



1

Grant Agreement: 287829

2

3 Comprehensive Modelling for Advanced Systems of Systems

3

4 C O M P A S S

4

## 5 **Simulator/Animator Design Document**

5

6 Technical Note Number: DXX

6

7 Version: 0.1

7

8 Date: Month Year

8

9 Public Document

9

10 <http://www.compass-research.eu>

10

<sup>11</sup> **Contributors:**

<sup>12</sup> Anders Kaelis Malmos, AU

<sup>13</sup> **Editors:**

<sup>14</sup> Peter Gorm Larsen, AU

<sup>15</sup> **Reviewers:**

<sup>16</sup> **Document History**<sup>17</sup>

Ver	Date	Author	Description
0.1	25-04-2013	Anders Kaels Malmos	Initial document version

## **Abstract**

This document describes the overall design of the CML simulator/animator and provides an overview of the code structure targeting developers.

21	<b>Contents</b>	
22	<b>1 Preface</b>	<b>6</b>
23	<b>2 Overall Structure</b>	<b>6</b>
24	2.1 The Core Structure . . . . .	6
25	2.2 The IDE Structure . . . . .	7
26	<b>3 Simulation/Animation</b>	<b>7</b>
27	3.1 Static Structure . . . . .	7
28	3.2 Dynamic Structure . . . . .	10

## 1 Preface

This document is targeted at developers and describes the overall structure and design of the CML simulator, it is not a detailed description of each component. This kind of documentation is done in Javadoc and can be generated automatically from the code.

## 2 Overall Structure

This section describes the overall source code structure of the CML interpreter. At the top level the code can be split up in two separate components:

**Core component** Implements the operational semantics that are defined in [?] and is located in the java package named *eu.compassresearch.core.interpreter*

**IDE component** Exposes the core component to the Eclipse framework as an integrated debugger. It is located in the *eu.compassresearch.ide.cml.interpreter\_plugin* package.

Each of these components will be described in further detail in the following sections.

### 2.1 The Core Structure

The following two packages defines the top level structure of the core:

**eu.compassresearch.core.interpreter** This package contains all the classes and interfaces that defines the core functionality of the interpreter.

**eu.compassresearch.core.interpreter.api** This package contains all the public classes and interfaces that defines the API of the interpreter.

The reason for this top level structure is to encapsulate all the classes and interfaces that makes up the core functionality of the interpreter and only expose the classes and interfaces that are needed to utilize it without knowing the details. This provides a clean separation between the implementation and the public interface.

The *eu.compassresearch.core.interpreter* package are split into several folders, each representing a different logical component. The following folders are

58 present  
 59 **cml**  
 60 **visitors**  
 61 **util**  
 62 **debug**  
 63 ...

## 64 2.2 The IDE Structure

## 65 3 Simulation/Animation

66 This section describes the static and dynamic structure of the components  
 67 involved in simulating/animating a CML model.

### 68 3.1 Static Structure

69 The top level interface of the interpreter is depicted in figure 1, followed by  
 a short description of each the depicted components.

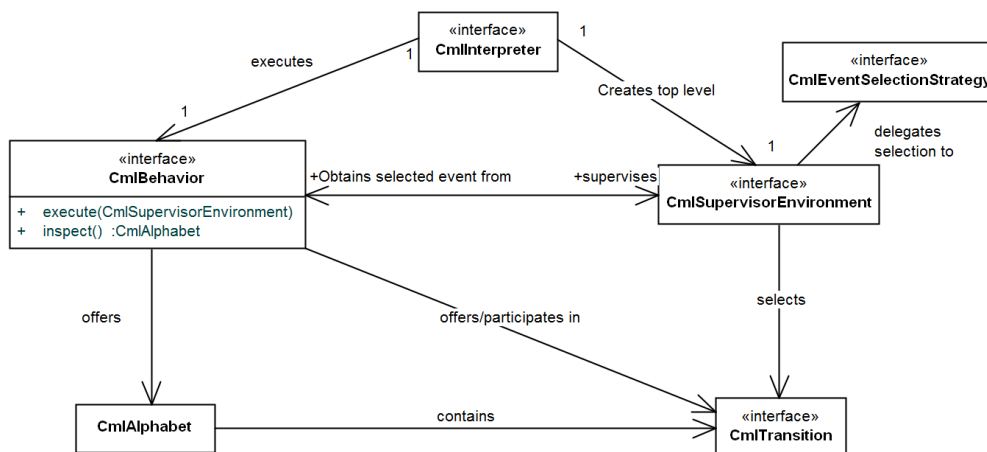


Figure 1: The high level classes and interfaces of the interpreter core component

70

71 **CmlInterpreter** The main interface exposed by the interpreter component.  
 72 This interface has the overall responsibility of interpreting. It exposes  
 73 methods to execute, listen on interpreter events and get the current  
 74 state of the interpreter. It is implemented by the **VanillaCmlInter-**  
 75 **preter** class.

76 **CmlBehaviour** Interface that represents a behaviour specified by either a  
 77 CML process or action. It exposes two methods: *inspect* which cal-  
 78 culates the immediate set of possible transitions that the current be-  
 79 haviour allows and *execute* which takes one of the possible transitions  
 80 determined by the supervisor. A specific behaviour can for instance  
 81 be the prefix action “a -¿ P”, where the only possible transition is to  
 82 interact in the a event. in any

83 **CmlSupervisorEnvironment** Interface with the responsibility of acting as  
 84 the supervisor environment for CML processes and actions. A super-  
 85 visor environment selects and exposes the next transition/event that  
 86 should occur to its pupils (All the CmlBehaviors under its supervi-  
 87 sion). It also resolves possible backtracking issues which may occur in  
 88 the internal choice operator.

89 **CmlEventSelectionStrategy** This interface has the responsibility of choos-  
 90 ing an event from a given CmlAlphabet. This responsibility is delegated  
 91 by the CmlSupervisorEnvironment interface.

92 **CmlTransition** Interface that represents any kind of transition that a Cml-  
 93 Behavior can make. This structure will be described in more detail in  
 94 section 3.1.1.

95 **CmlAlphabet** This class is a set of CmlTransitions. It exposes convenient  
 96 methods for manipulating the set.

97 To gain a better understanding of figure 1 a few things needs mentioning.  
 98 First of all any CML model (at least for now) has a top level Process. Be-  
 99 cause of this, the interpreter need only to interact with the top level CmlBe-  
 100 haviour instance. This explains the one-to-one correspondence between the  
 101 CmlInterpreter and the CMLBehaviour. However, the behavior of top level  
 102 CmlBehaviour is determined by the binary tree of CmlBehaviour instances  
 103 that itself and it's child behaviours defines. So in effect the CmlInterpreter  
 104 controls every transition that any CmlBehaviour makes through the top level  
 105 behaviour.



### 3.1.1 Transition/Event Structure

As described in the previous section a CML model is represented by a binary tree of CmlBehaviour instances and each of these has a set of possible transitions that they can make. A class diagram of all the classes and interfaces that makes up transitions are shown in figure 2, followed by a description of each of the elements.

A transition can be either observable or silent. An observable transition occurs either when time passes or a communication/synchronization takes place on a channel. All of these transitions are captured in the ObservableEvent interface. A silent transitions is captured by the SilentTransition interface and can either mark the occurrence of a hidden event or an internal transition.

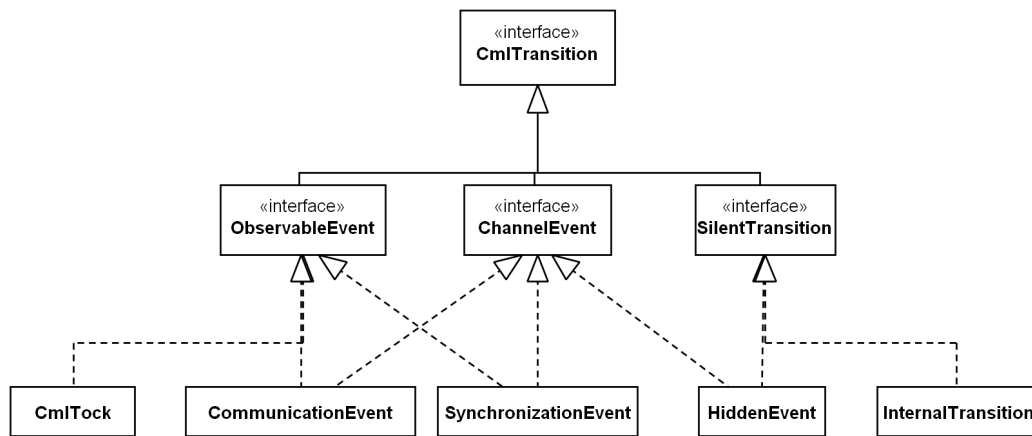


Figure 2: The classes and interfaces that defines transitions/events

**CmlTransition** Represents any possible transition.

**ObservableEvent** This represents any observable transition.

**ChannelEvent** This represents any event that occurs on a channel.

**CmlTock** This represents a Tock event marking the passage of a time unit.

**CommunicationEvent** This represents a communication event on a specific channel and carries a value to be communicated.

**SynchronizationEvent** This represents a synchronization event on a specific channel and carries no value.

**SilentTransition** This represents any non-observable transition.

127 **HiddenEvent** This represents an observable event that has been hidden by  
 128 the hiding operator.

129 **InternalTransition** This represents any transition that are internal to a  
 130 process, like assignemnt, the invocation of a method and etc.

### 131 3.1.2 Action/Process Structure

132 Actions and processes are both represented by the CmlBehaviour interface.

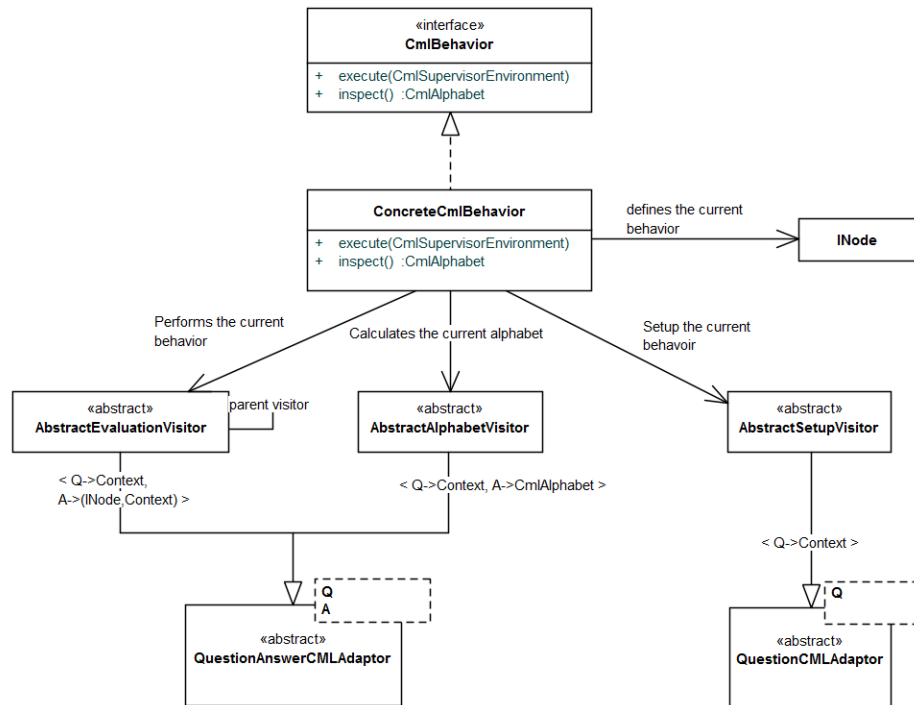


Figure 3: hej

133

## 134 3.2 Dynamic Structure

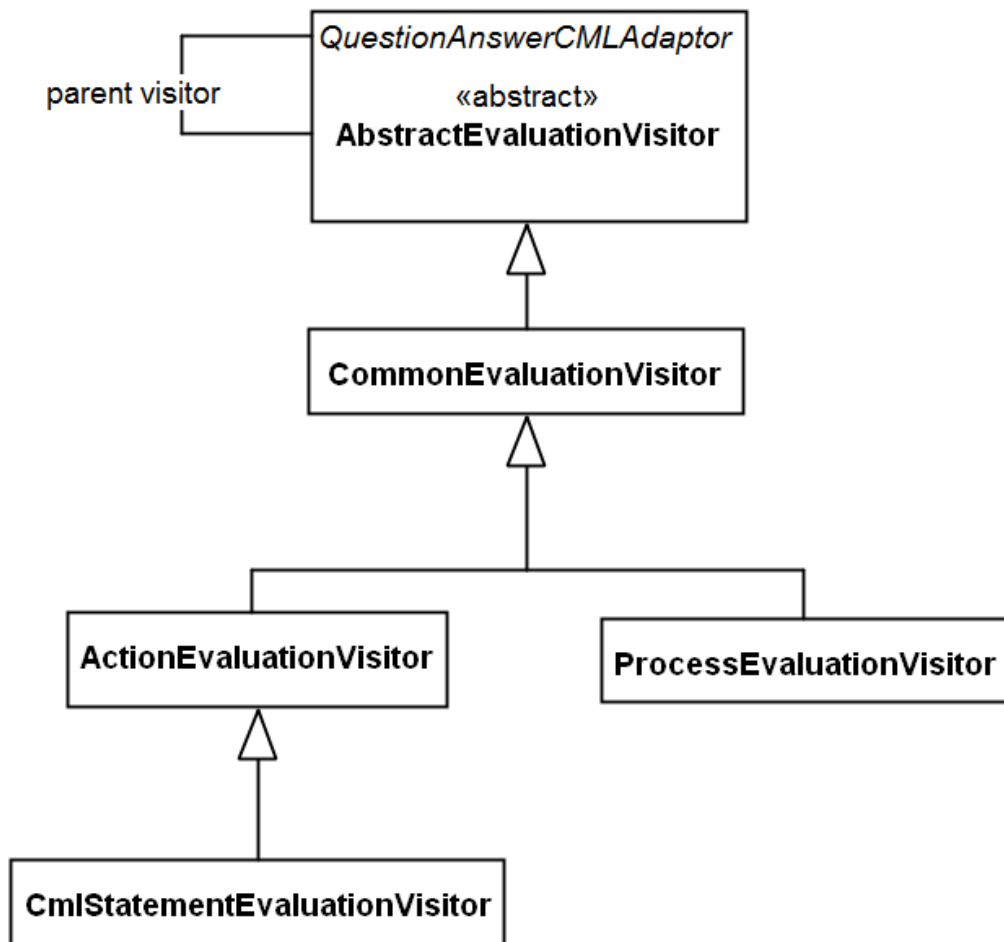


Figure 4: Visitor structure