Grant ANR-12-INSE-0011

ANR INS GEMOC

D4.1.1 - Gemoc Architectural description
Task 4.1.1

Version 1.1

DOCUMENT CONTROL

	–: 03/06/2013	A: 10/09/2013	B:	C:	D:
Written by Signature	Didier Vojtisek	Vincent Fontanella			
Approved by Signature	Benoit Combemale				

Revision index	Modifications
–	Initial version
A	Update of the global architecture
B	
C	
D	

Authors

Author	Partner	Role
Didier Vojtisek	INRIA	Lead author
Mélanie Bats	OBEO	Author
Vincent Fontanella	INRIA	Author
All	All	Contributor

Contents

1. Introduction

1.1 Purpose

This document presents the architectural design of the GEMOC Studio.
We describe the various components and metamodels integrated into the studio.
A website is available to have more informations and follow the `GEMOC Initiative` : `www.gemoc.org`
A continuous integration is available at the link below :
`https://ci.inria.fr/gemoc`

1.2 Perimeter

1.3 Definitions, Acronyms and Abbreviations

- **Architecture** : fundamental conception of a system in its environment embodied in its elements, their relationships between each other and to its environment, and the principles guiding its design and evolution
- **Architectural Description** : collection of information items used to describe an architecture
- **Architecture Framework** : set of common practices for ArchitecturalDescription established within a specific domain or stakeholder community
- **Bundle** : a group of Java classes and additional resources equipped with a detailed manifest defining the entire component, its provided and required classes
- **Model** : model which contributes to the convent of a View
- **Update-Site** : an update-site is an aggregation of deployment units which might be used to provision a target platform
- **Stakeholder** : (of a system) individual, team, organization, or classes thereof, having concerns with respect to a system
- **Language workbench** : a language workbench offers the facilities for designing and implementing modeling languages.
- **Language designer** : a language designer is the user of the langage workbench.
- **Modeling workbench** : a modeling workbench offers all the required facilities for editing and animating domain specific models according to a given modeling language.
- **Domain engineer** : user of the modeling workbench.
- **GEMOC Studio** : Eclipse-based studio integrating both the language workbench and the modeling workbench.
- **DSML** : Domain Specific Modeling Language
- **xDSML** : Executable Domain Specific Modeling Language
- **AS** : Abstract Syntax
- **MOCC** : Model of Computation and Communication

- **DSA** : Domain Specific Action
- **DSE** : Domain Specific Event

1.4 References

The basis of the GEMOC Studio architecture is described in the project proposal document.

1.5 Summary

This document describes the architecture of the GEMOC Studio, outlining every logical component along with its role and how it relates to the other components. We will detail all required and provided interfaces and define the exchanged data's format, thus describing how each individual piece of the studio is going to communicate with the others. This document also lists the requirements regarding the technical integration of these components and guides the GEMOC partners in providing their artifacts.

2. Architecture

Starting from the project prososal view of the Studio (see figure 2.1), it is refined into a component based architecture as shown in figure 2.2.

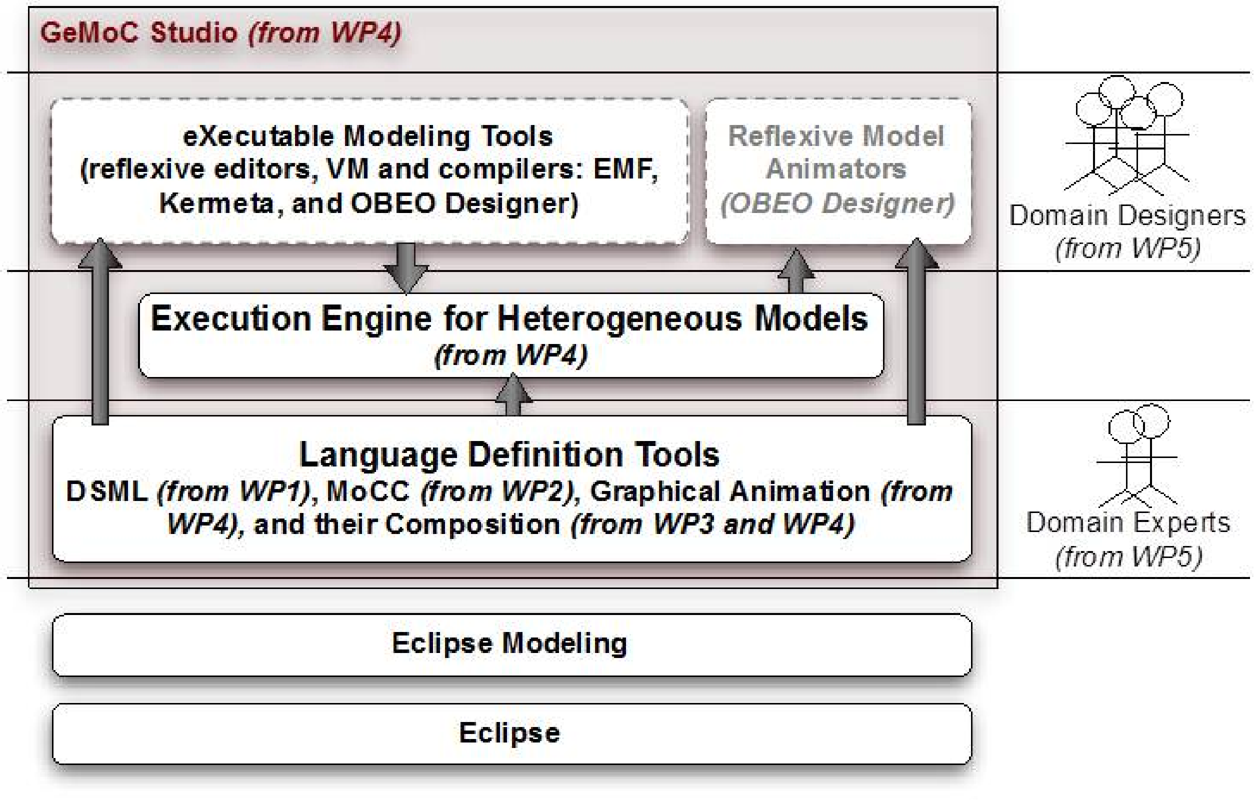

GeMoC Studio (from WP4)
eXecutable Modeling Tools (reflexive editors, VM and compilers: EMF, Kermeta, and OBEO Designer)
Reflexive Model Animators (OBEO Designer)
Domain Designers (from WP5)
Execution Engine for Heterogeneous Models (from WP4)
Language Definition Tools
DSML (from WP1), MoCC (from WP2), Graphical Animation (from WP4), and their Composition (from WP3 and WP4)
Domain Experts (from WP5)
Eclipse Modeling
Eclipse

Figure 2.1: Project proposal view of the GEMOC Studio

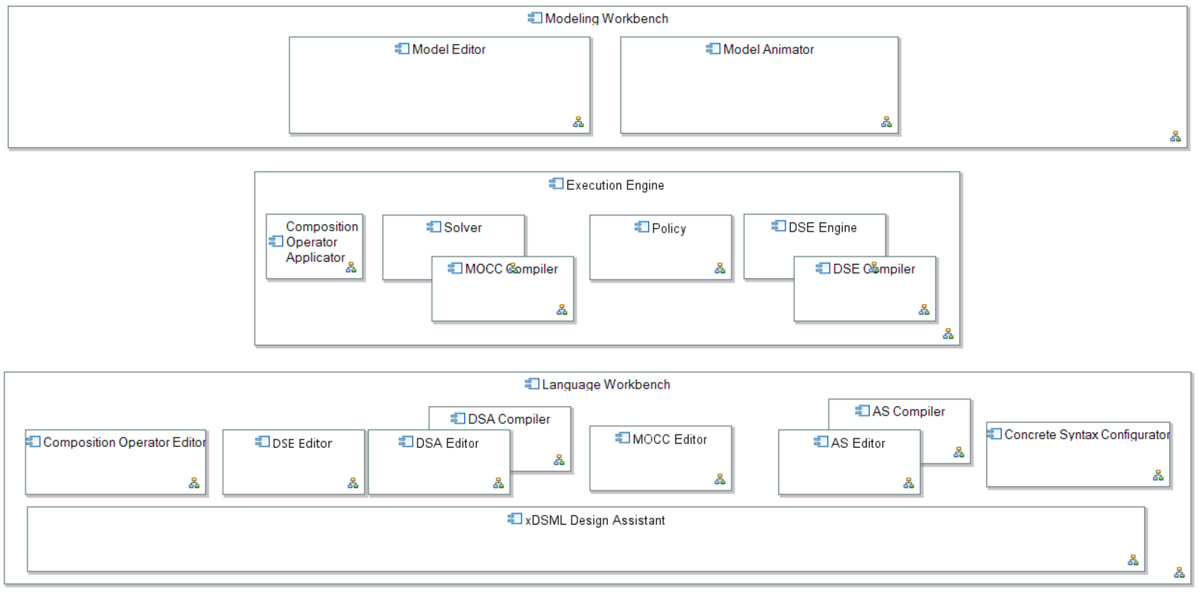


Figure 2.2: Architecture of the GEMOC Studio

2.1 Introduction

The logical architecture will be described with a model-based formalism describing components with input and output interfaces (including the data flow).

We can identify 3 main groups of components and artefacts in the studio.

- **GEMOC components:** are the components offered by the studio that constitutes the base of the studio and provide the main functionalities (see figures 2.3 and 2.4).
- **Metamodels:** these elements are the languages used by the "GEMOC components" to exchange data between them. (see figure 2.5).
- **Domain Specific Components:** These artefacts are the result of some of the GEMOC components and are specific to the domain. They are produced by the Domain Designers. Then, they are used by the Domain Experts in the main process as input of other component or as a tool to produce other artefact. As they are specific to the domain, these artifacts are prefixed with "Domain Specific". In a real domain application, this prefix would be replaced but the actual name of the domain. (see figure 2.6).

2.2 Components

2.2.1 EMF Tree Editor Generator

An `EMF Tree Editor` is in charge of generating a Domain Specific Editor. More specifically it generates a tree editor component for the xDSML.

Eclipse Modeling Framework provides such component. It uses the Domain Specific metamodel provided as an ecore model and a genmodel configuration.

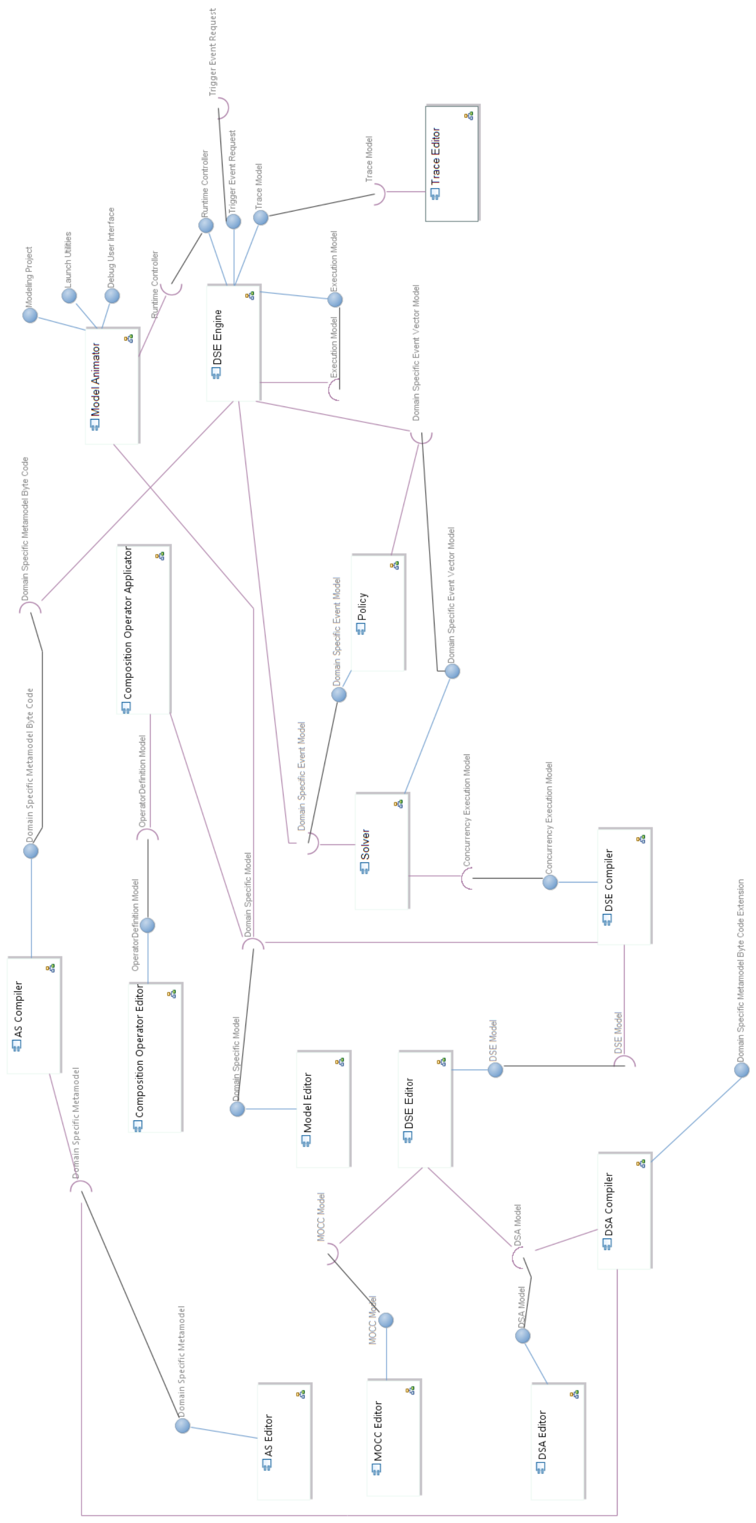


Figure 2.3: Logical View of the GEMOC Studio

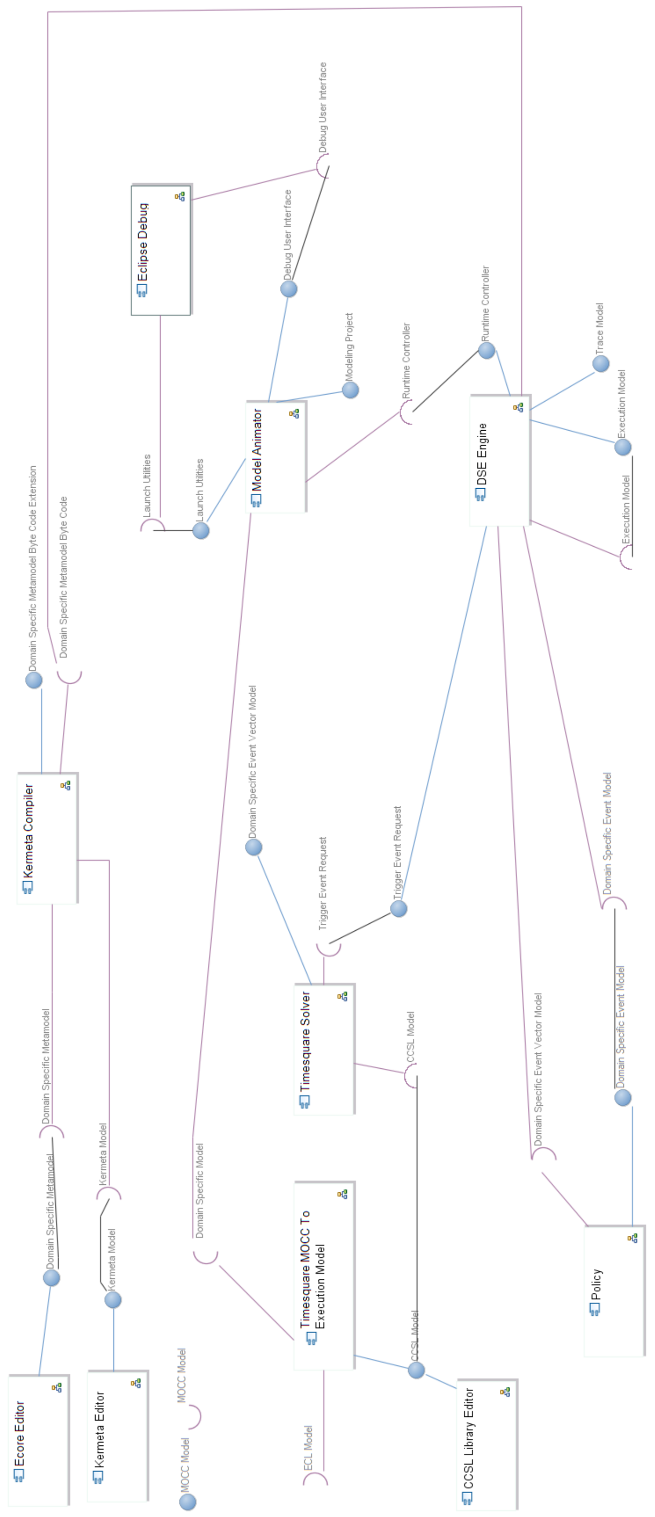


Figure 2.4: Implemetation view of the Execution Engine of the GEMOC Studio

Metamodels Class Diagram.jpg

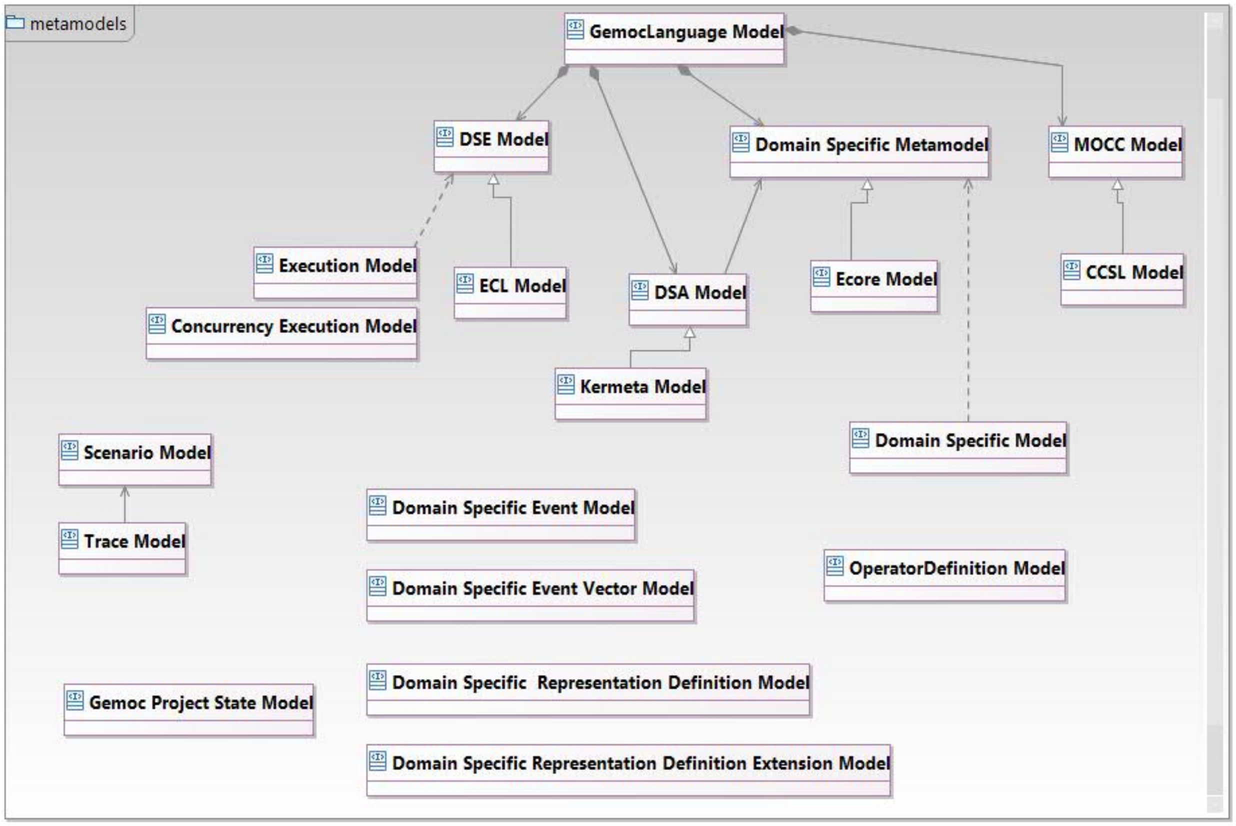
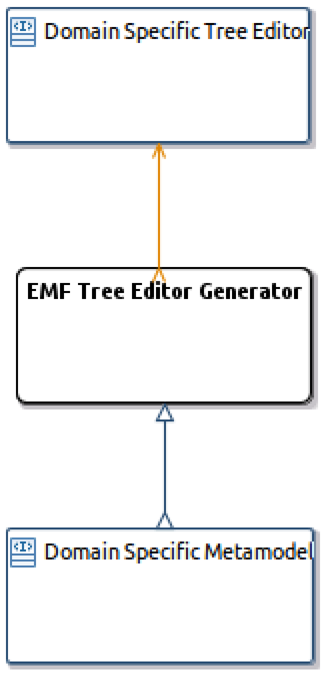

metamodels
GemocLanguage Model
DSE Model
Domain Specific Metamodel
MOCC Model
Execution Model
ECL Model
DSA Model
Ecore Model
CCSL Model
Concurrency Execution Model
Kermeta Model
Domain Specific Model
Scenario Model
Trace Model
Domain Specific Event Model
OperatorDefinition Model
Domain Specific Event Vector Model
Gemoc Project State Model
Domain Specific Representation Definition Model
Domain Specific Representation Definition Extension Model

Figure 2.5: List of the metamodels of the GEMOC Studio


Domain Specific Tree Editor
EMF Tree Editor Generator
Domain Specific Metamodel

This component port requires the following inputs:

- **Domain Specific Metamodel** : An ecore model representing the domain concepts of the user DSML. (see section 2.4.7.)

This component port provides the following outputs:

- **Domain Specific Tree Editor** : This component produces a new component that we will reference as *Domain Specific Tree Editor*. (see section 2.5.6.)

2.2.2 Eclipse

Eclipse is an integrated development environment composed of a runtime system and an plug-in mechanism used to extend and customize the environment. It is released under the terms of the Eclipse Public License, and is a free and open source software. The GEMOC studio will be based on and fully integrated to Eclipse. Consequently most of the end user GEMOC components will be released as Eclipse plugins.

2.2.3 Eclipse Debug

The Debug component of the Eclipse platform defines language independent facilities and mechanisms for:

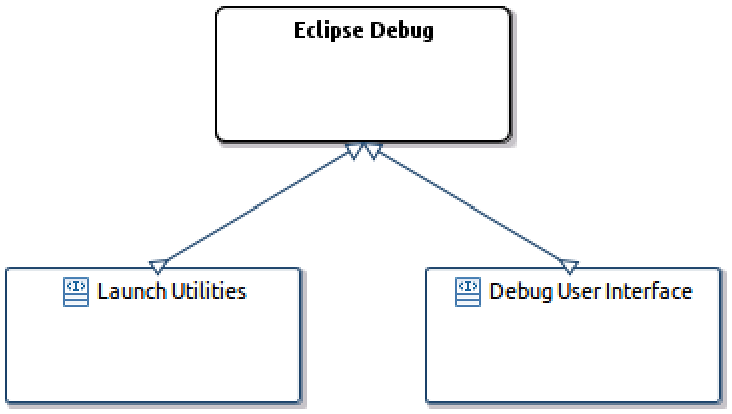
- Launching programs
- Source lookup
- Defining and registering breakpoints

- Event notification from programs being debugged
- A language independent debug model
- A language independent debug UI

The Eclipse debugger allows to control the execution process of a program interactively while watching the source code and the variables during the execution. It is possible to stop the execution on a specific element by defining breakpoints. When the program is paused, it is possible to see the variables current states and change their values.

Eclipse provides a special Debug perspective with default views and actions available:

- Set/Unset **breakpoints**
- **Start** and **Stop** the debugger
- Control the program execution : **Step into**, **Step over**, **Step out** and **Resume**
- See the **Call Stack**
- **Variables** view : display variables from the current executing stack and change the values assigned to a variable
- **Expressions** view : get the current value of an expression from the current executing stack



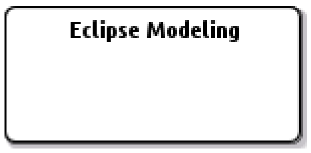
This component port requires the following inputs:

- **Debug User Interface** : Java API to provide a new debugging framework. It defines interfaces for a language independent debug model, which abstract common debugging features of many languages. (see section 2.5.1.)
- **Launch Utilities** : JAVA API to add launch configuration types to the platform. A launch configuration is a description of how to launch a program. Launching in Eclipse is closely tied to the infrastructure for debugging. (see section 2.5.8.)

2.2.4 Eclipse Modeling

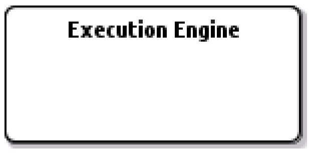
The Eclipse Modeling Project focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations. The GEMOC Studio is based on many different Eclipse modeling projects :

- EMF : a modeling framework and code generation facility for building tools and other applications based on a structured data model.
- GEF : provides technology to create rich graphical editors and views for the Eclipse Workbench UI.
- GMF : an application framework for creating graphical editors using EMF and GEF.
- Sirius : a framework which provides specific multi-view workbenches through graphical, table or tree modeling editors.
- Xtext : a framework/tool for developing external textual DSLs based on EMF.
- Unified Modeling Language 2.x : an EMF-based implementation of the UML 2.x metamodel for the Eclipse platform.
- Acceleo : a template based code generation framework with high quality tooling : complete Editor, Debugger and Profiler.
- ...



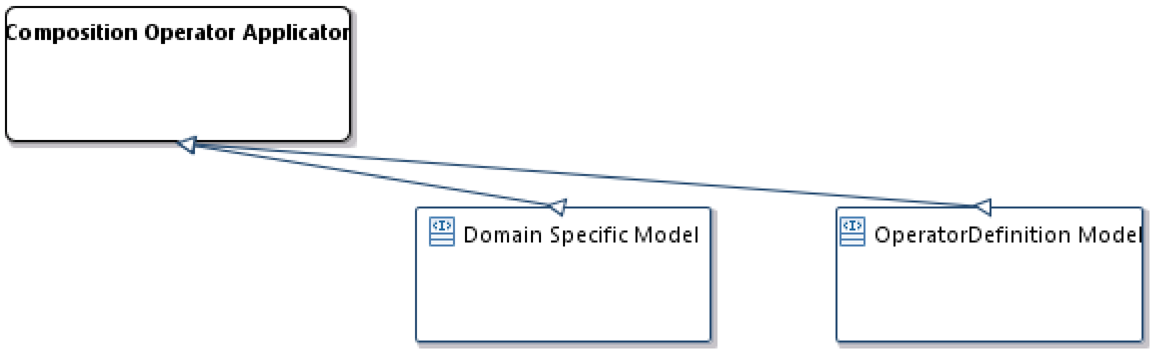
2.2.5 Execution Engine

An `Execution Engine` acts on the execution state of models under the direction of the Solver and Policy, and according to the corresponding domain specific action.



2.2.5.1 Composition Operator Applicator

A `Composition Operator Applicator` is a compiler of Composition Operator for specifics models (in compliance with the corresponding xDSMls).



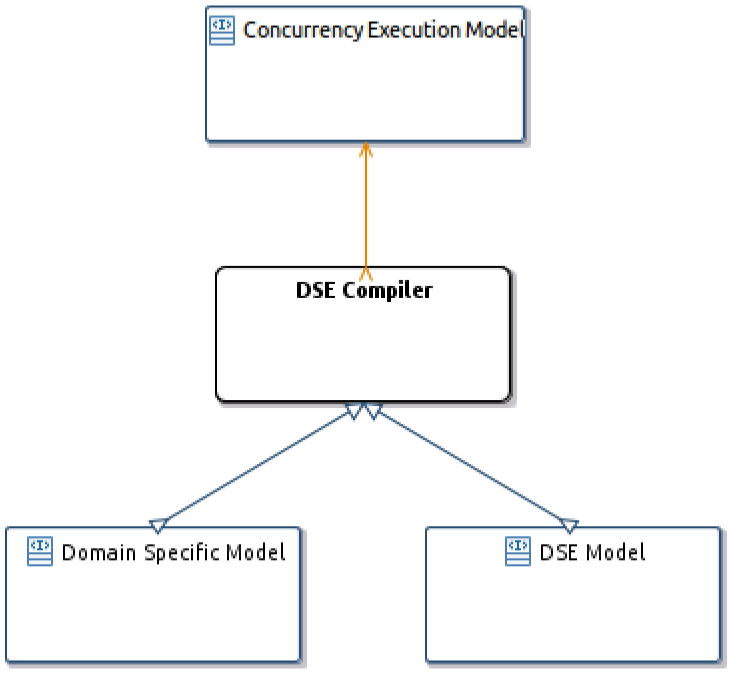
This component port requires the following inputs:

- **OperatorDefinition Model** : (see section 2.4.19.)
- **Domain Specific Model** : (see section 2.4.8.)

CEP Engine A `CEP Engine` provides a synchronization of two sets of steps events of various models from one coordination model.

2.2.5.2 DSE Compiler

A `DSE Compiler` provides a Concurrency Execution Model from a `Domain Specific Model`, and a `DSE Model`.



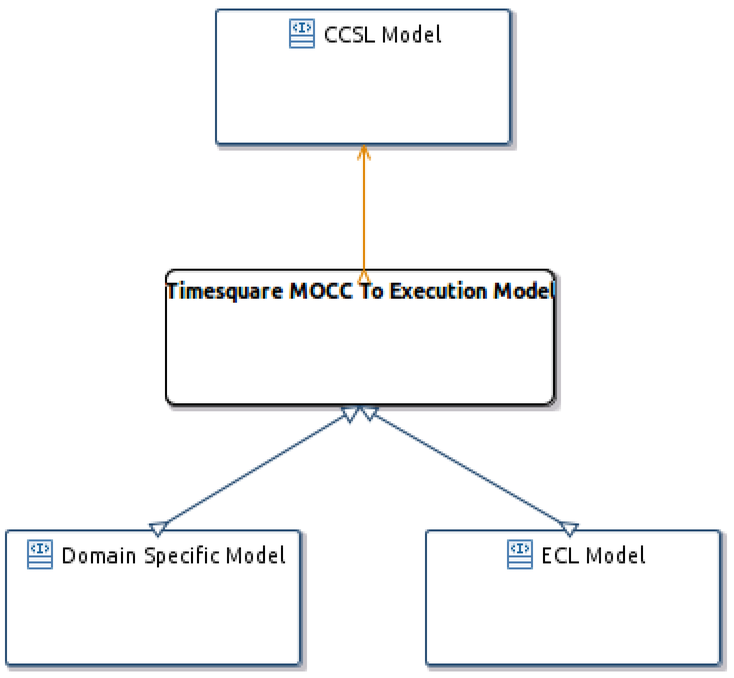
This component port requires the following inputs:

- **DSE Model** : (see section 2.4.4.)
- **Domain Specific Model** : (see section 2.4.8.)

This component port provides the following outputs:

- **Concurrency Execution Model** : (see section 2.4.2.)

Timesquare MOCC To Execution Model



This component port requires the following inputs:

- **ECL Model** : (see section 2.4.11.)

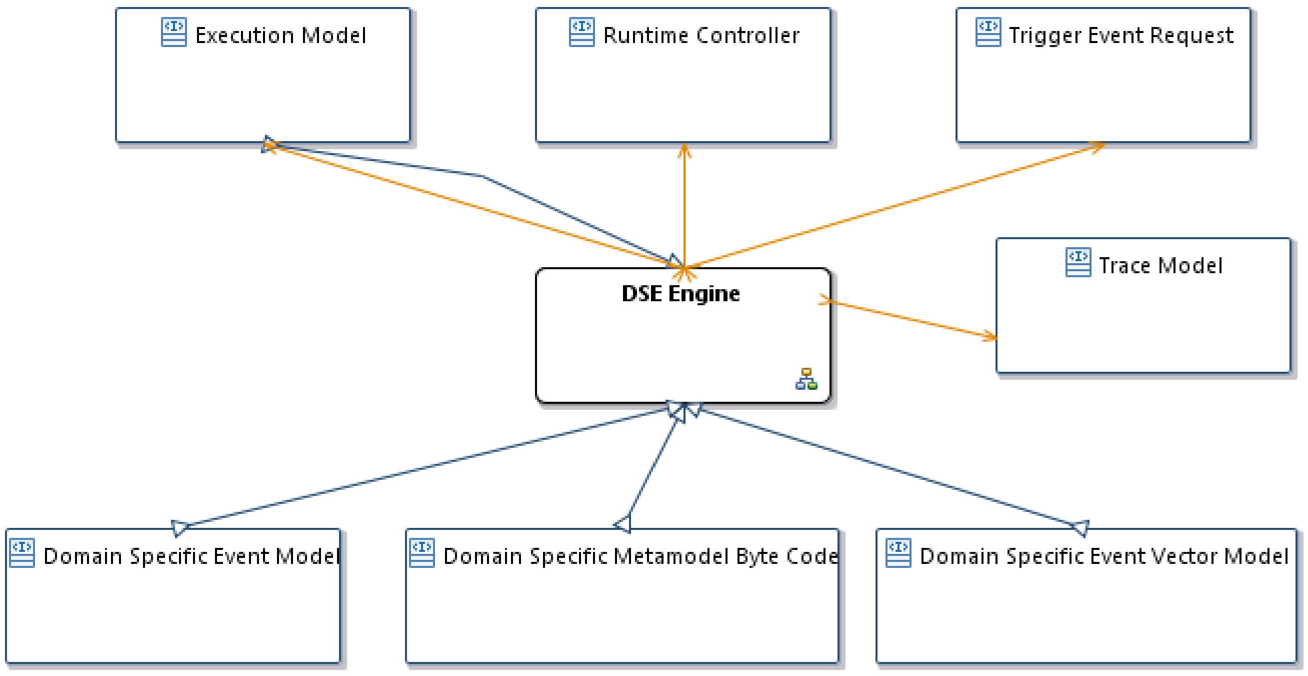
- **Domain Specific Model** : (see section 2.4.8.)

This component port provides the following outputs:

- **CCSL Model** : (see section 2.4.1.)

2.2.5.3 DSE Engine

A `DSE Engine` acts on an `Execution Model`. It provides an execution trace.


Execution Model
Runtime Controller
Trigger Event Request
DSE Engine
Trace Model
Domain Specific Event Model
Domain Specific Metamodel Byte Code
Domain Specific Event Vector Model

This component port requires the following inputs:

- **Domain Specific Event Vector Model** : (see section 2.4.6.)
- **Domain Specific Event Model** : (see section 2.4.5.)
- **Execution Model** : (see section 2.4.13.)
- **Domain Specific Metamodel Byte Code** : (see section 2.5.4.)

This component port provides the following outputs:

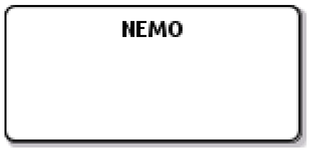
- **Execution Model** : (see section 2.4.13.)
- **Trigger Event Request** : (see section 2.5.11.)
- **Runtime Controller** : (see section 2.5.10.)
- **Trace Model** : (see section 2.4.21.)

2.2.5.4 MOCC Compiler

A `MOCC compiler` takes the specification of a `MOCC Model` to provide an implementation of this model.

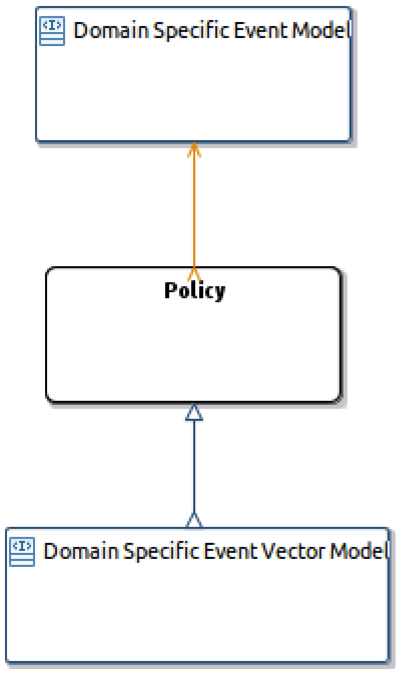
2.2.5.5 NEMO

`NEMO` is a specific MOCC based on a state machin. It interprets one concurrency model to schedule events.



2.2.5.6 Policy

A `Policy` is part of the description of the control in a language. It works in conjunction with a model of computation. The Policy component is an abstract componant whose implementations are responsible for selecting one Domain Specific Event from a vector of triggerable Domain Specific Event. Several selection strategies are provided in GEMOC. The purpose of a Policy is to select a DSE among the possible DSE occurrences allowed by the model of computation. It can be considered as a refinement of the model of computation.


Domain Specific Event Model
Policy
Domain Specific Event Vector Model

This component port requires the following inputs:

- **Domain Specific Event Vector Model** : (see section 2.4.6.)

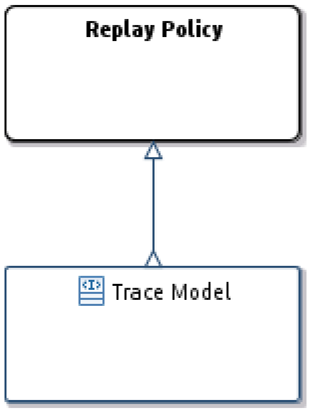
This component port provides the following outputs:

- **Domain Specific Event Model** : The Policy component provides one Domain Specific Event selected out of the triggerable set. (see section 2.4.5.)

Interactive Policy The Interactive Policy is a policy for selecting a DSE among the all the DSE that are allowed to occur according to the model of computation. It inherits from Policy component and have all its input/outputs. The purpose of the Interactive Policy is to let the user explore the possible behaviors of a model by choosing the DSE he wants when several DSE are allowed to occur according to the model of computation.

Random Policy The Random Policy is a policy for selecting a DSE among the all the DSE that are allowed to occur according to the model of computation. The purpose of the Random Policy is to randomly explore the possible behaviors of a model by making a random choice among all possible DSE occurrences. For instance, when executing a non deterministic automata, the random policy will randomly choose one transition among all the fireable transitions determined by the model of computation.

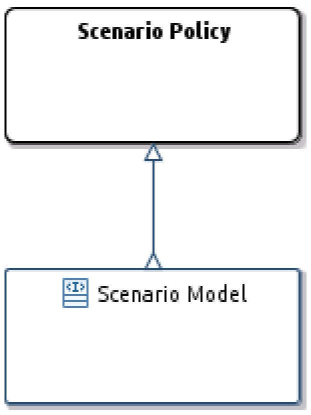
Replay Policy The Replay Policy is a policy for selecting a DSE among the all the DSE that are allowed to occur according to the model of computation. It inherits from Policy component and have all its input/outputs. The purpose of the Replay Policy is to replay a simulation by choosing the DSE indicated in a trace model when several DSE when several DSE are allowed to occur according to the model of computation. It should also raise an error when the DSE which is indicated at a given point in the trace model is not among the DSE allowed by the model of computation (the trace model is not compatible with the behavior of the model).



This component port requires the following inputs:

- **Trace Model** : the model of the trace to replay. (see section 2.4.21.)

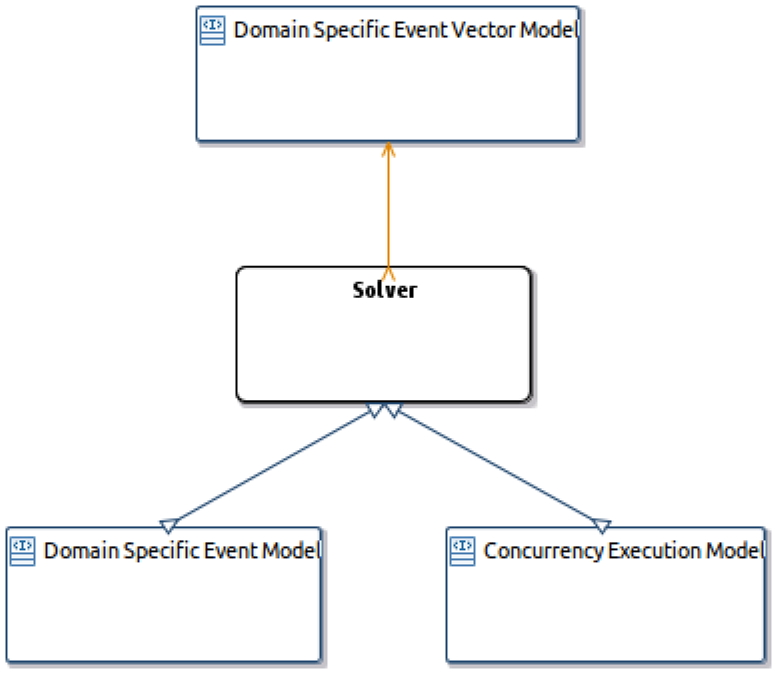
Scenario Policy The `Scenario Policy` is a policy for selecting a DSE among the all the DSE that are allowed to occur according to the model of computation. It inherits from Policy component and have all its input/outputs. The purpose of the Scenario Policy is to drive a simulation by choosing the DSE indicated in a scenario when several DSE when several DSE are allowed to occur according to the model of computation.



This component port requires the following inputs:

- **Scenario Model** : (see section 2.4.20.)

2.2.5.7 Solver


Domain Specific Event Vector Model
Solver
Domain Specific Event Model
Concurrency Execution Model

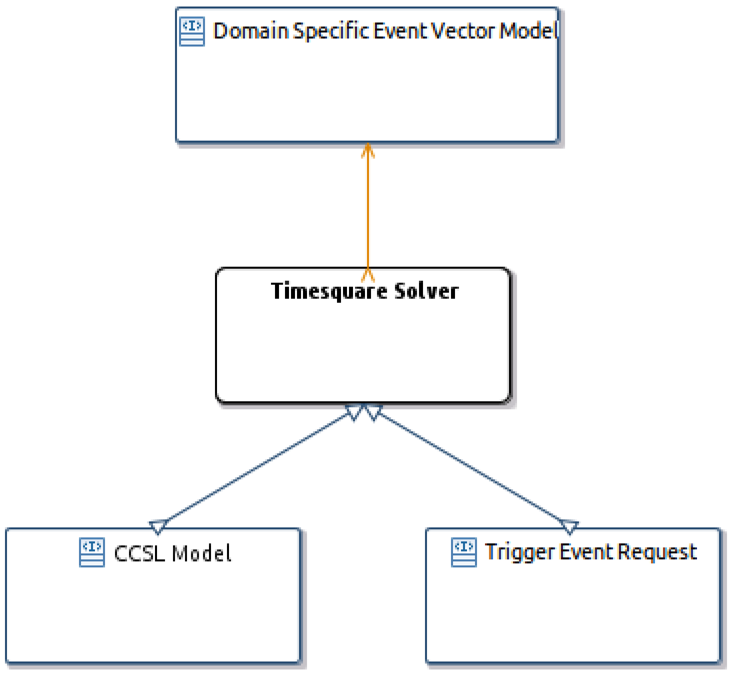
This component port requires the following inputs:

- **Domain Specific Event Model** : (see section 2.4.5.)
- **Concurrency Execution Model** : (see section 2.4.2.)

This component port provides the following outputs:

- **Domain Specific Event Vector Model** : (see section 2.4.6.)

Timesquare Solver



This component port requires the following inputs:

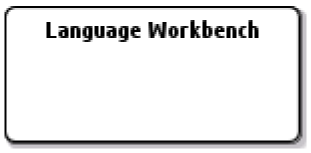
- **CCSL Model** : (see section 2.4.1.)
- **Trigger Event Request** : (see section 2.5.11.)

This component port provides the following outputs:

- **Domain Specific Event Vector Model** : (see section 2.4.6.)

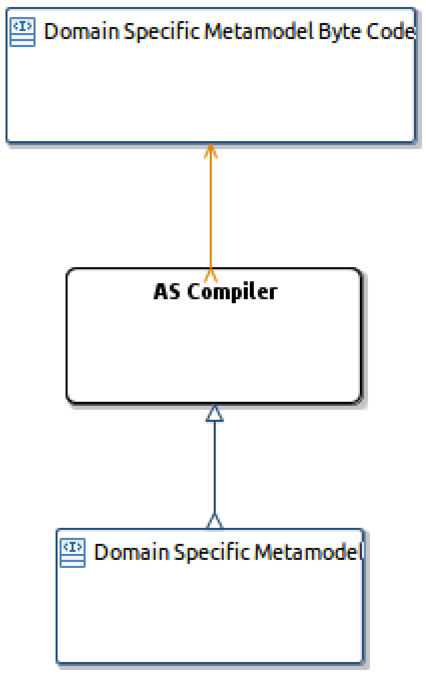
2.2.6 Language Workbench

A `Language Workbench` is a tool to design a language.



2.2.6.1 AS Compiler

An `AS Compiler` takes the specification of an `AS Model` to provide an implementation of this model.


Domain Specific Metamodel Byte Code
AS Compiler
Domain Specific Metamodel

This component port requires the following inputs:

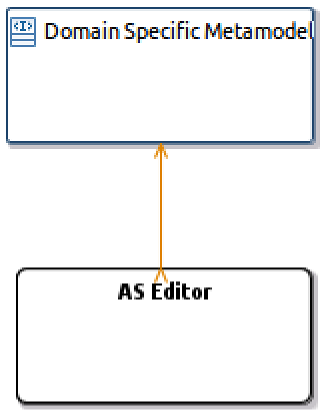
- **Domain Specific Metamodel** : (see section 2.4.7.)

This component port provides the following outputs:

- **Domain Specific Metamodel Byte Code** : (see section 2.5.4.)

2.2.6.2 AS Editor

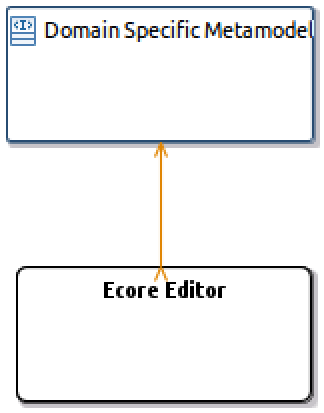
An `AS Editor` is a tool to design the abstract syntax of a modeling language.



This component port provides the following outputs:

- **Domain Specific Metamodel** : (see section 2.4.7.)

Ecore Editor The Ecore Editor is a tool to design the Ecore model of a modeling language.

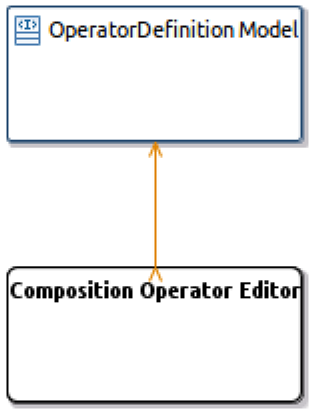


This component port provides the following outputs:

- **Domain Specific Metamodel** : This component produces an ecore model. (see section 2.4.7.)

2.2.6.3 Composition Operator Editor

A `Composition Operator Editor` is a tool to design a composition between two xDSMLs

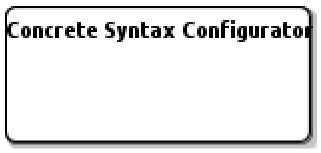


This component port provides the following outputs:

- **OperatorDefinition Model** : (see section 2.4.19.)

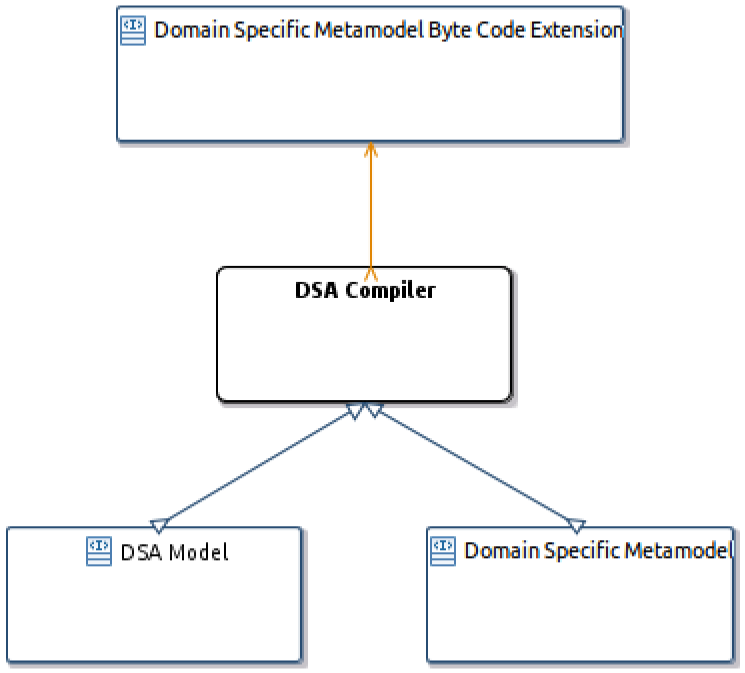
2.2.6.4 Concrete Syntax Configurator

A `Concrete Syntax Configurator` is a tool to design a concrete syntax for a domain specific modeling language.



2.2.6.5 DSA Compiler

A `DSA Compiler` takes the specification of a DSA model to provide an implementation of this model.


Domain Specific Metamodel Byte Code Extension
DSA Compiler
DSA Model
Domain Specific Metamodel

This component port requires the following inputs:

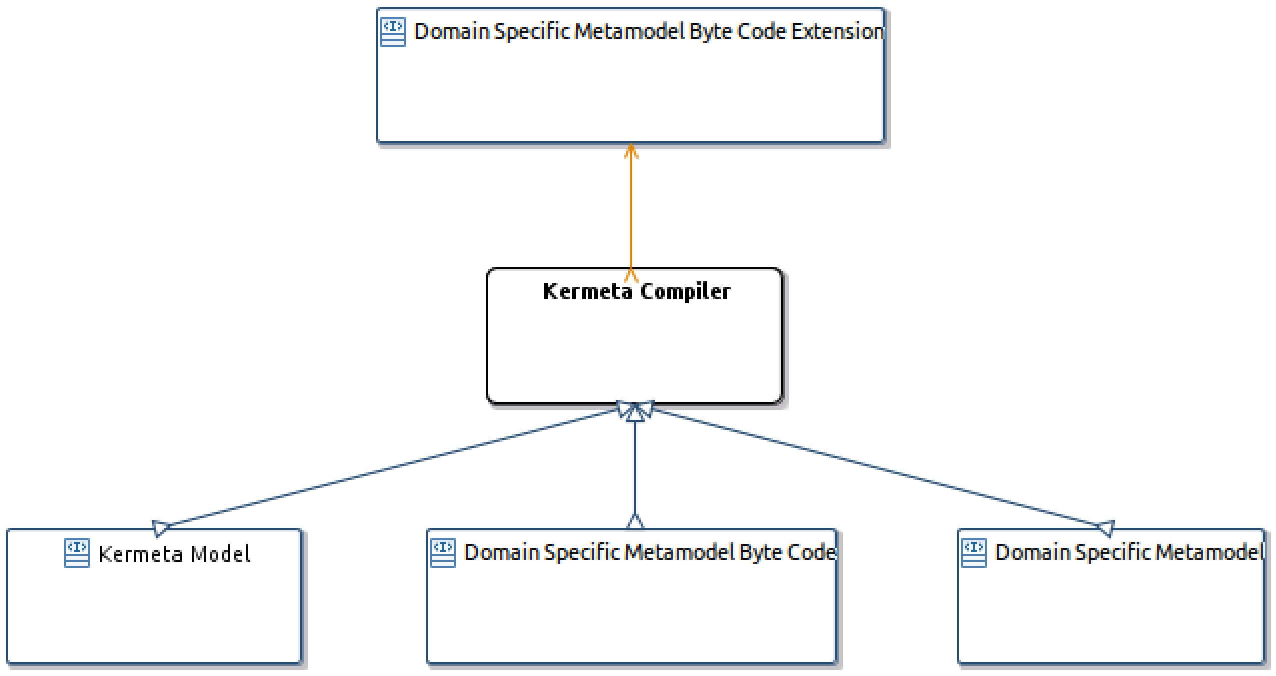
- **DSA Model** : A model of DSA. (see section 2.4.3.)

- **Domain Specific Metamodel** : (see section 2.4.7.)

This component port provides the following outputs:

- **Domain Specific Metamodel Byte Code Extension** : The bytecode implementing the actions in the DSA. (see section 2.5.5.)

Kermeta Compiler



Domain Specific Metamodel Byte Code Extension
Kermeta Compiler
Kermeta Model
Domain Specific Metamodel Byte Code
Domain Specific Metamodel

This component port requires the following inputs:

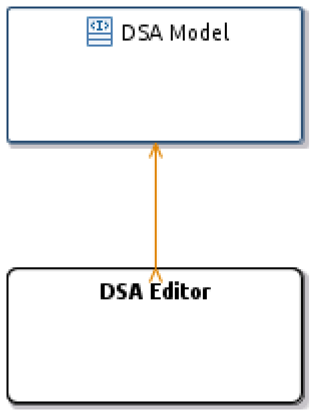
- **Kermeta Model** : (see section 2.4.17.)
- **Domain Specific Metamodel Byte Code** : (see section 2.5.4.)
- **Domain Specific Metamodel** : (see section 2.4.7.)

This component port provides the following outputs:

- **Domain Specific Metamodel Byte Code Extension** : (see section 2.5.5.)

2.2.6.6 DSA Editor

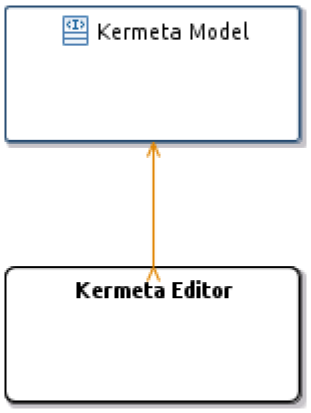
A `DSA Editor` is a tool to design `DSA Models`. These models define the behavorial semantics of the actions in the Domain Specific Metamodel.



This component port provides the following outputs:

- **DSA Model** : (see section 2.4.3.)

Kermeta Editor A `Kermeta Editor` is used in the context of GEMOC as a `DSA editor`. `Kermeta Editor` is a tool to design `Kermeta models`. These models defines the operationnal semantics of the actions in the DSM.

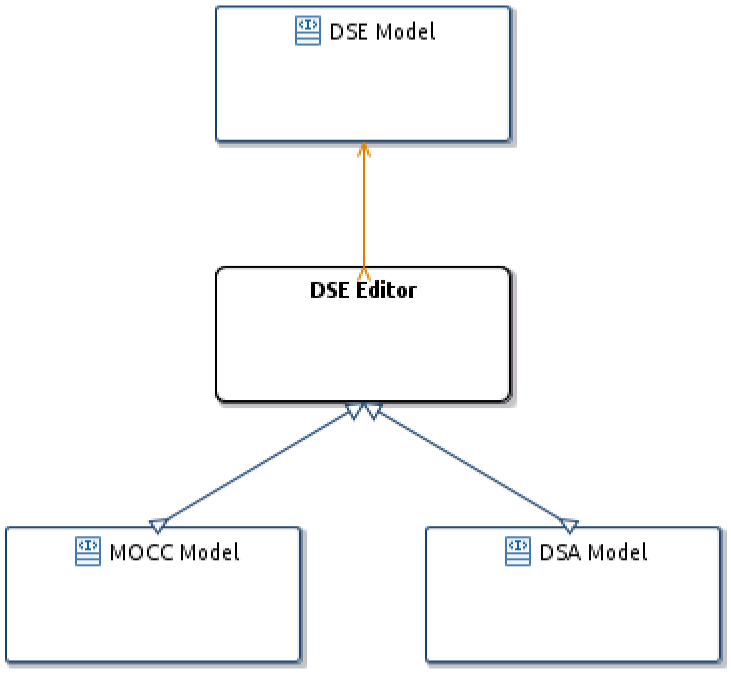


This component port provides the following outputs:

- **Kermeta Model** : This component produces a Kermeta model that will constitute the DSA part of Domain Specific Language according to GEMOC methodology (see section 2.4.17.)

2.2.6.7 DSE Editor

A `DSE Editor` is a tool to design `DSE Models`.
To design a DSE Model, DSE Editor needs to know the model of the MOCC and the DSA associated.



This component port requires the following inputs:

- **MOCC Model** : (see section 2.4.18.)

- **DSA Model** : (see section 2.4.3.)

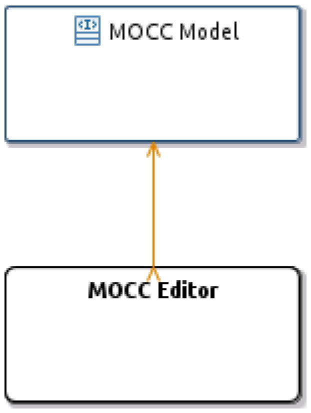
This component port provides the following outputs:

- **DSE Model** : (see section 2.4.4.)

ECL Editor

2.2.6.8 MOCC Editor

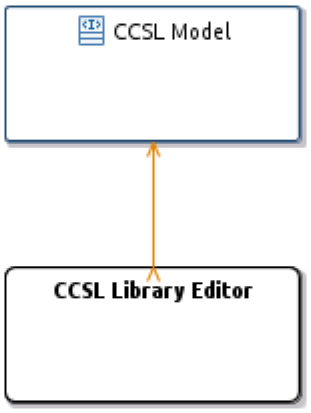
A `MOCC Editor` is a tool to design a `MOCC Model` that defined operational semantics.



This component port provides the following outputs:

- **MOCC Model** : (see section 2.4.18.)

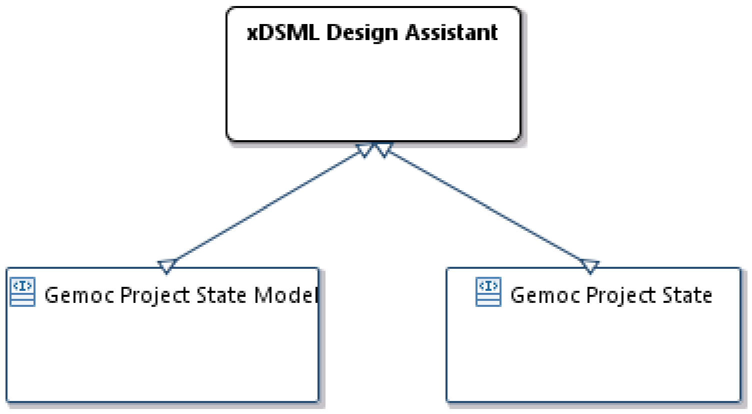
CCSL Library Editor



This component port provides the following outputs:

- **CCSL Model** : (see section 2.4.1.)

2.2.6.9 xDSML Design Assistant

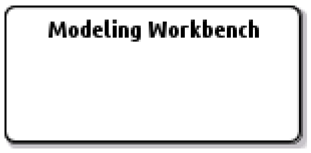


This component port requires the following inputs:

- **Gemoc Project State Model** : (see section 2.4.15.)
- **Gemoc Project State** : (see section 2.4.14.)

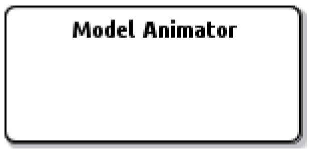
2.2.7 Modeling Workbench

A `Modeling Workbench` is a tool to run languages.



2.2.7.1 Model Animator

A `Model Animator` is a model of a visual animation which consists to display at run time the behavior of an `Execution Model`.



This component port requires the following inputs:

- **Runtime Controller** : (see section 2.5.10.)
- **Domain Specific Model** : (see section 2.4.8.)

This component port provides the following outputs:

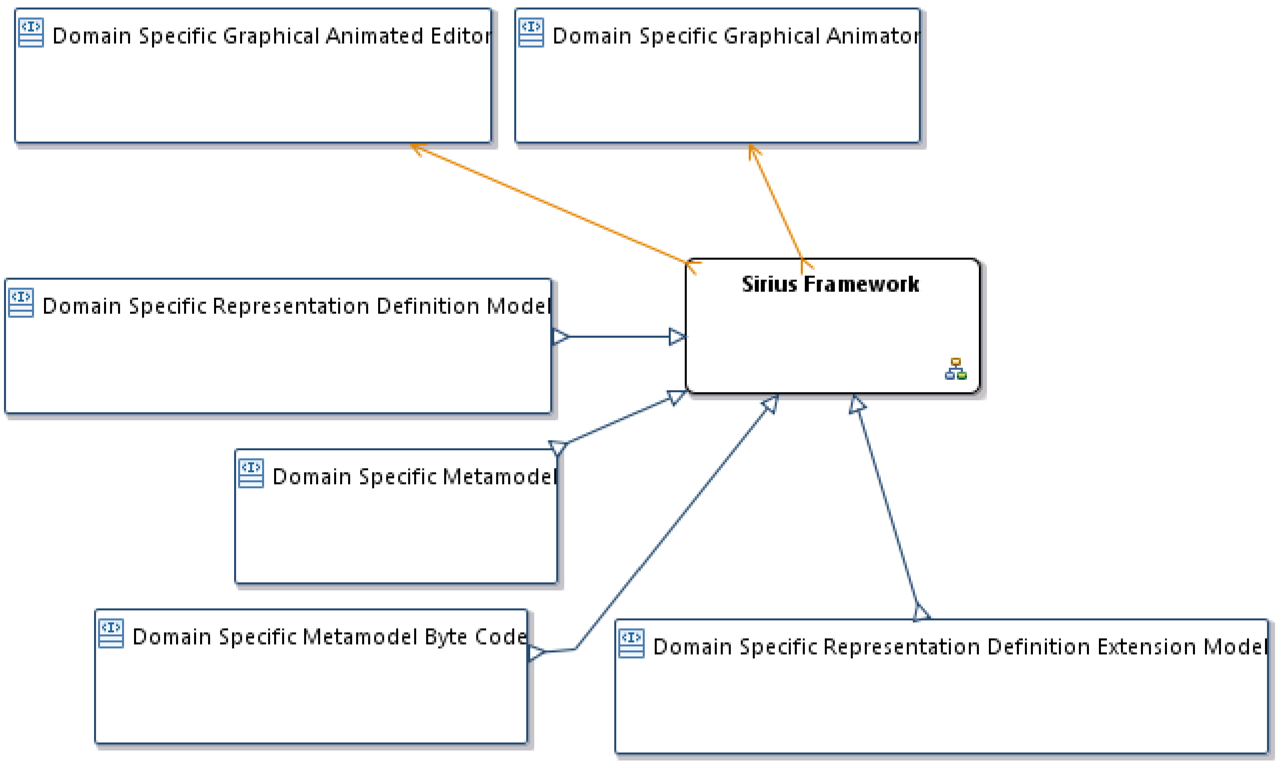
- **Debug User Interface** : (see section 2.5.1.)
- **Launch Utilities** : (see section 2.5.8.)
- **Modeling Project** : (see section 2.5.9.)
- **Domain Specific Graphical Animated Editor** : (see section 2.5.2.)

Sirius Framework The Sirius framework is an Eclipse project which allows to define multi-view workbenches through graphical, table or tree modeling editors. From the specifier/developer perspective, Sirius provides:

- The ability to define workbenches providing editors including diagrams, tables or trees.
- The ability to integrate and deploy the aforementioned environment into Eclipse IDE's or RCP applications.
- The ability to customize existing environments by specialization and extension.

From the end user perspective, Sirius provides:

- Rich and specialized modeling editors to design their models.
- Synchronization between these different editors.


Domain Specific Graphical Animated Editor
Domain Specific Graphical Animator
Domain Specific Representation Definition Model
Sirius Framework
Domain Specific Metamodel
Domain Specific Metamodel Byte Code
Domain Specific Representation Definition Extension Model

This component port requires the following inputs:

- **Domain Specific Metamodel** : Eclipse EMF is used to model the domain model. The meta-model describes the structure of the domain specific model. (see section 2.4.7.)
- **Domain Specific Representation Definition Model** : (see section 2.4.10.)
- **Domain Specific Metamodel Byte Code** : Once the EMF meta-model is specified it is possible to generate the corresponding Java implementations classes from this model. As Sirius relies on the EMF reflexive API, it is able to work with any kind of metamodel byte code which implements this API. (see section 2.5.4.)

- **Domain Specific Representation Definition Extension Model** : Sirius has specific support for extending and refining diagram descriptions and mappings without modifying the original. The extensions are applied transparently when the viewpoint they are defined in are enabled, and removed when the viewpoint is disabled. The purpose is to augment the base diagram with new graphical elements related to the animation feature. (see section 2.4.9.)

This component port provides the following outputs:

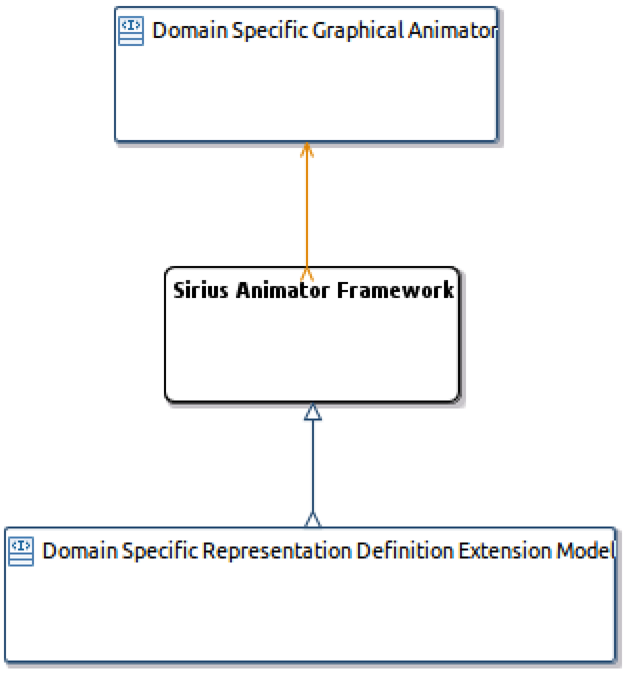
- **Domain Specific Graphical Animator** : Sirius will provide an animator integrated to the domain specific graphical editor which allows to animate and debug a model. (see section 2.5.3.)
- **Domain Specific Graphical Animated Editor** : Sirius provides a completely customized graphical editor dedicated to a domain specific languages (DSL). The graphical designers created with Sirius provide diagrams, tables, matrix or trees to create and edit models according to the domain specific vocabulary and process. (see section 2.5.2.)

Sirius Animator Framework The Sirius animator framework is a part of the Sirius project which allows to animate graphical editors created with Sirius. From the specifier/developer perspective, Sirius animator provides:

- The ability to customize existing environments by specialization and extension.

From the end user perspective, Sirius provides:

- Graphical editor enhanced with animation features.


Domain Specific Graphical Animator
Sirius Animator Framework
Domain Specific Representation Definition Extension Model

This component port requires the following inputs:

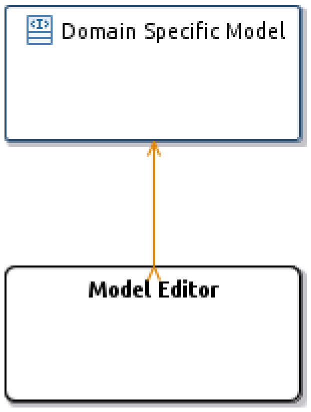
- **Domain Specific Representation Definition Extension Model** : Sirius has specific support for extending and refining diagram descriptions and mappings without modifying the original. The purpose is to augment the base diagram with new graphical elements related to the animation feature. (see section 2.4.9.)

This component port provides the following outputs:

- **Domain Specific Graphical Animator** : The Sirius animator framework allows to provide to domain specific graphical editor created with Sirius the ability to animate models. (see section 2.5.3.)

2.2.7.2 Model Editor

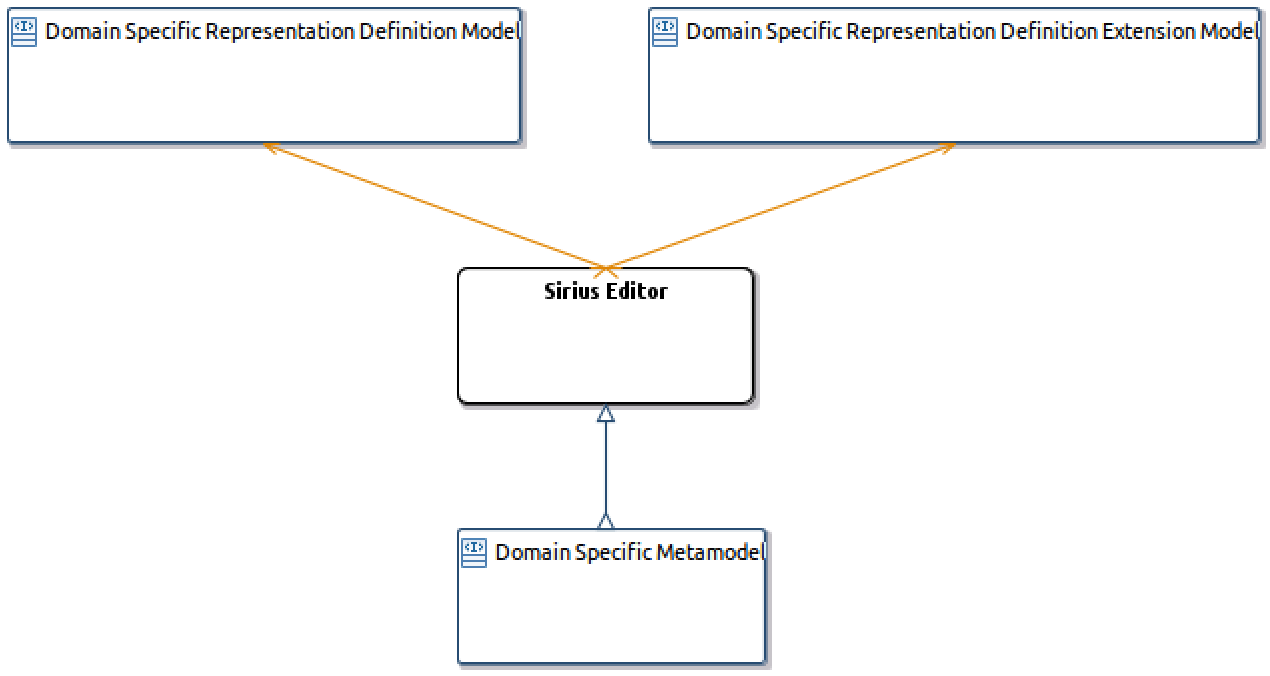
A `Model Editor` is a tool to design `Domain Specific Models`.



This component port provides the following outputs:

- **Domain Specific Model** : (see section 2.4.8.)

Sirius Editor The Sirius editor provides tools to specify modeling workbenches. A graphical designer allows users to create or visualize models of a given domain specific model defined with EMF (Ecore model) using the Ecore graphical designer. Then the user has to describe its graphical designer. This is done thanks to a model which is interpreted by Sirius. This model contains viewpoints which define representations (diagrams, tables, matrix and trees). Each representation defines how the model elements can be graphically represented (shape, color, font, ...) and edited (palette, tools, ...). The Sirius editor is a graphical editor used to edit this viewpoint specification model.


Domain Specific Representation Definition Model
Domain Specific Representation Definition Extension Model
Sirius Editor
Domain Specific Metamodel

This component port requires the following inputs:

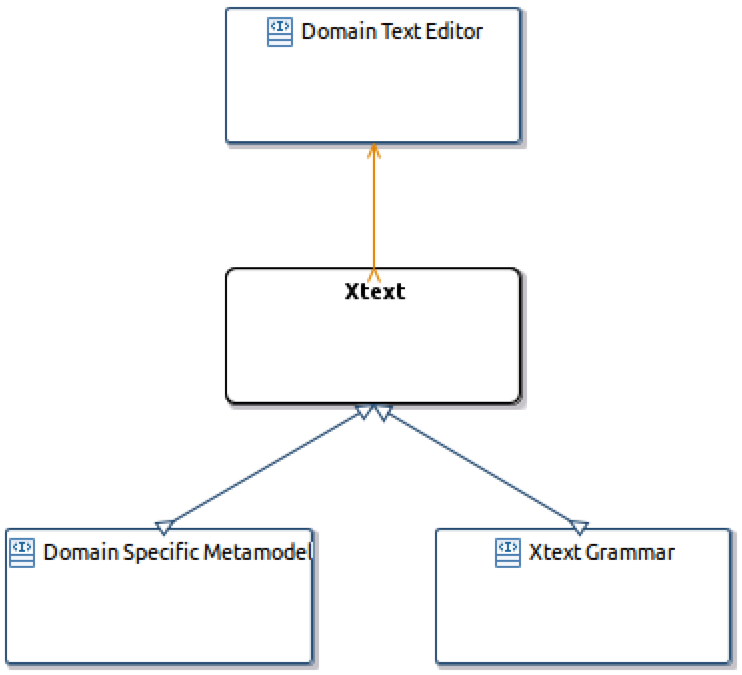
- **Domain Specific Metamodel** : The EMF meta-model which describes the structure of the domain specific model. (see section 2.4.7.)

This component port provides the following outputs:

- **Domain Specific Representation Definition Model** : (see section 2.4.10.)

- **Domain Specific Representation Definition Extension Model** : Configuration file which extends and refines existing diagram descriptions and mappings. (see section 2.4.9.)

Xtext Xtext is an open source framework available in Eclipse for development of domain specific languages. It covers all aspects of a complete language infrastructure, from parsers, over linker, compiler or interpreter and provides an easy way to generate textual editor fully integrated in Eclipse. Xtext uses EMF models as the in-memory representation of any parsed text files. This in-memory object graph is called the Abstract Syntax Tree (AST). It is possible to give to Xtext as input an existing domain specific metamodel or a grammar.



This component port requires the following inputs:

- **Xtext Grammar** : To create a new domain specific language with Xtext, the first thing to do is to define the grammar of the new language. The Xtext grammar language is a domain specific language designed for the description of textual languages. The main idea is to describe the concrete syntax and how it is mapped to an in-memory representation - the semantic model. This model will be produced by the parser on-the-fly when it consumes an input file. (see section 2.4.22.)

- **Domain Specific Metamodel** : The domain specific metamodel from which Xtext is able to generate a textual editor. The language in which the meta model is defined is called Ecore. In this case Xtext will generate the Xtext grammar automatically based on the given metamodel. (see section 2.4.7.)

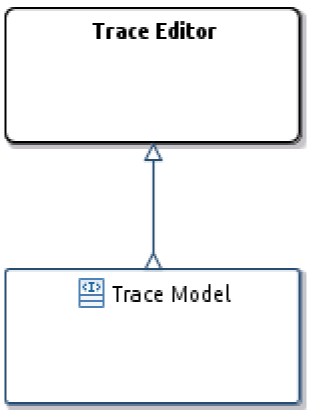
This component port provides the following outputs:

- **Domain Text Editor** : The generated textual editor which provides syntax coloring, content assists, validation and quick fixes fully integrated in Eclipse. (see section 2.5.7.)

2.2.8 Trace Editor

A `trace editor` takes as input a `Trace model` and allows the analysis of this trace by the user. The purpose of the Trace Editor is to present different views on a trace by masking some events, highlighting relations between event occurrences in the trace (coincidence for instance) and searching for occurrence patterns (for instance, looking for all occurrences of C that follow an occurrence of A without an occurrence of B in between).

If the trace model supports a notion of date associated to an occurrence, the trace editor should also allow searching for sequences in a trace that have timing properties (for instance, two consecutive occurrences of A with dates separated by more than some duration).



This component port requires the following inputs:

- **Trace Model** : the model of the trace to be edited. (see section 2.4.21.)

2.3 Domain Specific Components

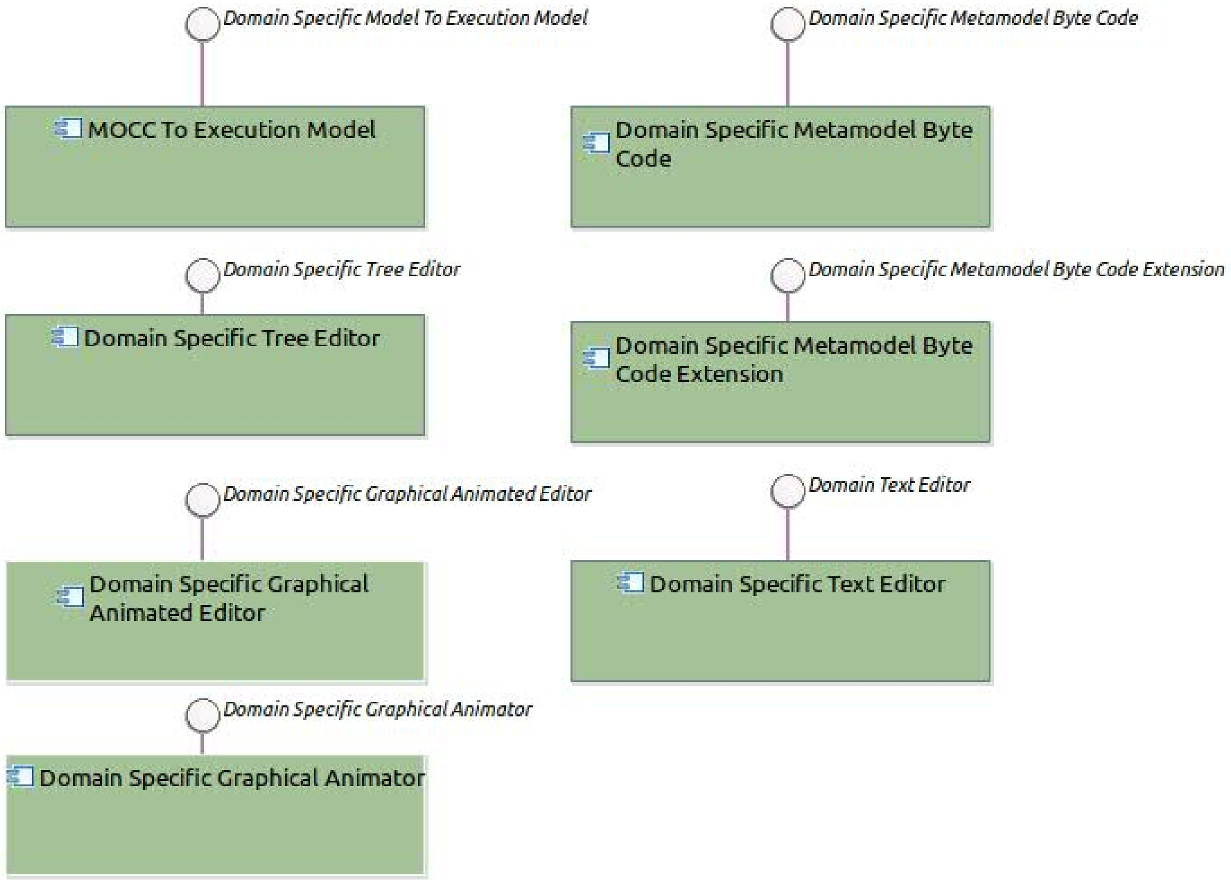

Domain Specific Model To Execution Model
MOCC To Execution Model
Domain Specific Metamodel Byte Code
Domain Specific Metamodel Byte Code
Domain Specific Tree Editor
Domain Specific Tree Editor
Domain Specific Metamodel Byte Code Extension
Domain Specific Metamodel Byte Code Extension
Domain Specific Graphical Animated Editor
Domain Specific Graphical Animated Editor
Domain Text Editor
Domain Specific Text Editor
Domain Specific Graphical Animator
Domain Specific Graphical Animator

Figure 2.6: List of the Domain Specific components

A `Domains Specific Components` are components produced by the language designer by using the language workbench provided by the GEMOC Studio. The figure 2.6 illustrates the point that they are output of some other component.

2.4 Metamodels

This section describes the metamodels used by the components in order to exchange models.

2.4.1 CCSL Model

A `CCSL (Clock Constraint Specification Language) Model` specifies the causal and temporal relationships between the events of a DSML model
A CCSL model represents the acceptable scheduling of the events of a specific DSML model, by applying the constraints from a MoCC, as specified by an ECL model.

2.4.2 Concurrency Execution Model

2.4.3 DSA Model

2.4.4 DSE Model

2.4.5 Domain Specific Event Model

2.4.6 Domain Specific Event Vector Model

2.4.7 Domain Specific Metamodel

2.4.8 Domain Specific Model

2.4.9 Domain Specific Representation Definition Extension Model

Sirius has specific support for extending and refining diagram descriptions and mappings without modifying the original. The purpose is to augment the base diagram with new graphical elements. In fact this metamodel is just a subpart of the domain specific representation definition metamodel.

2.4.10 Domain Specific Representation Definition Model

2.4.11 ECL Model

An `ECL (Event Constraint Language) Model` is defining the link between the couple AS/DSA and the MoCC.
It gets the definitions from the MoCC library and specify how they are actually used according to the AS. It also specifies (in a very crude way for now) how some events are associated to the DSA functions.
It is specified at the language level.

2.4.12 Ecore Model

2.4.13 Execution Model

The Execution Model is used to describe the state of a model during its execution. This includes attributes that are used and modified by the DSA, as well as execution state attributes used by the model of computation.

2.4.14 Gemoc Project State

2.4.15 Gemoc Project State Model

This structure is used to capture the state of the GEMOC process in the IDE. It is then used by the wizard component to drive the user in his xDSML design tasks.

2.4.16 GemocLanguage Model

This structure represents the definition of a language according to GEMOC methodology.

It is composed of a DSE, a DSA and a Domain Specific Metamodel.

2.4.17 Kermeta Model

In GEMOC, Kermeta language is used as a DSA. It is able to specify the actions of a Domain Specific Metamodel.

Kermeta metamodel uses an operationnal semantics to weave the actions in the DSM.

2.4.18 MOCC Model

2.4.19 OperatorDefinition Model

2.4.20 Scenario Model

The Scenario Model is used to represent input scenarios for the Scenario Policy. An input scenario contains the information needed to choose a DSE among the DSE occurrences that can occur according to the model of computation. A scenario is more abstract than a trace (which makes a difference between the Scenario Policy and the Replay Policy) and may consist in tactics like “alternate between two choices”.

2.4.21 Trace Model

The Trace Model is used to represent execution traces as sequences of domain specific event occurrences, eventually annotated with values when the behavior of the model produces data.

2.4.22 Xtext Grammar

2.5 Component Interfaces

This section describes the interfaces used by the components.

2.5.1 Debug User Interface

2.5.2 Domain Specific Graphical Animated Editor

2.5.3 Domain Specific Graphical Animator

2.5.4 Domain Specific Metamodel Byte Code

2.5.5 Domain Specific Metamodel Byte Code Extension

2.5.6 Domain Specific Tree Editor

2.5.7 Domain Text Editor

2.5.8 Launch Utilities

2.5.9 Modeling Project

2.5.10 Runtime Controller

2.5.11 Trigger Event Request

3. Component integration process

In order to create the GEMOC Studio, the components and metamodels must follow our development process.

3.1 Providing and updating a contribution

3.1.1 Minimum requirement for a component

A component should be integrated to the GEMOC Studio if the component :

- is referenced in the GEMOC architecture model
- is a set of eclipse plugins/OSGI bundles
- has one feature referencing all plugins/bundles
- compiles using a maven script
- is deployed as a P2 repository (on Jenkins server or external URL)
- exposed a minimal documentation as Eclipse help (Quick start)
- offers a package sample (accessible through File¿New¿Sample in eclipse)

3.1.2 Minimum requirement for a metamodel

- is referenced in the GEMOC architecture model
- is distributed as a part of an editor component (should follow the component minimal requirements)

3.2 Continuous integration

The resulting GEMOC Studio is available via the continuous integration server at the link below :
`http://ci.inria.fr/gemoc/job/org.gemoc.gemoc_studio.root/`

4. Appendix A : Tutorial "How to deliver a GEMOC Component distribution"

This annex presents some guidelines to help the partners building the GEMOC Studio components.

4.1 Sources

4.1.1 SCM

The sources of the metamodels, components and samples developed for the project should be stored in the git repository available from the forge at link below :
`https://gforge.inria.fr/projects/gemoc-dev/`

Typical source retrieval should be done via :
git clone git+ssh://yourforgelogin@scm.gforge.inria.fr/gitroot/gemoc-dev/gemoc-dev.git

4.1.2 Naming conventions

The naming convention for a component contributions plug-in projects is the following :
<qualified name >.<component contribution name>.<contribution part>

the qualified name depends to the kind of knowledge of the contributions :

- New knowledge : the qualified name shall be **org** and the component contribution name shall be **gemoc** : *org.gemoc* (*followed by the contribution part*)
- Appropriate knowledge : the qualified name shall be the one of partners

4.1.3 Project organization

4.2 Continuous Integration

The GEMOC Studio is built with the continuous integration at the link below :
`https://ci.inria.fr/gemoc`

4.3 Update site

All Components of the GEMOC Studio shall be provided by an update site.

4.4 GEMOC Studio

GEMOC Studio should be used to design its plug-ins which will design all along the project.
It currently contains following tools :

- Mylyn
- M2e (maven integration for eclipse)
- ComponentDSL
- SVN/Git

- XML editor
- some EMF tools
- Obeo designer
- Kermeta2
- Papyrus
- xText

It is downloading at link below :
`https://ci.inria.fr/gemoc/job/org.gemoc.gemoc_studio.root/`

4.5 Required configuration

To launch GEMOC Studio, these components should setup on your computer :

- JDK JAVA 1.6 minimum version, JDK should download to link below :
  `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
- Scala IDE 2.9 for juno eclipse edition, it should download to link below :
  `http://download.scala-ide.org/sdk/e38/scala29/stable/site`

4.6 Life-cycle of new logical component

4.6.1 Presentation of an average component life-cycle

A new component should have some validation steps :

- Compilation of the project
- Build with Maven
- The new component shaould have a quickstart documentation
- The new component should contain an example presenting its functionalities

4.6.2 Minimum requirement for a component

A component shall be integrated to the GEMOC studio if the component :

- is referenced in the GEMOC architecture model
- is a set of eclipse plugins/OSGI bundles
- has one feature referencing all plugins/bundles
- compiles using a maven script
- is deployed as a P2 repository (on Jenkins server or external URL)
- exposes a minimal documentation as Eclipse help (Quick start)
- offers a package sample (accessible through File>New>Sample in eclipse)