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

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. These entities and the flow of a co-simulation using MaestroV2 is depicted in fig. 1 and described in this section.

Maestro-Program concerns the program that specifies how to conduct a co-simulation. A Program consists of commands to be carried out by Maestro-Runtime. In order to create a Program, Maestro-Program employs plugins. The creation of a Program is referred to as the Program Phase.

Maestro-Runtime concerns the execution of a Program.

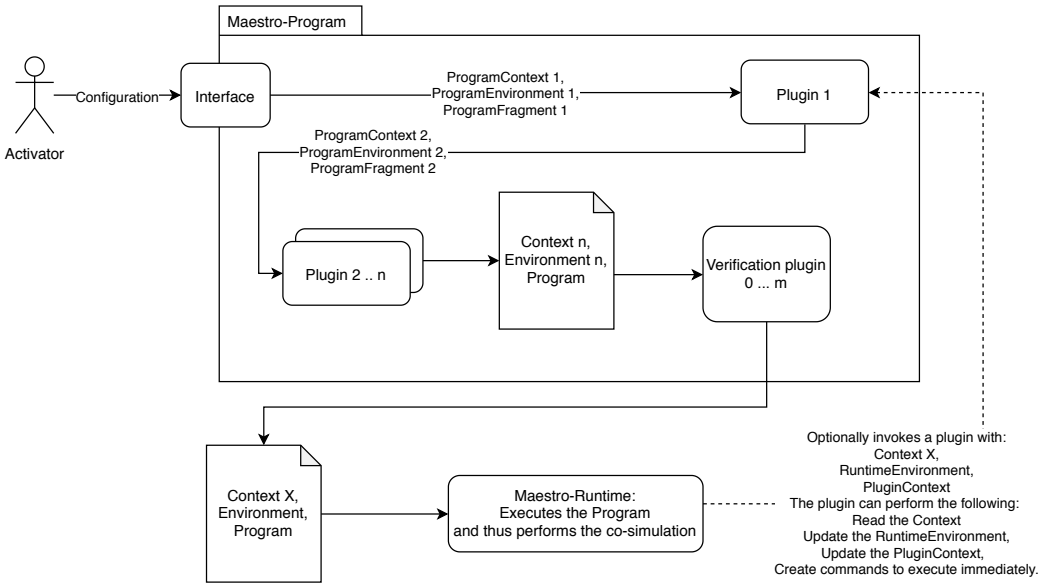

Figure 1: Conducting a co-simulation with Maestro-Program and Maestro-Runtime

The following paragraphs describes the content of the figure. Initially, some terminology and definitions are presented followed by a description of the consecutive behavior.

Context Context 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.

Environment The environment is information related to variables. For example, type information or the value of a given variable.

ProgramContext Terminology for the Context used in relation to Maestro-Program.

ProgramEnvironment Terminology for the Environment used in relation to Maestro-Program.

RuntimeContext Terminology for Context used in relation to Maestro-Runtime.

RuntimeEnvironment Terminology for the Environment used in relation to Maestro-Runtime.

PluginContext A plugin specific Context. Maestro-Runtime does not know its content and the code to process it must be provided by the respective plugin.

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.

Program A Program is a complete in the sense that it can be passed to Maestro-Runtime for execution as opposed to a ProgramFragment described below.

Program Fragment A Program Fragment is part of a Program.

Plugin In the ProgramA plugin can read/update the Program Fragment, and/or read/update the ProgramContext and/or read/update the ProgramEnvironment. 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 add information to the Environment by creating a new Environment, which shall contain data from the old environment that has not been updated.

2.1 Environment Structure

This section describes the structure of the Root Environment and thereby what is supported natively. The reason for natively supporting some data is that it is considered essential for FMI co-simulation. First, the entries are described in an overall format. Afterwards, the types mentioned in the overall description. Note, that not all entries are populated by Maestro-Program prior to employing plugins. Some of these entries will be populated by plugins. TODO: Clearly describe which entries Maestro-Program populates.

Overall Structure

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 String: Unique identifier for a given FMU

FmuPath URI: Location of the FMU

InstanceKey String: Unique identifier for a given instance

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

Fmu Data Object with FmuKey, FmuPath and ModelDescription.

Instance Data Object that consists of parent FMU and InstanceKey

ModelDescription The parsed model description.

Connection Data Object with from of type ScalarVariableID and to of type Set[ScalarVariableID]

ScalarVariableID Data Object that consists of FmuKey, InstanceKey, ScalarVariableValueReference

CustomDataKey String: Unique identifier for some custom data.

Any Represents the value associated with a CustomDataKey. TODO: Represent as Sum Type: Any (Any) | JSON (String) | Text (String) | Byte (Array[Byte]) | ?

References

A List of Acronyms

XML Extensible Markup Language