GEMOC Studio Guide

Francois Tanguy <francois.tanguy@inria.fr>,
Didier Vojtisek <didier.vojtisek@inria.fr>

GEMOC Studio Guide

Francois Tanguy <francois.tanguy@inria.fr>, Didier Vojtisek <didier.vojtisek@inria.fr>

Table of Contents

List of Figures

Introduction

The GEMOC Studio offers 2 main usages:

- Building and composing new executable DSML. This mode is intended to be used by language designers (aka domain experts).
- Creating and executing models conformant to executable DSMLs. This mode is intended to be used by domain designers.

Each of these usage has it own set of tools that are referenced as **Gemoc Language workbench** for the tools for *language designers* and **Gemoc Modeling Workbench** for the tools for *domain designers*.

Chapter 1. Gemoc Language workbench

1.1. Language Workbench overview

The GEMOC Language Workbench, intended to be used by language designers: it allows building and composing new executable DSMLs.

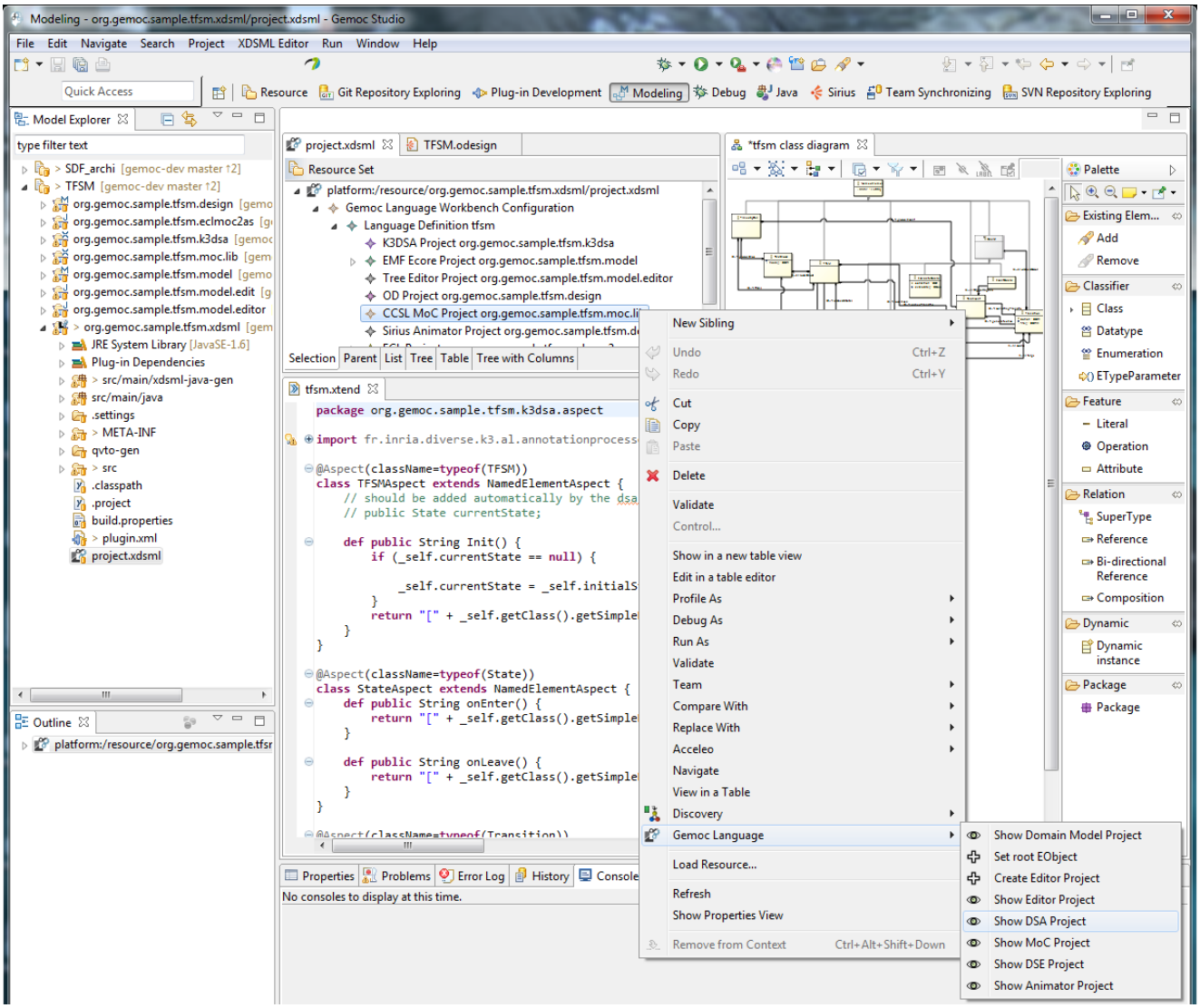

Figure 1.1. Screenshot of GEMOC Studio Language Workbench on the TFSM (Timed Finite State Machine) example.

1.2. Gemoc Language project

In the menu go to: File> New> Projects…> Gemoc Language project

This project will be the aggregator of the other Eclipse projects that will constitute the components of a Language Unit of the xDSML.

It mainly defines a project.xdsml file that references these other components.

The xDSML file isn't supposed to be edited manually. The pop up menu on the project (available on right click) or the Section 1.11, "Process support view" provides a set of wizard to fill this file.

From these information, the project will generate additional code that will be used by the execution engine when the language will be deployed and used by the Chapter 2, *Gemoc Modeling workbench*.

1.3. Defining Abstract Syntax (or domain concepts)

Defining the abstract syntax (AS) in the Language Workbench is done thanks to an EMF Ecore project that will be associated to the xDSML definition.

All ecore editors can be used to edit the concepts in the ecore file.

The EMF project can be created manually and then associated to the xDSML or directly created for the specific use of a given xDSML.

1.3.1. Method 1 : Creation via xDSML popup

To create the EMF project:

- right click on the xDSML project
- navigate in the *Gemoc Language* entry.
- click on *create Domain Model project*

This will open a wizard asking either to create a new EMF project or select an existing one (usefull for method 2). Creating a new EMF project will ask for some

> **Note**
>
> In future version of the studio, this method may disapear and be replaced by method 3.

1.3.2. Method 2 : Manual creation and association to the xDSML

From eclipse menu:

- *File > New > Projects… > Ecore Modeling Project*
- in the Ecore diagram, define the concepts for your language as a class diagram.

> **Tip**
>
> There are several other ways to create valid EMF projects. Amung them we can consider: *File > New > Projects… > EMF Project*, *File > New > Projects… > Empty EMF Project*, *File > New > Projects… > xText Project* … However, this isn't in the scope of this document to detail them or explain how to correctly configure them for being used by Gemoc.

Then associate the EMF project to the xDSML. The association is done either with right click on the xDSML project, *Gemoc Language > create Domain Model project > Select existing EMF project*; or via the Gemo Guideline view.

1.3.3. Method 3 : Creation or selection via Gemoc Guideline view

The Gemoc Guideline view offers an integrated vision of the most important steps of the xDSML creation process. When selecting the *Define Domain Model* task, a button allows to launch the wizards of method 1.

TODO new project wizard (via popup, via process) TODO select existing project TODO Ecore editor

1.4. Defining RunTime Data

1.5. Defining Domain-Specific Actions (DSAs)

In Gemoc, the DSAs are developped using Kermeta 3.

> **Note**
>
> There may exist other ways to create DSAs, but these methods aren't in the scope of this document.

- right click on the xDSML project
- navigate in the *Gemoc Language* entry.
- click on *create DSA project*

In the wizard you can select an existing templates to help create the aspects classes that will extend the classes of the Domain model.

You can manually create a Kermeta 3 project in the main eclipse menu, and then associate it to an xDSML project.

1.6. Defining Domain-Specific Constraints

1.7. Defining a concrete syntax

The xDSML can support different concrete syntaxes. Most EMF based editors should work however Gemoc provides additional support for some editors. Editors explictly supported are: EMF tree editor, xText editor, Sirius editor.

1.7.1. Defining a concrete syntax with xText

1.7.2. Defining a concrete syntax with Sirius

1.8. Defining a Model of Concurrency (MoC)

1.9. Defining Domain Specific Events (DSE)

1.10. Defining an animation view

The animation layer is an extension on top of a graphical editor defined with Sirius.

TODO Debug layer, Animation layer

1.11. Process support view

TODO present process view

Chapter 2. Gemoc Modeling workbench

2.1. Modeling workbench overview

The GEMOC Modeling Workbench, intended to be used by domain designers: it allows creating and executing models conformant to executable DSMLs.

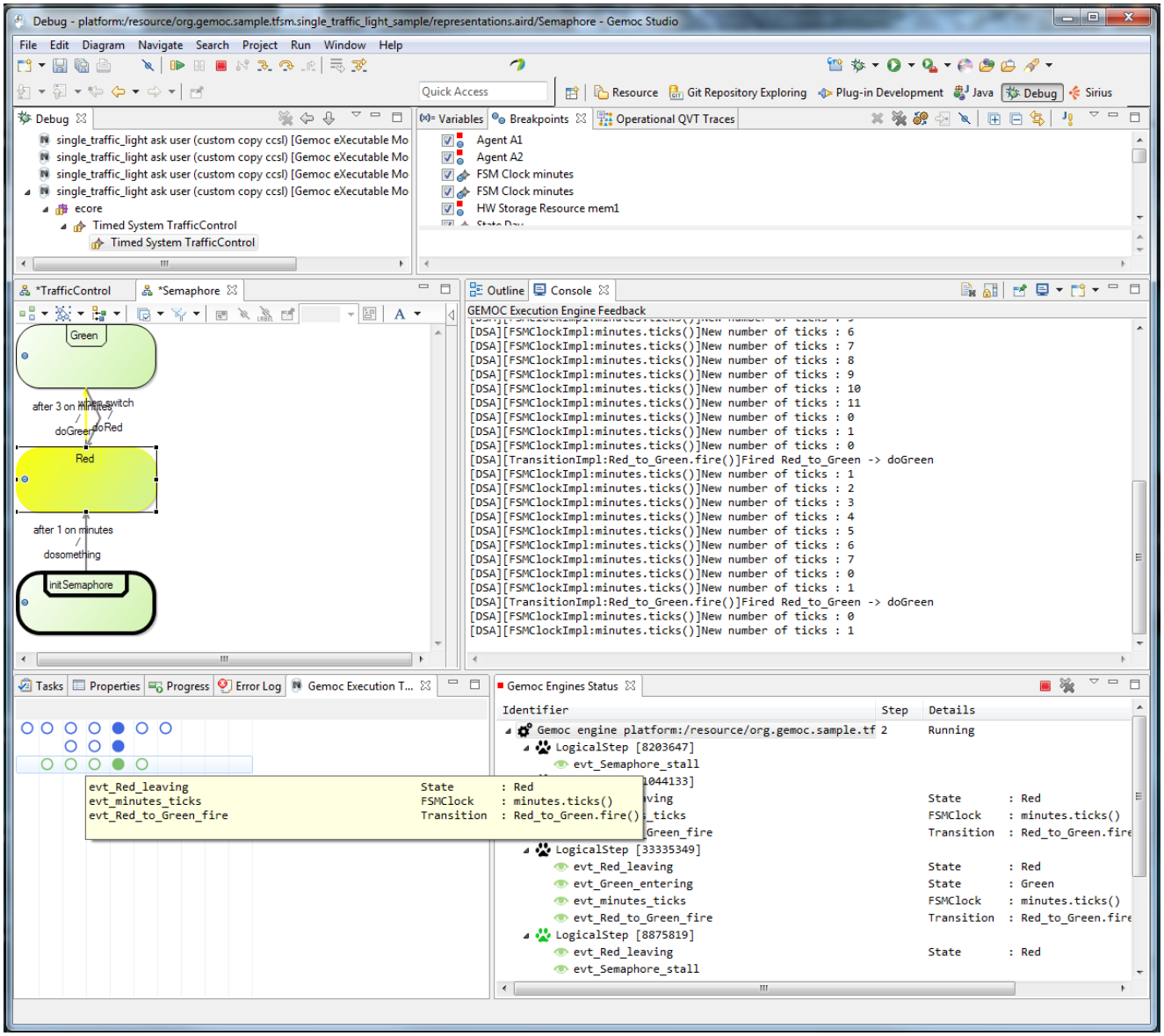

Figure 2.1. Screenshot of GEMOC Studio Modeling Workbench on the TFSM example (execution and animation).

2.2. Editing model

2.3. Executing model

2.3.1. Launch configuration

The Gemoc launch configuration offers both a Run and a Debug mode.

General options

- Model to execute : this is the model that will be run
- xDSML : this field allows to select among the valid variants of the executable language that are available for the model (Ie. the combinaison of DSA, DSE and MoCC that can be used on the given domain model)
- Decider : this field allows to select the solver strategy used by the engine when several Logical Steps can be triggered. Possible choice are :
  - Solver proposition : the solver internal stategy will be used to selecton Logical Step
  - Random : will randomly select one of the available Logical Step (warning: execution cannot be reproduced when using this Decider)
  - Ask user : (available only in Debug mode), this option will use the Logical Step View or the Timeline View to present the available Logical Steps and pause if there are more than one Logical Step. The user will then need to click on one of the Logical Step to continue.
  - Ask user (Step by step) : (available only in Debug mode), this option is similar to the previous one. However, it will pause on every Logical Step, even if there is only one Logical Step that can be triggered. This is more or less equivalent as putting a breakpoint on every MSE of the language.

More Deciders will be developped (for example for playing predefined scenario).

Run mode

In run mode, it offers the faster way to run the model. It cannot be paused. However, you can stop it. It offers a limited set of views :

- the Engine View allows to stop a running model.
- the Timeline View is displayed at the end of the execution in order to control the resulting execution trace.

If more feed back are required, please use one of the front end or back end available for the xDSML.

Debug mode

In debug mode, the engine offers more control on the execution. It allows to pause, add break point, and run in a step by step mode.

It reuses the Eclipse Debug perspective and some of its views and add some Gemoc specific views.

- the Engine View allows to stop a running model.
- the Timeline View is displayed during all the simulation.
- the Event Manager View is displayed during all the simulation.
- the Event Manager View is displayed during all the simulation. It can display both an animation layer and a debug layer.
- the Debug View. This view presents an interface for Step by Step execution at the Logical Step level or even at the DSA level.
- the Variable View. This view presents the Runtime Data as a (EMF based) tree.

When running a simulation in Debug mode, it is configured to activate automatically the Debug layer and the Animation layer in the Animation view.

Backends and frontends

Back ends and front ends offer additionnal view that can respectivly display informations from the running model or provide event input to the running model.

These backends and front ends usually open dedicated views. These views are always opened in all modes (Run or Debug).

2.3.2. Engine View

The engine view displays a list of execution engine and their statuses:

- its number of execution steps,
- its current running status,
- and its logical steps deciding strategy.

2.3.3. Logical Steps View

The logical steps view displays the list of possible future execution. This list is provided by the solver. This view is organized around a tree. For each logical step, its underlying events can be seen and possibly for each event the associated operation is visible.

> **Note**
>
> This view displays nothing when execution runs in "run mode", per say this view is only of use when running in "debug mode".

2.3.4. Timeline View

This view represents the line of the model's execution. It displays:

- the different logical steps proposed by the solver in the past in blue color,
- the selected logical steps at each execution step in green color,
- and the possible future logical steps in yellow color,
- the model specific events for each logical step.

> **Note**
>
> The possible future logical steps are shown under the condition that the model is executing.

In addition to displaying information, it also provides interaction with the user. During execution, it is possible to come back into the past by double-clicking on any of the blue logical steps. It does two things:

1. it resets the solver's state to the selected execution step,
2. and it resets the model's state to the selected execution step.

2.3.5. Event Manager View

2.3.6. Animation View

- Debug Layer

- Animation Layer

2.3.7. Debug View

This view is part of the Debug perspective. It presents an interface for Step by Step execution at the Logical Step level or even at the DSA level. When an execution is paused, this view presents the current Logical Step.

When paused on a Logical Step, the Step over command allows to go to the next Logical Step. The Step Into command allows to run separatly each of the internal DSA calls associated to the Logical Step.

2.3.8. Variable View

This view is available on the Debug perspective. When an execution is paused, this view presents the current Runtime Data as an EMF based tree.

Bibliography

The bibliography lists some useful external documents. For a more complete list, please refer to the publications section on http://gemoc.org site.

Articles

[globalizing-modeling-languages] Globalizing Modeling Languages [http://hal.inria.fr/hal-00994551] (Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert France, Jean-Marc Jezequel, Jeff Gray), In Computer, IEEE, 2014.

Glossary

AS
: Abstract Syntax.

API
: Application Programming Interface.

Behavioral Semantics
: see Execution semantics.

CCSL
: Clock-Constraint Specification Language.

Domain Engineer
: user of the Modeling Workbench.

DSA
: Domain-Specific Action.

DSE
: Domain-Specific Event.

DSML
: Domain-Specific (Modeling) Language.

Dynamic Semantics
: see Execution semantics.

Eclipse Plugin
: an Eclipse plugin is a Java project with associated metadata that can be bundled and deployed as a contribution to an Eclipse-based IDE.

ED
: Execution Data.

Execution Semantics
: Defines when and how elements of a language will produce a model behavior.

GEMOC Studio
: Eclipse-based studio integrating both a language workbench and the corresponding modeling workbenches

Language Workbench
: a language workbench offers the facilities for designing and implementing modeling languages.

Language Designer
: a language designer is the user of the language workbench.

MoCC
: Model of Concurrency and Communication

Model
: model which contributes to the convent of a View

Modeling Workbench
: a modeling workbench offers all the required facilities for editing and animating domain specific models according to a given modeling language.

MSA
: Model-Specific Action.

MSE
: Model-Specific Event.

RTD
: RunTime Data.

Static semantics
: Constraints on a model that cannot be expressed in the metamodel. For example, static semantics can be expressed as OCL invariants.

xDSML
: Executable Domain-Specific Modeling Language.

Index