SEVENTH FRAMEWORK
PROGRAMME

Grant Agreement: 287829

Comprehensive Modelling for Advanced Systems of Systems

COMPASS

Simulator/Animator Design Document

Technical Note Number: DXX

Version: 0.1

Date: Month Year

Public Document

http://www.compass-research.eu

Contributors:

Anders Kaels Malmos, AU

Editors:

Peter Gorm Larsen, AU

Reviewers:

Document History

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.

Contents

1 Preface

This document is targeted at developers and describes the overall strucure 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 design philosophy of the top-level structure is to encapsulate all the classes and interfaces that makes up the implementation of the core functionality and only expose those that are needed to utilize the interpreter. This provides a clean separation between the implementation and interface and makes it clear for both the users, which not necessarily wants to know about the implementation details, and developers which parts they need to work with.

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

eu.compassresearch.core.interpreter.api This package and sub-packages contains all the public classes and interfaces that defines the API of the interpreter. This package includes the main interpreter interface **CmlInterpreter** along with additional interfaces. The api sub-packages groups the rest of the API classes and interfaces according to the responsibility they have.

eu.compassresearch.core.interpreter.api.behaviour This package contains all the components that define any CML behavior. A CML behaviour is either an observable event like a channel synchronization or a internal event like a change of state. The main interface is **CmlBehaviour**.

eu.compassresearch.core.interpreter.api.events This package contains all the public components that enable users of the interpreter to listen on event from both **CmlIntepreter** and **CmlBehaviour** instances.

eu.compassresearch.core.interpreter.api.transitions This package contains all the possible types of transitions that a **CmlBehaviour** instance can make. This will be explained in more detail in section **??**.

eu.compassresearch.core.interpreter.api.values This package contains all the values used in the CML interpreter. Values are used to represent the the result of an expression or the current state of a variable.

eu.compassresearch.core.interpreter.debug TBD

eu.compassresearch.core.interpreter.utility The utility packages contains components that generally reusable classes and interfaces.

eu.compassresearch.core.interpreter.utility.events This package contains components helps to implement the Observer pattern.

eu.compassresearch.core.interpreter.utility.messaging This package contains components to pass message along a stream.

eu.compassresearch.core.interpreter This package contains all the classes and interfaces that defines the core functionality of the interpreter. The important class for any user of the interpreter is the **VanillaInterpreteFactory** that creates **CmlInterpreter** instances.

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 present

cml

visitors

util

debug

...

2.2 The IDE Structure

3 Simulation/Animation

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

3.1 Static Structure

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

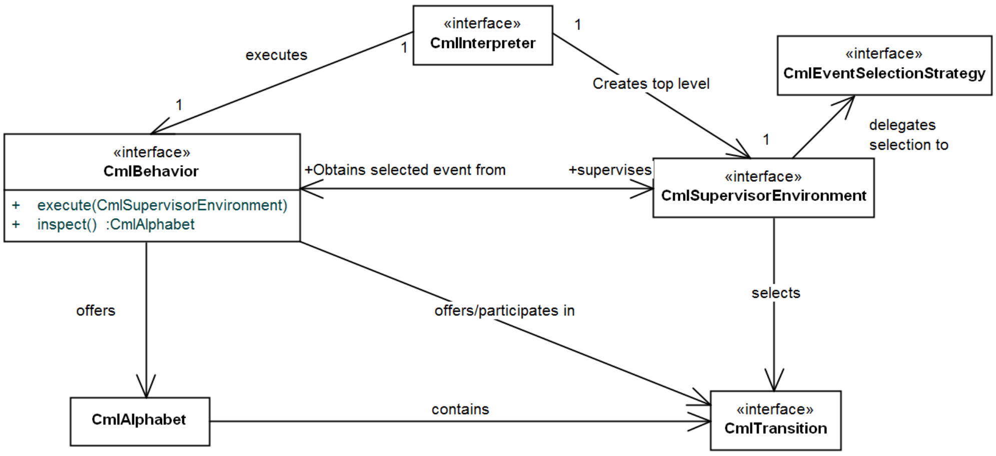

Figure 1: The high level classes and interfaces of the interpreter core component

CmlInterpreter The main interface exposed by the interpreter component. This interface has the overall responsibility of interpreting. It exposes methods to execute, listen on interpreter events and get the current state of the interpreter. It is implemented by the **VanillaCmlInterpreter** class.

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

CmlSupervisorEnvironment Interface with the responsibility of acting as the supervisor environment for CML processes and actions. A supervisor environment selects and exposes the next transition/event that should occur to its pupils (All the CmlBehaviors under its supervision). It also resolves possible backtracking issues which may occur in the internal choice operator.

CmlEventSelectionStrategy This interface has the responsibility of choosing an event from a given CmlAlphabet. This responsibility is delegated by the CmlSupervisorEnvironment interface.

CmlTransition Interface that represents any kind of transition that a CmlBehavior can make. This structure will be described in more detail in section 3.1.1.

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

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

3.1.1 Transition 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.

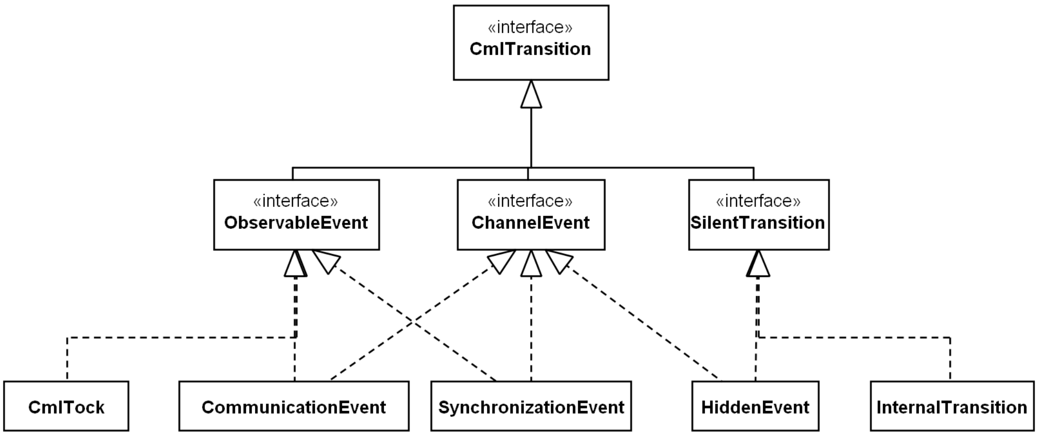

Figure 2: The classes and interfaces that defines transitions/events

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.

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

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

3.1.2 Action/Process Structure

Actions and processes are both represented by the CmlBehaviour interface. A class diagram of the important classes that implements this interface is shown in figure 3

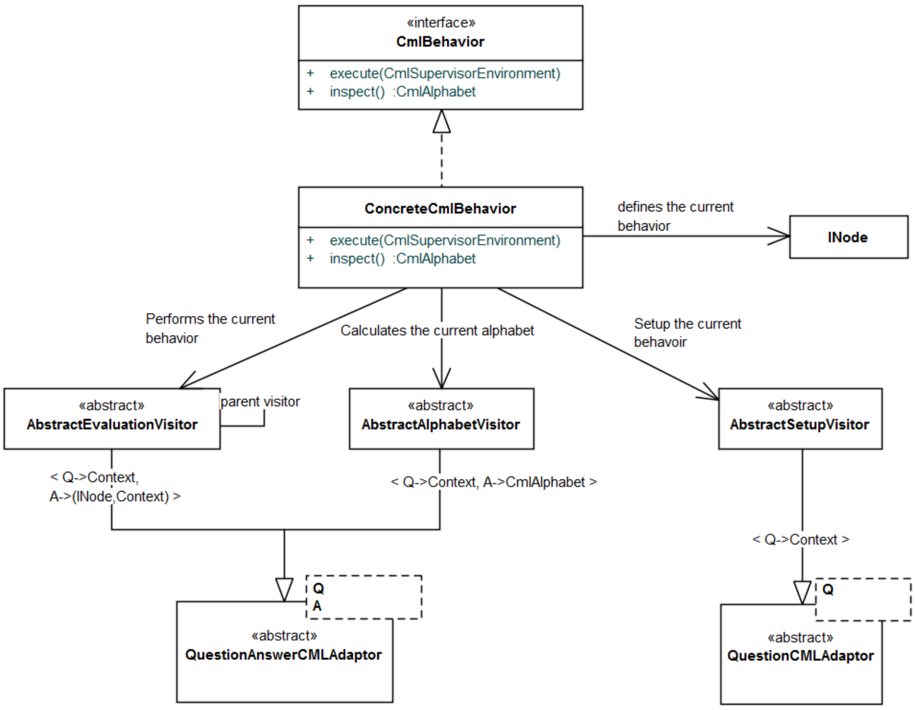
«interface»
CmlBehavior
+ execute(CmlSupervisorEnvironment)
+ inspect() :CmlAlphabet
ConcreteCmlBehavior
+ execute(CmlSupervisorEnvironment)
+ inspect() :CmlAlphabet
defines the current behavior
INode
Performs the current behavior
Calculates the current alphabet
Setup the current behavoir
«abstract»
AbstractEvaluationVisitor
parent visitor
«abstract»
AbstractAlphabetVisitor
«abstract»
AbstractSetupVisitor
< Q->Context, A->(INode,Context) >
< Q->Context, A->CmlAlphabet >
< Q->Context >
Q
A
«abstract»
QuestionAnswerCMLAdaptor
Q
«abstract»
QuestionCMLAdaptor

Figure 3: The implementing classes of the CmlBehavior interface

As shown the **ConcreteCmlBehavior** is the implementing class of the CmlBehavior interface. However, it delegates a large part of its responsibility to other classes. The actual behavior of a ConcreteCmlBehavior instance is decided by its current instance of the INode interface, so when a ConcreteCmlBehavior instance is created a INode instance must be given. The INode interface is implemented by all the CML AST nodes and can therefore be any CML process or action. The actual implementation of the behavior of any process/action is delegated to three different visitors all extending a generated abstract visitor that have the infrastructure to visit any CML AST node.

The following three visitors are used:

AbstractSetupVisitor This has the responsibility of performing any required setup for every behavior. This visitor is invoked whenever a new INode instance is loaded.

AbstractEvaluationVisitor This has the responsibility of performing the actual behavior and is invoked inside the execute method. This involves taking one of the possible transitions.

AbstractAlphabetVisitor This has the responsibility of calculating the alphabet of the current behavior and is invoked in the inspect method.

3.2 Dynamic Structure

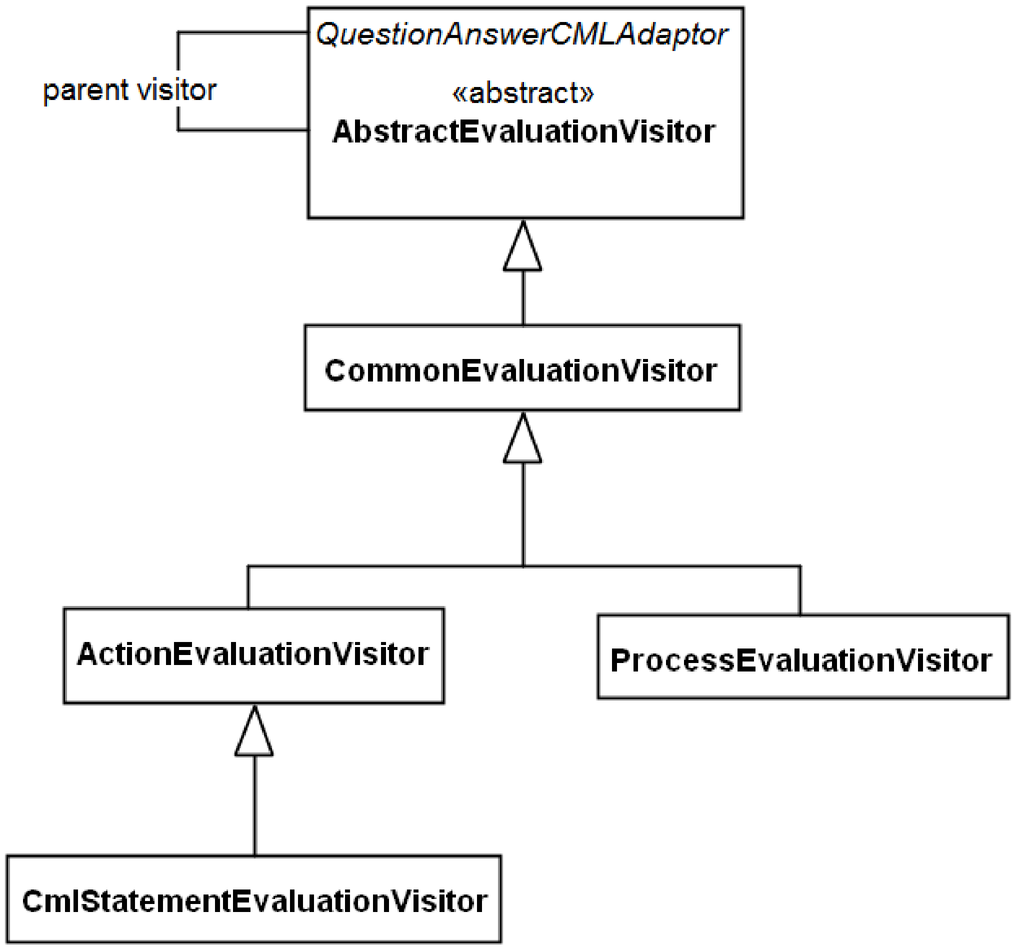

Figure 4: Visitor structure