INtegrated TOol chain for model-based design of CPSs

INTO-CPS Maestro Documentation

Version: 0.01

Date:

The INTO-CPS Association

http://into-cps.org

Contributors:

Kenneth Lausdahl, AU
Casper Thule, AU

Editors:

Casper Thule, AU

Document History

Ver	Date	Author	Description
0.01	November 18, 2019	Casper Thule	Initial Version.

Abstract

TBD

Contents

1 Introduction **6**
1.1 TO BE DONE 7

2 Environment **7**

A List of Acronyms **11**

1 Introduction

Maestro is a framework built for orchestrating co-simulations based on the Functional Mock-Up Interface 2.0 standard for co-simulation.

The framework is divided into two parts: Maestro-Program and Maestro-Runtime.

Maestro-Program concerns the specification of a co-simulation. Such a specification is referred to as Maestro-ProgramSpecification, or ProgramSpecification when the context is clear. A ProgramSpecification consists of commands to be carried out by the Maestro-Runtime. In order to create a ProgramSpecification, Maestro-Program employs plugins that each provides fragments of the ProgramSpecification, which Maestro-Program then merges.

Maestro-Runtime concerns the execution of a ProgramSpecification.

The flow of conducting a co-simulation using Maestro is depicted in fig. 1. The following paragraphs describes the content of the figure. Initially, some terminology and definitions are presented followed by a description of the consecutive behavior.

Environment An environment is data and information related to the co-simulation. For example, the FMUs to employ in a given co-simulation or the dependencies between the variables of the FMUs for a given co-simulation.

Program Environment Terminology for Environment being used in context of Maestro-Program. The terminology only applies to naming for descriptive purposes.

Runtime Environment Terminology for Environment being used in context of Maestro-Runtime. For example, a Program Environment passed to Maestro-Runtime becomes a Runtime Environment. The terminology only applies to naming for descriptive purposes.

Root Environment Terminology for the initial Program Environment. The Root Environment typically consists of the FMUs to use in a co-simulation and values for FMU parameters. The terminology only applies to naming for descriptive purposes.

ProgramSpecification A ProgramSpecification is a complete specification of a co-simulation to be carried out. A non-complete specification is referred to as a ProgramSpecification Fragment.

ProgramSpecification Fragment A ProgramSpecification Fragment is part of a ProgramSpecification.

Plugin A plugin can create a ProgramSpecification Fragment and/or add information to the environment. An example of environment information that a plugin can add is the dependencies between the variables of the FMUs. An example of a ProgramSpecification Fragment that a plugin can create is the necessary commands to perform initialisation of the FMUs.

The Activator in fig. 1 is the entity (person or tool) that launches a co-simulation. The Activator shall provide a Root Environment, see TODO, and a configuration of the plugins, see TODO.

Maestro-Program invokes the plugins according to the plugin configuration. This is demonstrated in fig. 1 where Plugin 1 receives the Root Environment, and creates a new Environment, Environment 1, and ProgramSpecification Fragment 1. This new environment is passed to Plugin 2, which creates Environment 2 and ProgramSpecification Fragment 2. Finally, Plugin N represents that this process can continue for several plugins. At the end of this process, Maestro-Program will have assembled a ProgramSpecification based on the ProgramSpecification Fragments created by the plugins.

Maestro-Runtime executes the ProgramSpecification and can utilise the Runtime Environment. In cases where Maestro-Runtime require additional information on how to continue, it will query a given plugin for such information. An example of such a case is debugging.

1.1 TO BE DONE

- Validation of ProgramSpecification

2 Environment

The Environment contains information available to Maestro-Program, plugins and Maestro-Runtime. These entities can also update the Environment and thus create a new Environment.

Some information is essential in FMI co-simulation, i.e. FMUs. For this reason, the Root Environment comes with the following entries (Types are described afterwards):

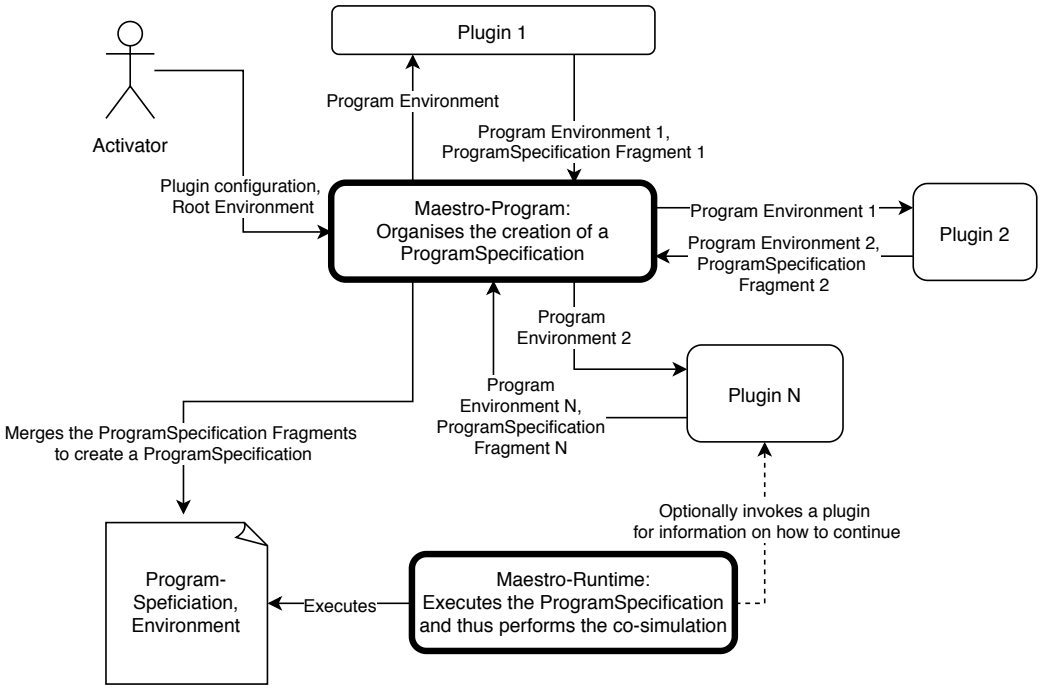

Figure 1: Conducting a co-simulation with Maestro-Program and Maestro-Runtime

RawFMUs The raw FMU information. Perhaps from a UI. Type: Map[FmuKey, FmuPath]

RawInstances The raw instance information. Perhaps from a UI. Type: Map[FmuKey, List[InstanceKey]]

RawConnections The raw connections information. Perhaps from a UI. Type: Map[FmuKey, FmuInstance, ScalarVariableName, Set[FmuKey, FmuInstance, ScalarVariableName]]

RawParameters The raw parameters information. Perhaps from a UI. Type: Map[ParameterKey, Value] and Map[FMU, Map[Instance, Map[ScalarVariableName, ParameterKey]]]

FMUsWithInstances Enriches RawFMUs with ModelDescription information and connects FMUs to their respective instances. Type: Map[Fmu, Set[Instance]]

Connections The connections based on FMUsWithInstances and variables from the corresponding ModelDescription files. Type: Set[Connection]

SortedDependantVariables An list that describes the order of setting and

getting dependant scalar variables according to internal and external dependencies. Type: List[ScalarVariableID]

Custom This entry can be used freely. Type: Map[CustomDataKey, Any].

Types:

FmuKey Unique identifier for a given FMU

FmuPath URI the a given FMU

InstanceKey Unique string identifier for a given instance

ScalarVariableName String with the name of a given scalar variable. TODO: Perhaps valuereference, perhaps both?

Fmu Data object with FmuKey, FmuPath and ModelDescription.

Instance Data Object with parent FMU and InstanceKey

ModelDescription The parsed model description.

Connection Data Object with from : ScalarVariableID and to : Set[ScalarVariableID]

ScalarVariableID Data object that consists of FmuKey, InstanceKey, ScalarVariableValueReference

CustomDataKey A unique identifier for some custom data.

Any Represents the value of custom data. TODO: Represent as Sum Type with Any, JSON X, String X, Byte X?

References

A List of Acronyms

XML Extensible Markup Language