-2006

Abstract

This manual presents the various aspects of the KerMeta language. It intends to be a reference manual for anybody who

Figure 6.1. Eclipse Help Contents for Kermeta

Tip

Alternatively, an online version of all those documents is available at http://www.kermeta.org/documents

Chapter 7
Transformations

The workbench provides several additional tools available directly on the files using the popup menu.

Important

Several of the wizards provided are multipages wizards. This means that several advanced options are available only after clicking on "Next" instead of clicking on "Finish". So if you get an error while transforming a file, make sure that none of the options (on the main page or on the "Next" pages) fix the default behavior for your special need.

7.1. Actions for Ecore files

- **Export functions:** On ecore file you can export it into a KM (kermeta program in xmi format) or in kmt (Kermeta program using its textual syntax). As the ecore model may not be valid, the wizard provides some quick fixes. Those quick fixes will for example detect and propose a default solution for multiple operations with the same name.

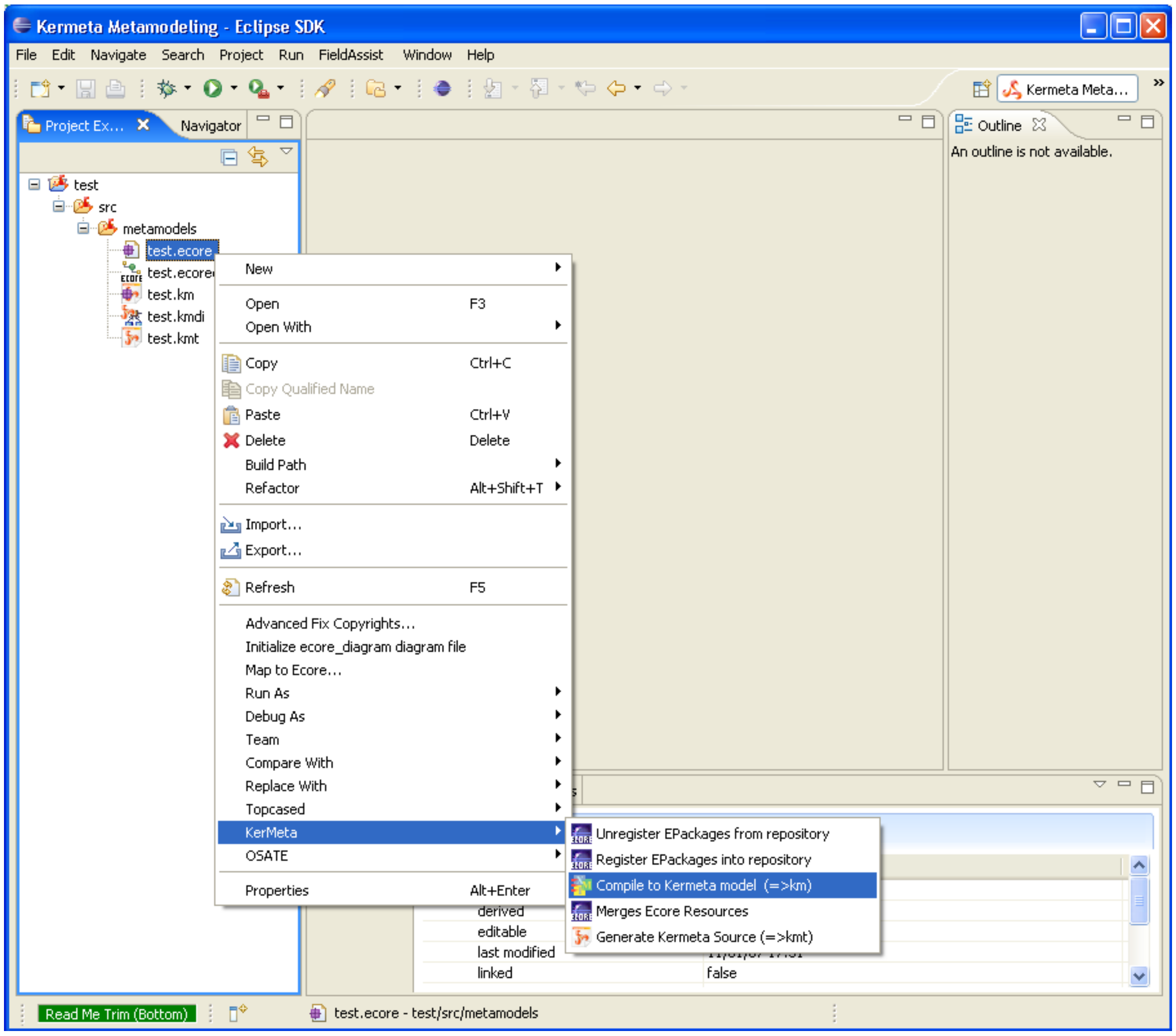


Figure 7.1. Ecore to Km transformation

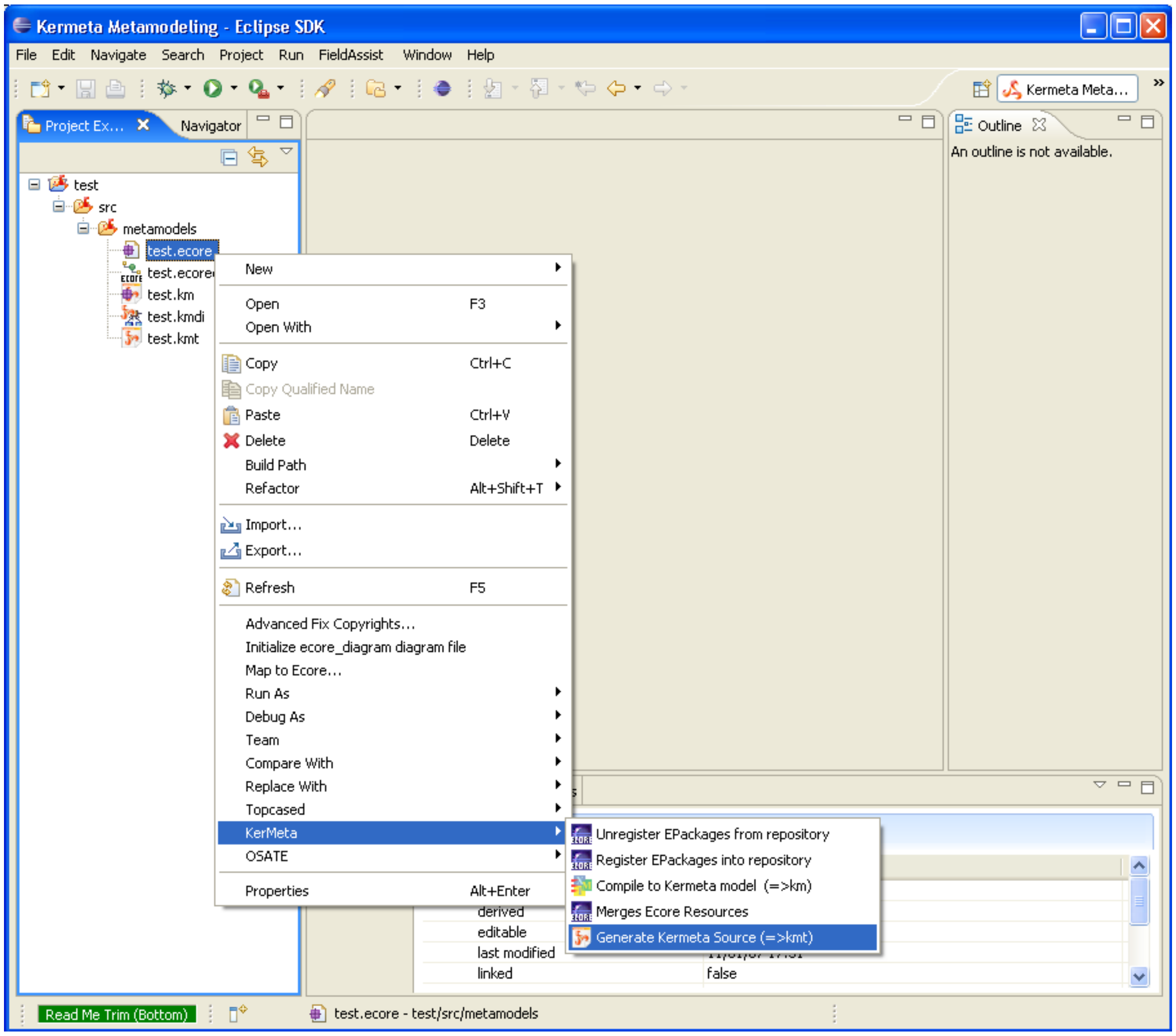


Figure 7.2. Ecore to Kmt transformation

- **Merge function:** You can merge several ecore files, simply select them and click "merge". Its algorithm is base on the names of the packages and classes.
- **Register/Unregister EPackages function:** You can register or unregister in the EMF registry the EPackages defined in an Ecore model, simply select them and right-click "Register" or "Unregister".

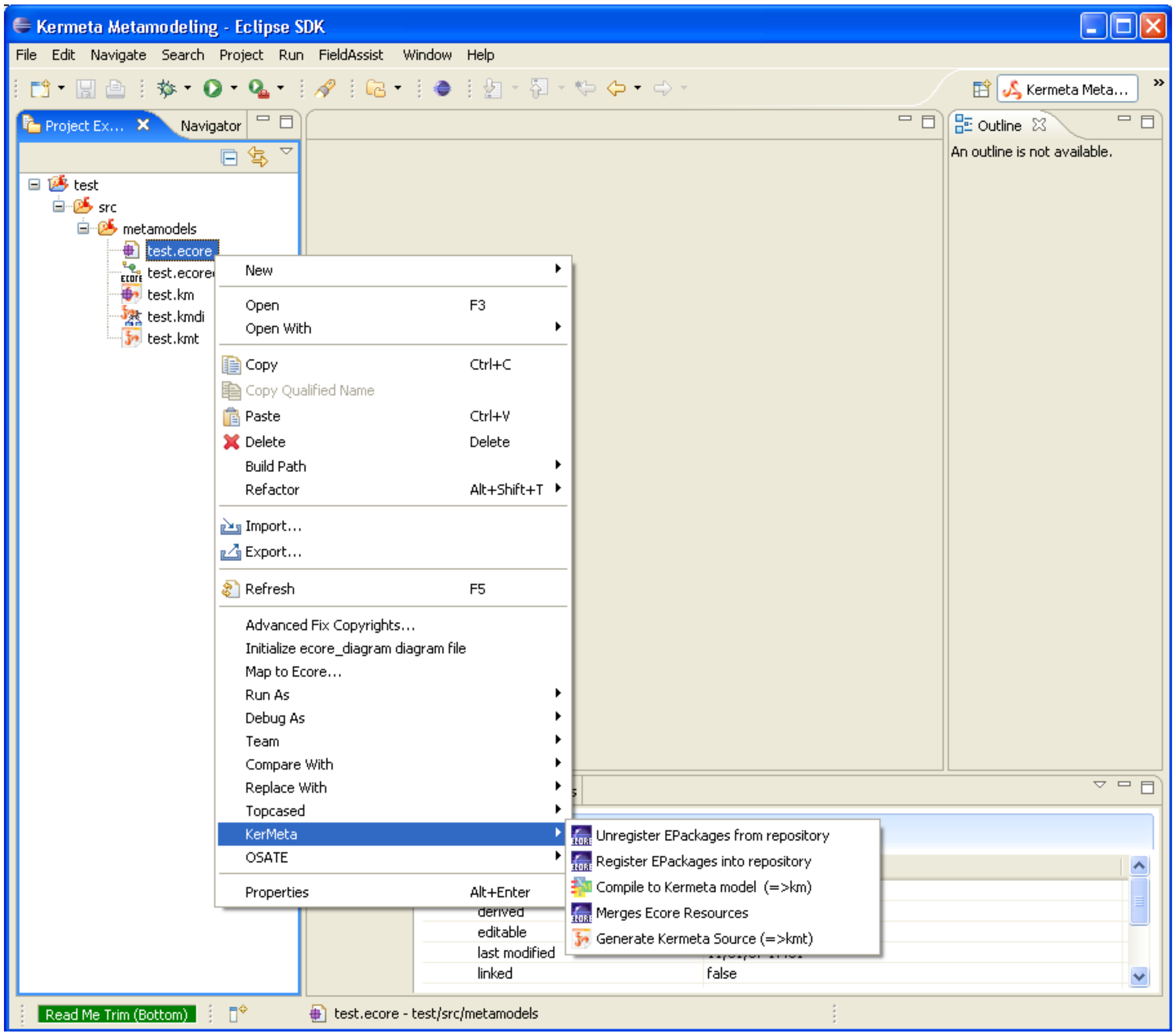


Figure 7.3. Actions on ecore model

7.2. Actions for kmt files

- From a kmt files you can export it to an ecore file. Each of the kermeta notion that don't have a specific representation in ecore are stored into ecore annotations that will allow to roundtrip back to kermeta.

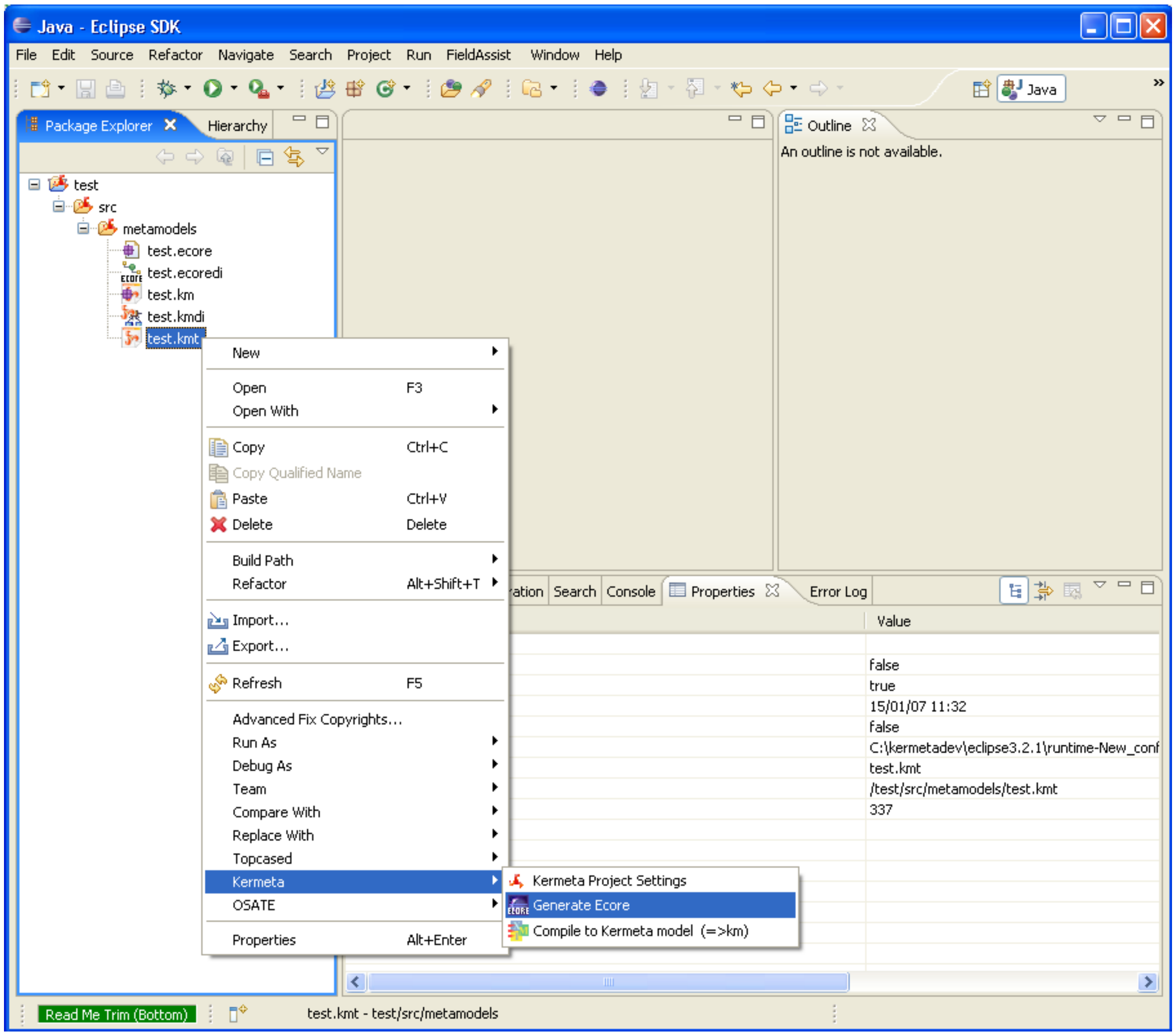


Figure 7.4. Kmt to Ecore transformation

- The kmt file can also be exported into its model version (ie. Into a XMI file conformant to kermeta metamodel). This is useful as this process also typecheck the model and thus allow to save some time when loading other file which rely on the km version rather than the kmt version.

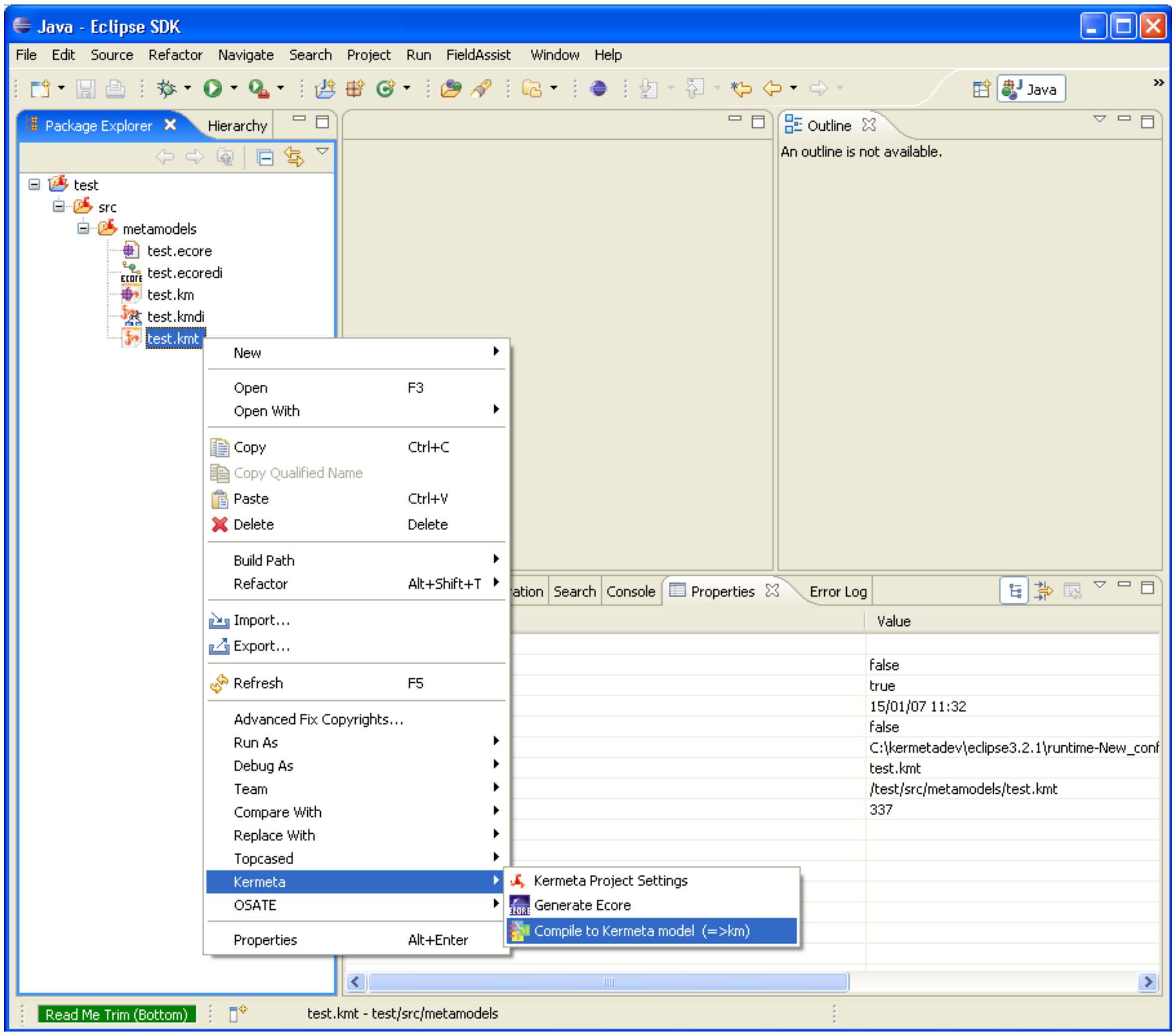


Figure 7.5. Kmt to Km transformation

7.3. Actions for km files

- From a km files you can export it to an ecore file. Each of the kermeta notion that don't have a specific representation in ecore are stored into ecore annotations that will allow to roundtrip back to kermeta.

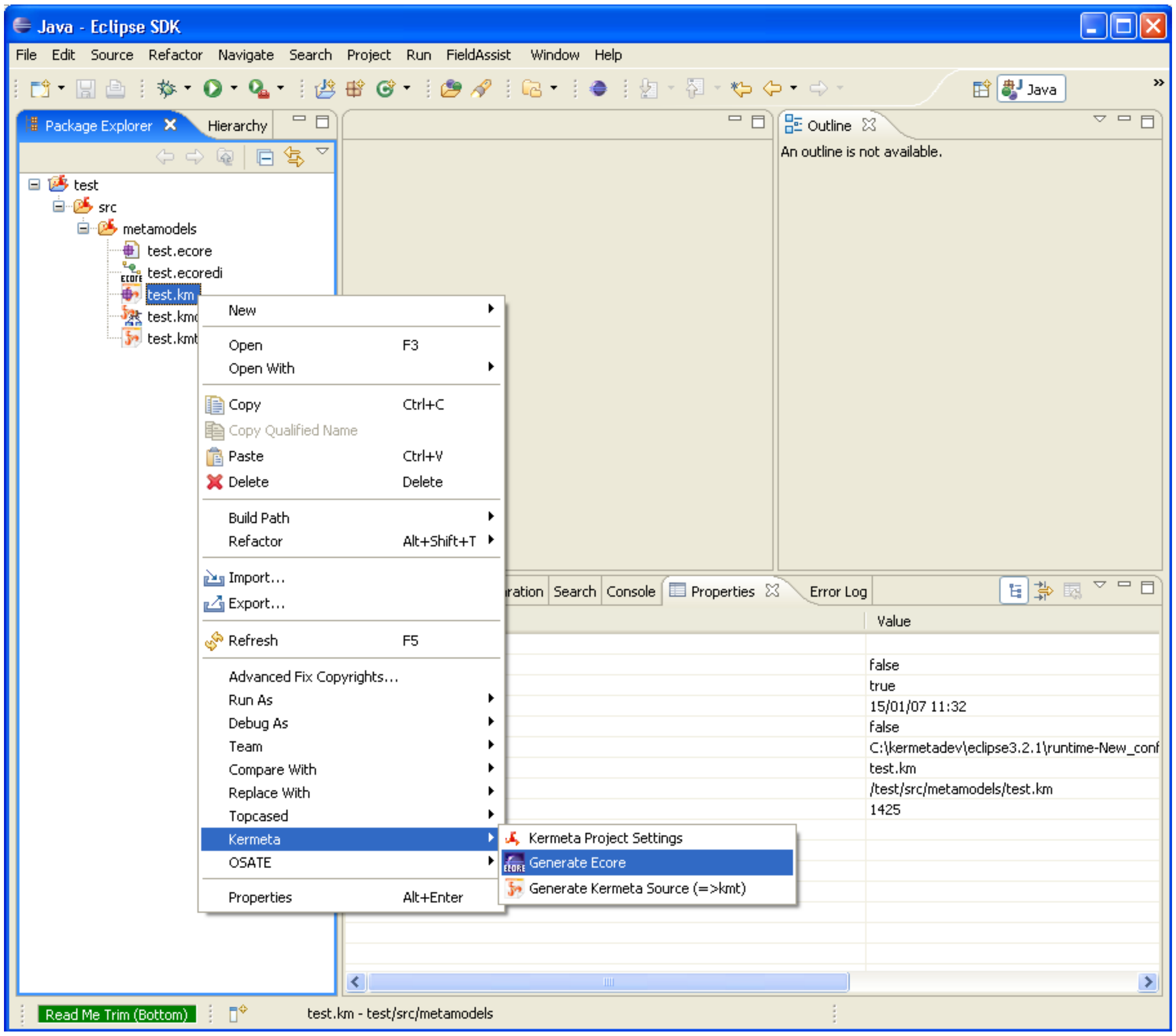


Figure 7.6. Km to Ecore transformation

- You can also prettyprint this model representation of a kermeta program into the textual representation of this program.

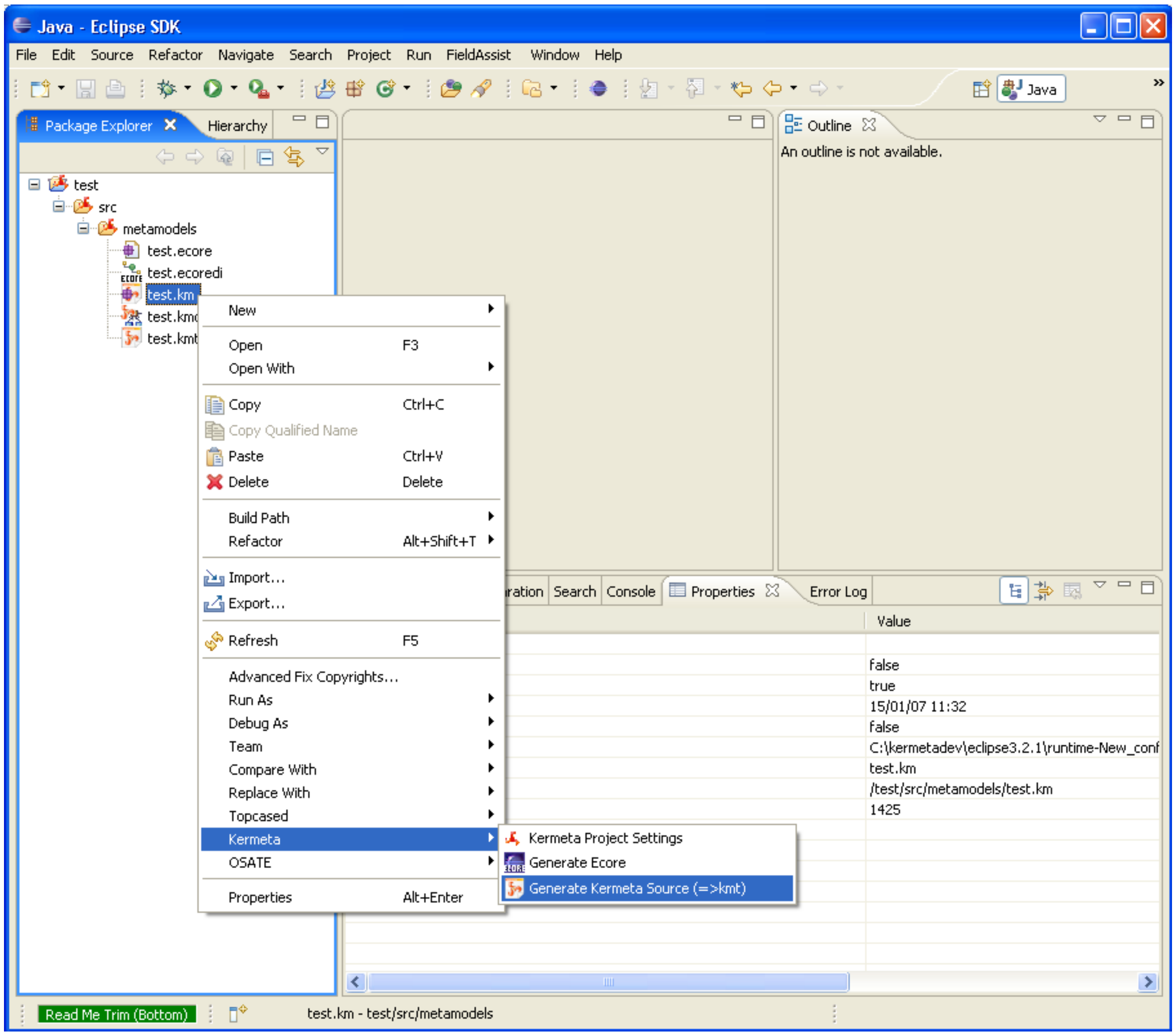


Figure 7.7. Km to Kmt transformation

7.4. Traceability

For many of those transformations, the wizard will propose you to store the traceability informations.

This is a special metamodel that stores a link between the sources elements and the target element so you can reuse it later. For example, it will help the debugger to retrieve the correct line in the text version of the source file even if you are running the km version.

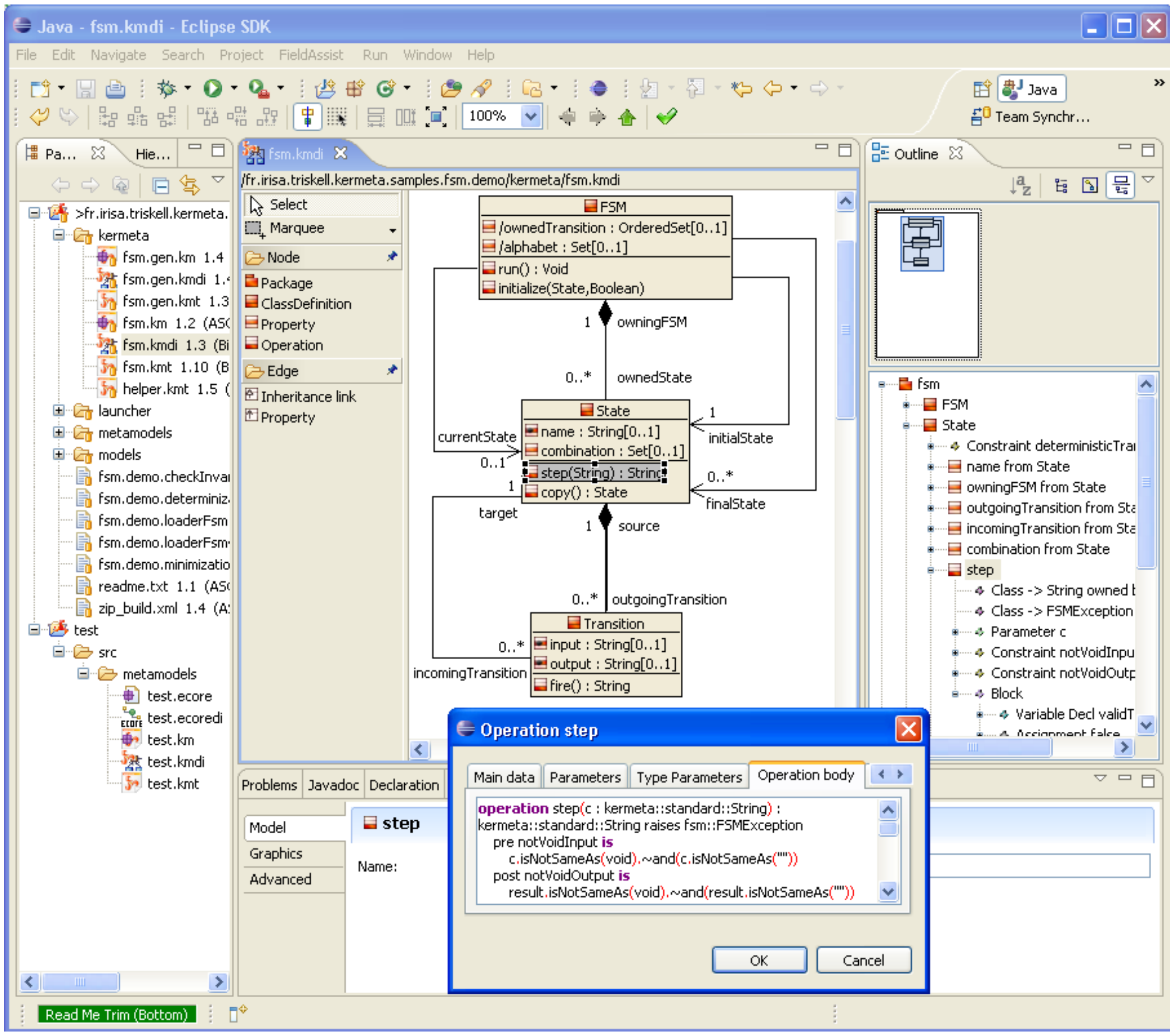
This metamodel is very generic and can be used for any model.

CHAPTER 8

Class diagram Kermeta Model editor

8.1. Overview

The Kermeta Graphical Editor is a view of the Kermeta model as a class diagram. It is generated with the tool "Topcased". The file containing the presentation informations is a *.kmdi. The *.kmdi is associated with a *.km file. The *.kmdi and *.km files must be in the same folder.



```
operation step(c : kermeta::standard::String) :
kermeta::standard::String raises fsm::FSMException
   pre notVoidInput is
      c.isNotSameAs(void).~and(c.isNotSameAs(""))
   post notVoidOutput is
      result.isNotSameAs(void).~and(result.isNotSameAs(""))
```

Figure 8.1. Class diagram Kermeta Model editor - overview

8.2. Opening of a Kermeta class diagram

We can open it, simply select and right-click "Open with / Kermeta class diagram"

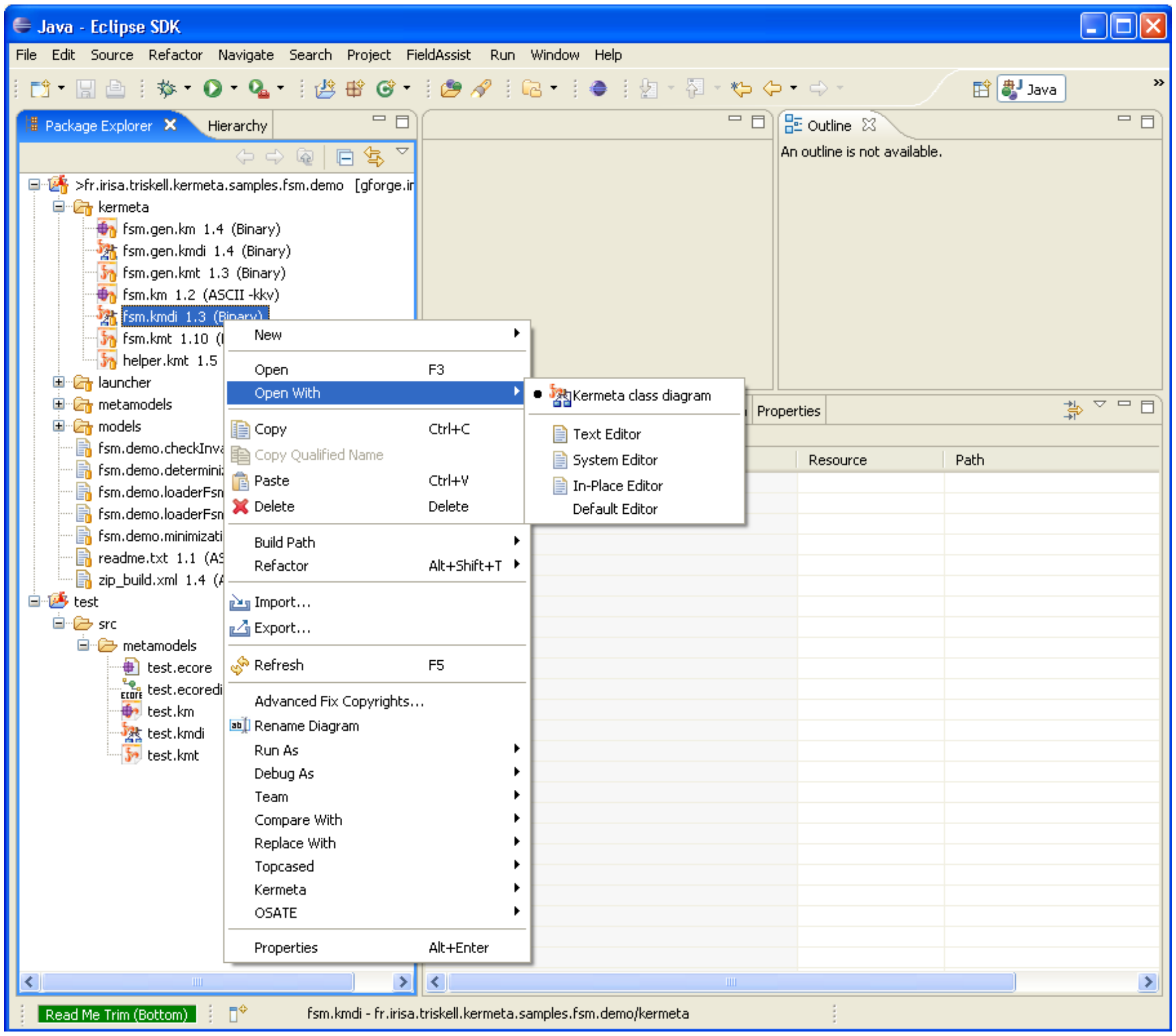
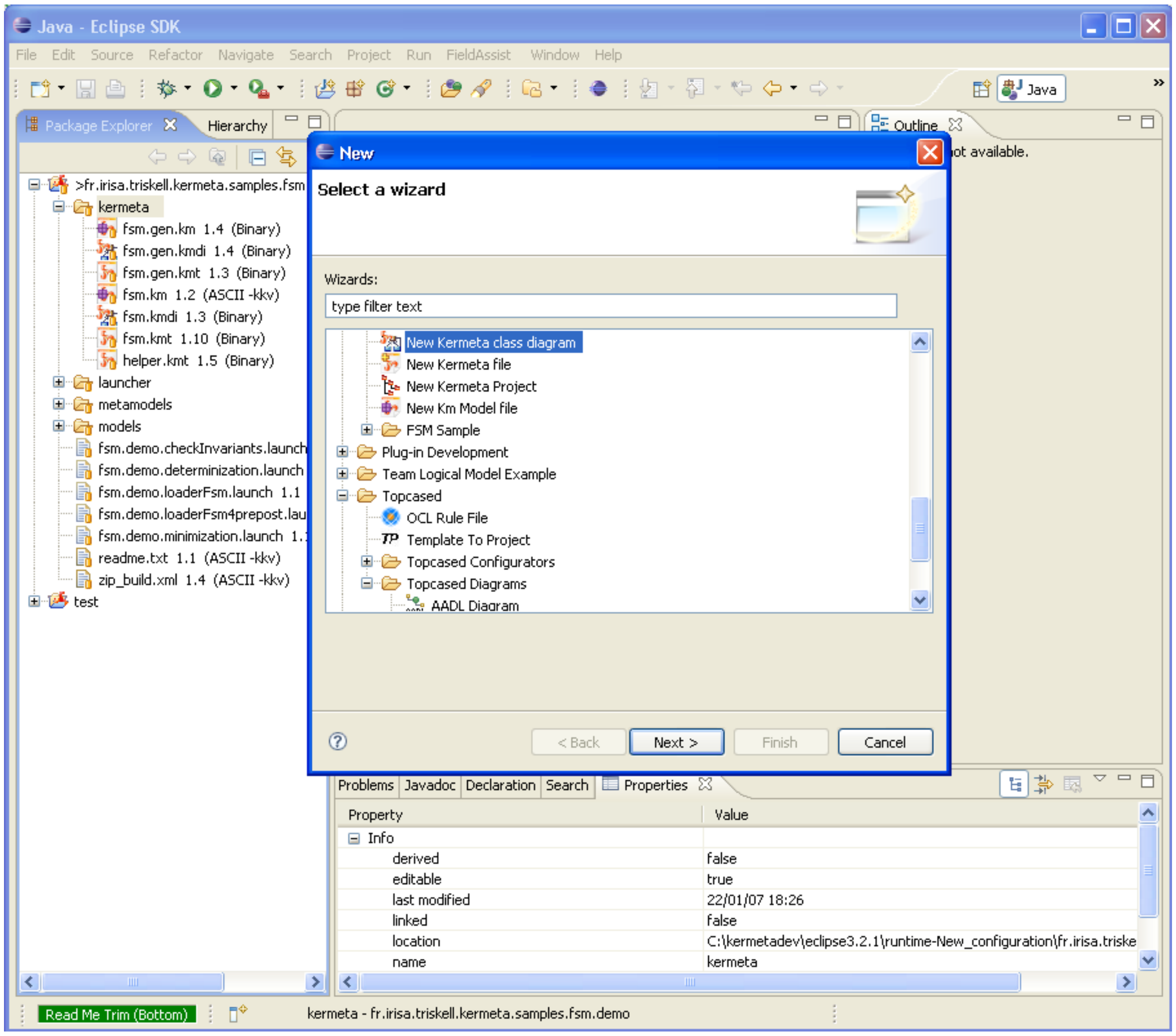


Figure 8.2. Opening of a Kermeta class diagram

8.3. Creating of a Kermeta class diagram

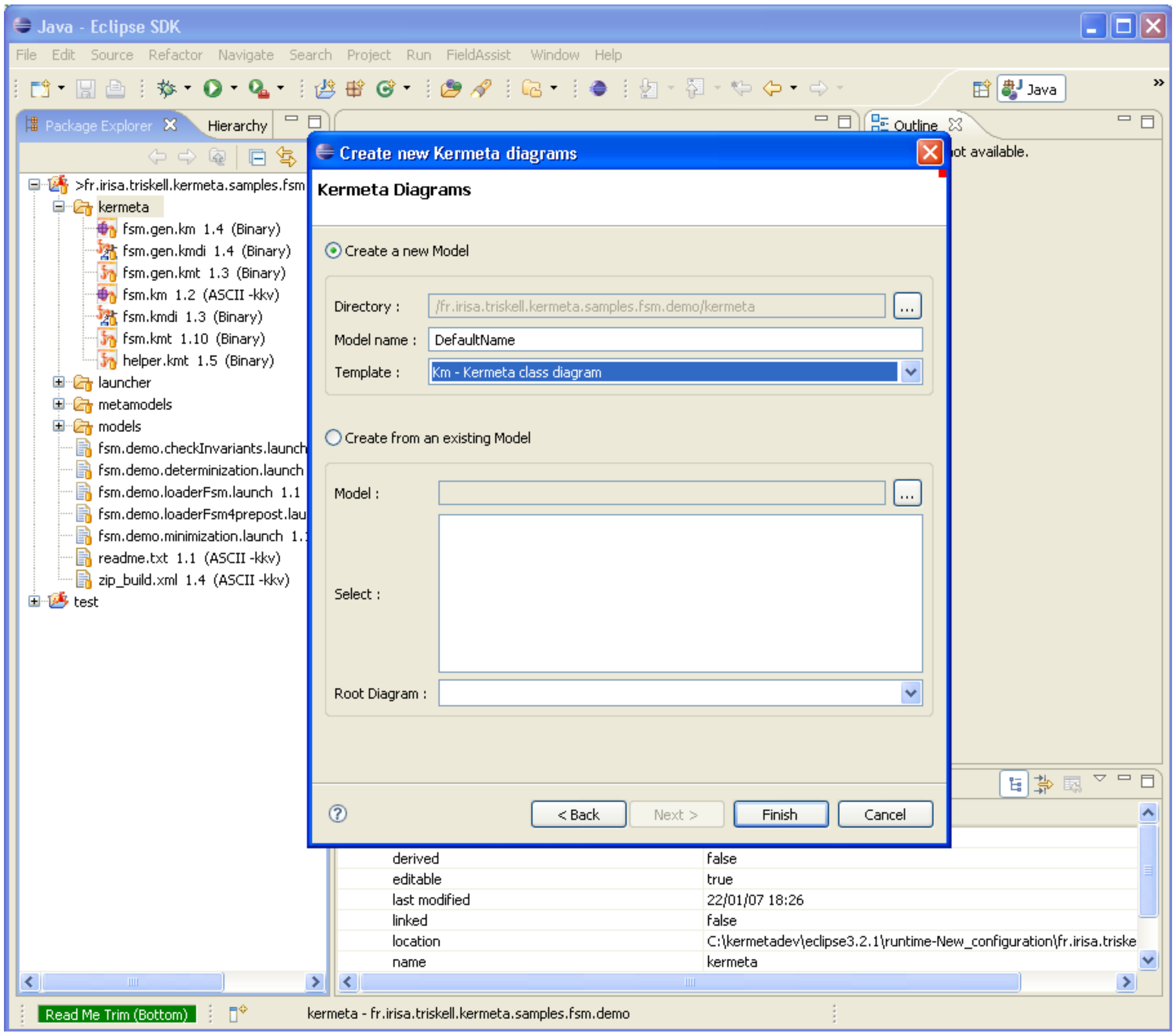
In order to create a Kermeta class diagram you have 2 ways.

- Create a class diagram and the Kermeta model from scratch. Select the project or the folder where you want create the class diagram and right-click "New / Other / Kermeta / New Kermeta class diagram". Change the model name if you want.


Java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run FieldAssist Window Help
Java
Package Explorer
Hierarchy
Outline
not available.
>fr.irisa.triskell.kermeta.samples.fsm
kermeta
fsm.gen.km 1.4 (Binary)
fsm.gen.kmdi 1.4 (Binary)
fsm.gen.kmt 1.3 (Binary)
fsm.km 1.2 (ASCII -kkv)
fsm.kmdi 1.3 (Binary)
fsm.kmt 1.10 (Binary)
helper.kmt 1.5 (Binary)
launcher
metamodels
models
fsm.demo.checkInvariants.launch
fsm.demo.determinization.launch
fsm.demo.loaderFsm.launch 1.1
fsm.demo.loaderFsm4prepost.lau
fsm.demo.minimization.launch 1.1
readme.txt 1.1 (ASCII -kkv)
zip_build.xml 1.4 (ASCII -kkv)
test
New
Select a wizard
Wizards:
type filter text
New Kermeta class diagram
New Kermeta file
New Kermeta Project
New Km Model file
FSM Sample
Plug-in Development
Team Logical Model Example
Topcased
OCL Rule File
Template To Project
Topcased Configurators
Topcased Diagrams
AADL Diagram
< Back
Next >
Finish
Cancel
Problems Javadoc Declaration Search Properties
Read Me Trim (Bottom)
kermeta - fr.irisa.triskell.kermeta.samples.fsm.demo

Property	Value
Info	
derived	false
editable	true
last modified	22/01/07 18:26
linked	false
location	C:\kermetadev\eclipse3.2.1\runtime-New_configuration\fr.irisa.triske
name	kermeta

Figure 8.3. Creating of a Kermeta class diagram from scratch - 1rst step



```
>fr.irisa.triskell.kermeta.samples.fsm
  kermeta
    fsm.gen.km 1.4 (Binary)
    fsm.gen.kmdi 1.4 (Binary)
    fsm.gen.kmt 1.3 (Binary)
    fsm.km 1.2 (ASCII -kkv)
    fsm.kmdi 1.3 (Binary)
    fsm.kmt 1.10 (Binary)
    helper.kmt 1.5 (Binary)
  launcher
  metamodels
  models
  fsm.demo.checkInvariants.launch
  fsm.demo.determinization.launch
  fsm.demo.loaderFsm.launch 1.1
  fsm.demo.loaderFsm4prepost.lau
  fsm.demo.minimization.launch 1.1
  readme.txt 1.1 (ASCII -kkv)
  zip_build.xml 1.4 (ASCII -kkv)
test
```

(●) Create a new Model

Directory : /fr.irisa.triskell.kermeta.samples.fsm.demo/kermeta

Model name : DefaultName

Template : Km - Kermeta class diagram

( ) Create from an existing Model

Model :

Select :

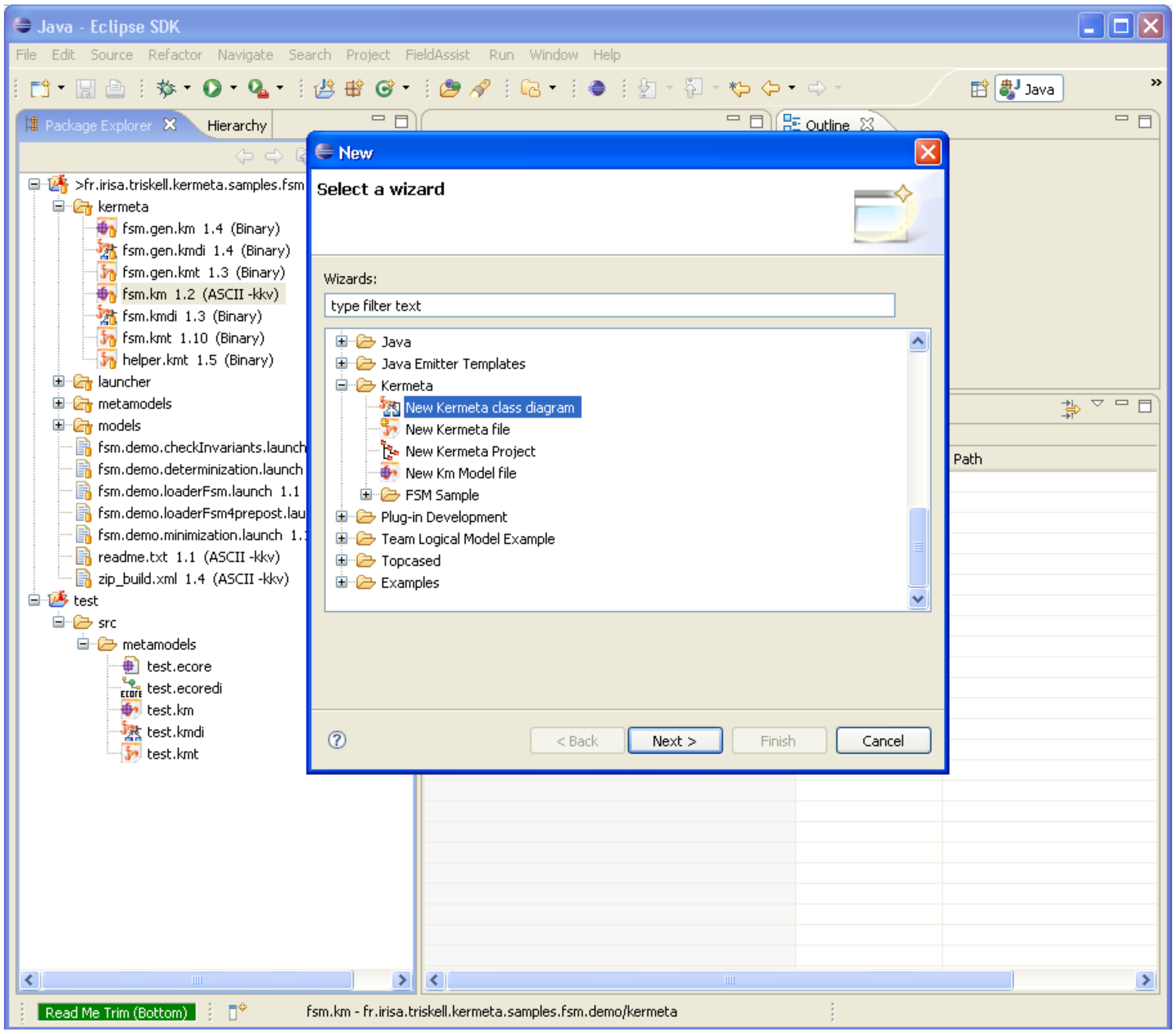
Root Diagram :

derived	false
editable	true
last modified	22/01/07 18:26
linked	false
location	C:\kermetadev\eclipse3.2.1\runtime-New_configuration\fr.irisa.triske
name	kermeta

kermeta - fr.irisa.triskell.kermeta.samples.fsm.demo

Figure 8.4. Creating of a Kermeta class diagram from scratch - 2nd step

- Create a class diagram from an existing Kermeta model. Select the existing km file and right-click "New / Other / Kermeta / New Kermeta class diagram".

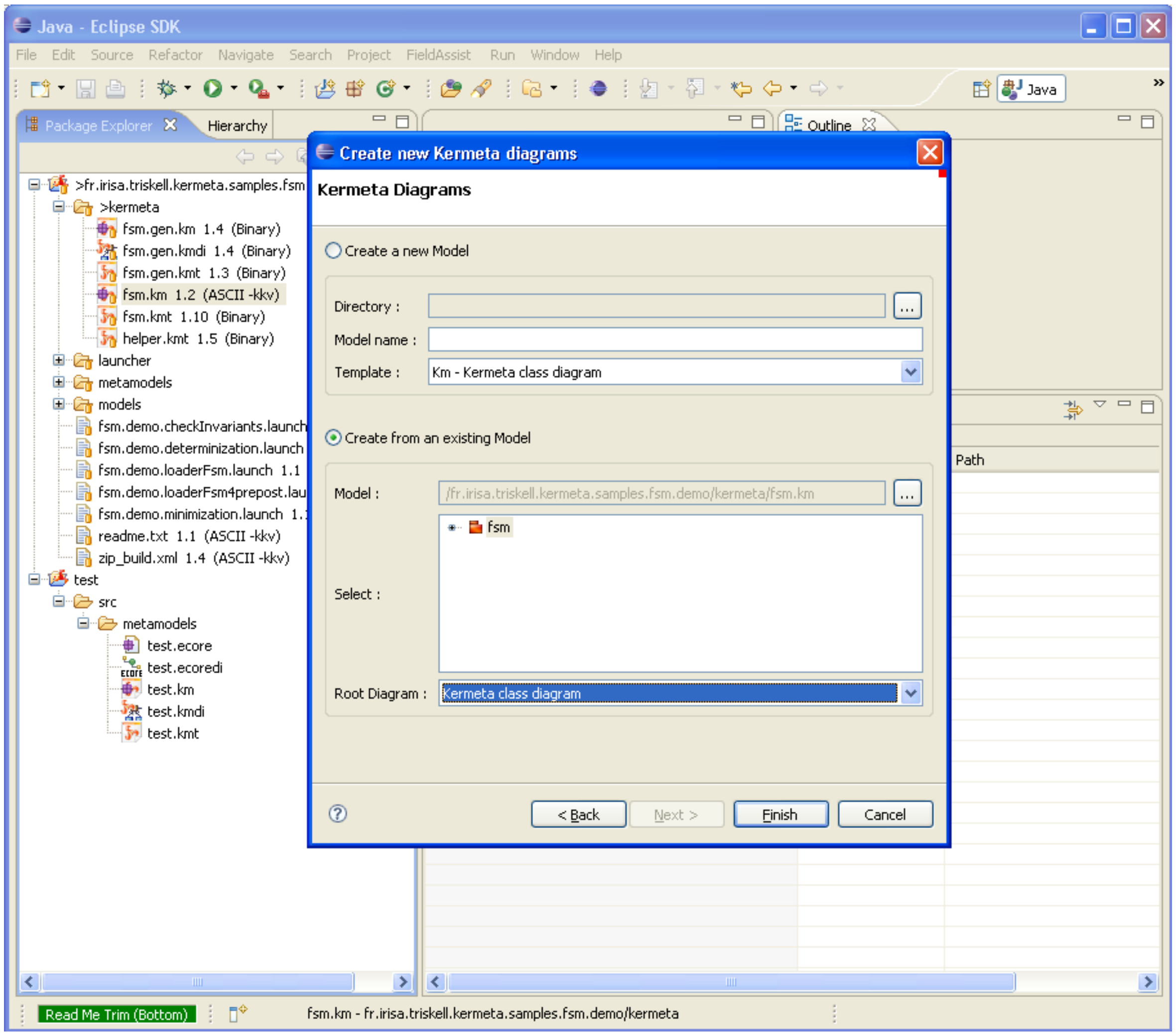


```
>fr.irisa.triskell.kermeta.samples.fsm
  kermeta
    fsm.gen.km 1.4 (Binary)
    fsm.gen.kmdi 1.4 (Binary)
    fsm.gen.kmt 1.3 (Binary)
    fsm.km 1.2 (ASCII -kkv)
    fsm.kmdi 1.3 (Binary)
    fsm.kmt 1.10 (Binary)
    helper.kmt 1.5 (Binary)
  launcher
  metamodels
  models
  fsm.demo.checkInvariants.launch
  fsm.demo.determinization.launch
  fsm.demo.loaderFsm.launch 1.1
  fsm.demo.loaderFsm4prepost.lau
  fsm.demo.minimization.launch 1.1
  readme.txt 1.1 (ASCII -kkv)
  zip_build.xml 1.4 (ASCII -kkv)
test
  src
    metamodels
      test.ecore
      test.ecoredi
      test.km
      test.kmdi
      test.kmt
```

```
Java
Java Emitter Templates
Kermeta
  New Kermeta class diagram
  New Kermeta file
  New Kermeta Project
  New Km Model file
  FSM Sample
Plug-in Development
Team Logical Model Example
Topcased
Examples
```

fsm.km - fr.irisa.triskell.kermeta.samples.fsm.demo/kermeta

Figure 8.5. Creating of a Kermeta class diagram from an existing Kermeta model - 1rst step



```
>fr.irisa.triskell.kermeta.samples.fsm
  >kermeta
    fsm.gen.km 1.4 (Binary)
    fsm.gen.kmdi 1.4 (Binary)
    fsm.gen.kmt 1.3 (Binary)
    fsm.km 1.2 (ASCII -kkv)
    fsm.kmt 1.10 (Binary)
    helper.kmt 1.5 (Binary)
  launcher
  metamodels
  models
  fsm.demo.checkInvariants.launch
  fsm.demo.determinization.launch
  fsm.demo.loaderFsm.launch 1.1
  fsm.demo.loaderFsm4prepost.lau
  fsm.demo.minimization.launch 1.1
  readme.txt 1.1 (ASCII -kkv)
  zip_build.xml 1.4 (ASCII -kkv)
test
  src
    metamodels
      test.ecore
      test.ecoredi
      test.km
      test.kmdi
      test.kmt
```

Create new Kermeta diagrams

Kermeta Diagrams

( ) Create a new Model

Directory :

Model name :

Template : Km - Kermeta class diagram

(●) Create from an existing Model

Model : /fr.irisa.triskell.kermeta.samples.fsm.demo/kermeta/fsm.km

Select : fsm

Root Diagram : Kermeta class diagram

< Back | Next > | Finish | Cancel

fsm.km - fr.irisa.triskell.kermeta.samples.fsm.demo/kermeta

Figure 8.6. Creating of a Kermeta class diagram from an existing Kermeta model - 2nd step

CHAPTER 9

Kermeta Class TouchNavigator

The Kermeta Class TouchNavigator is a view that dynamically focuses on the element you are viewing in the editor or the outline.

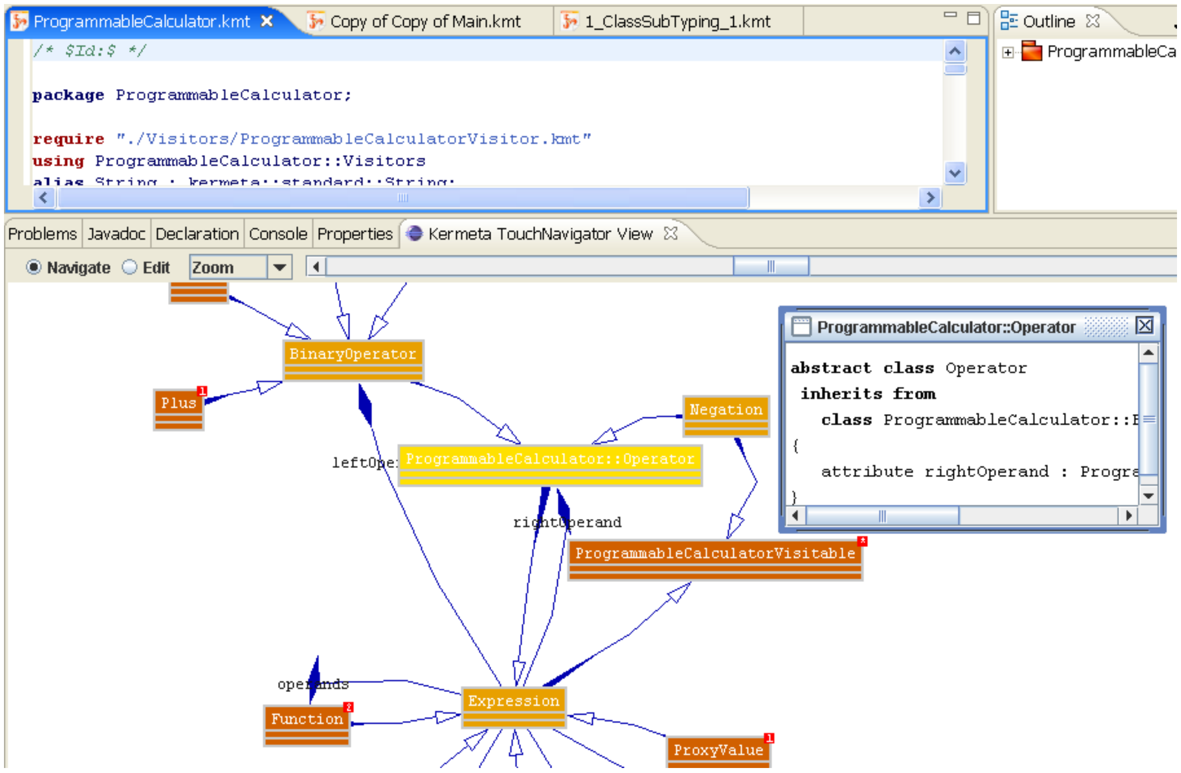
With its damping animation, it proposes an original way to navigate and analyze your metamodel classes.

>The color depends on the distance to the selected node.

Back and Forward button allow to navigate the history of selected nodes.

It provides several lens for smarter visualization. For example you can zoom, rotate, set the locality (ie. How much nodes you display at once depending on the distance to the selection) or use the hyperbolic lens (Ie. Change the edge length depending on the distance).

If you leave your mouse over a class, it shows its documentation.



```
/* $Id:$ */

package ProgrammableCalculator;

require "./Visitors/ProgrammableCalculatorVisitor.kmt"
using ProgrammableCalculator::Visitors
alias String : kermeta::standard::String;
```

```
abstract class Operator
 inherits from
   class ProgrammableCalculator::E
{
   attribute rightOperand : Progre
}
```

Figure 9.1. Kermeta Class TouchNavigator

Tip

It is in alpha stage, feel free to notify us what you found interesting or not in this view.

CHAPTER 10

Ecore Model Editor

This Editor is provided by Eclipse in EMF.

Since Kermeta metamodel is an extension of EMOF, it is possible to use this editor to edit Kermeta programs.

This based on the Ecore2Kermeta and Kermeta2Ecore transformations.

With the Ecore representation of your program, you can use the Ecore Model editor (from Eclipse\EMF) or any other Ecore tool (for example Omondo.graphic editor)

This is described with more details in the EMF tutorial: **ht-tp://www.kermeta.org/docs/html.single/KermetaEMFTutorial/**

Chapter 11

Tip and tricks

Tip

This section contains many useful information about how to use the Kermeta workbench.

11.1. Texteditor keyboard shortcuts

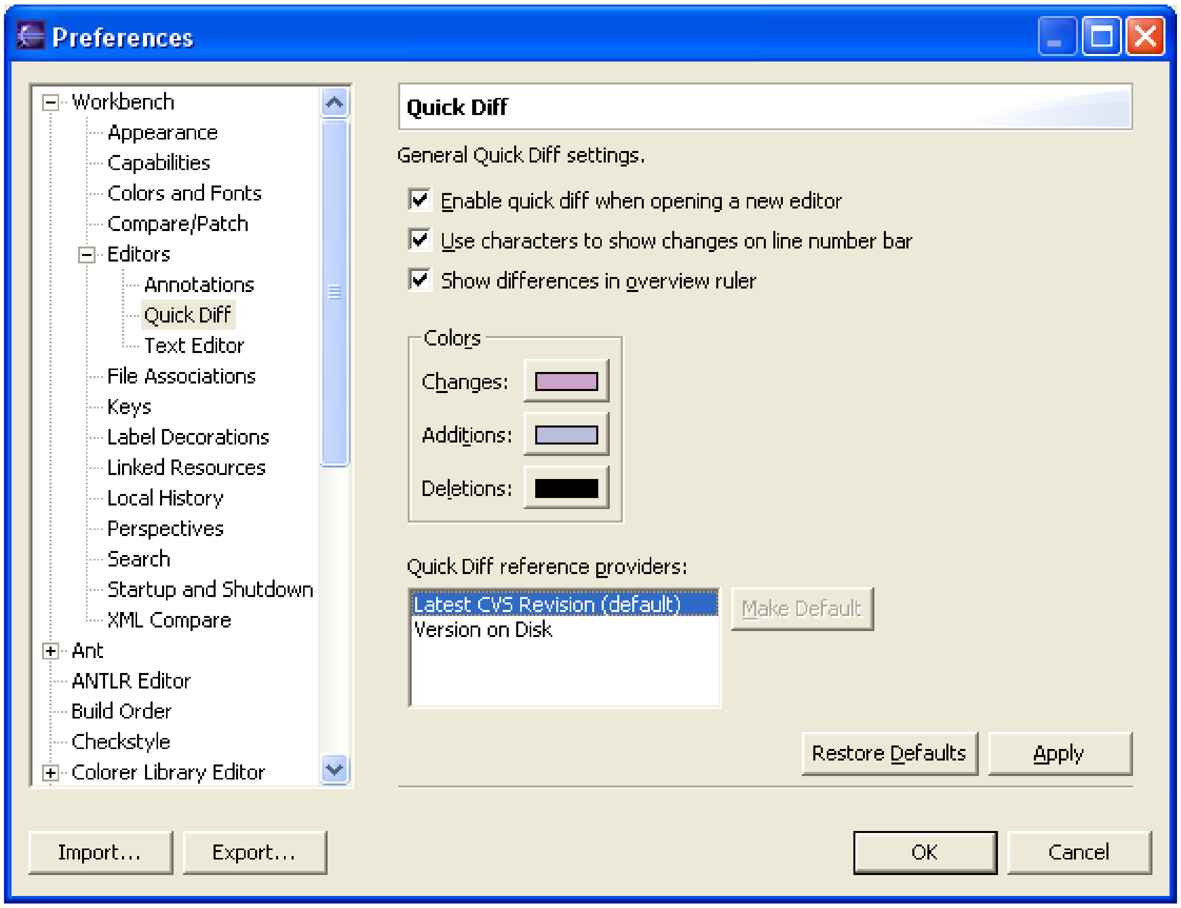
Keyboard shortcut	Action
<CTRL>+<SPACE>	Pertinent code completion. This work in several places in the code.

[TODO check with the platform tip and tricks page which eclipse shortcut correctly work with Kermeta edit-or]

11.2. Texteditor tips

11.2.1. Quickdiff

When working with CVS, the quickdiff is best used if it use the latest CVS revision as reference.



Preferences

- Workbench
  - Appearance
  - Capabilities
  - Colors and Fonts
  - Compare/Patch
  - Editors
    - Annotations
    - Quick Diff
    - Text Editor
  - File Associations
  - Keys
  - Label Decorations
  - Linked Resources
  - Local History
  - Perspectives
  - Search
  - Startup and Shutdown
  - XML Compare
- Ant
- ANTLR Editor
- Build Order
- Checkstyle
- Colorer Library Editor

Quick Diff

General Quick Diff settings.

[✓] Enable quick diff when opening a new editor

[✓] Use characters to show changes on line number bar

[✓] Show differences in overview ruler

Colors

Changes: ■

Additions: ■

Deletions: ■

Quick Diff reference providers:

Latest CVS Revision (default)

Version on Disk

Figure 11.1. Quickdiff recommended preferences