Mod Code Generator Introduction

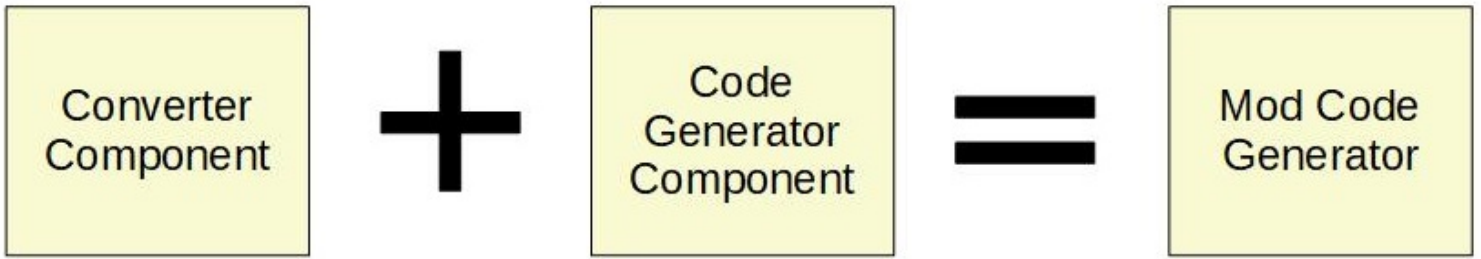
What is the Mod Code Generator (MCG)?

- The MCG is a code generator tool
- It is written in Python https://www.python.org/
- The MCG works in conjunction with Modelio environment https://www.modelio.org/
- The MCG repository is available under https://github.com/deckamil/mod-code-generator/

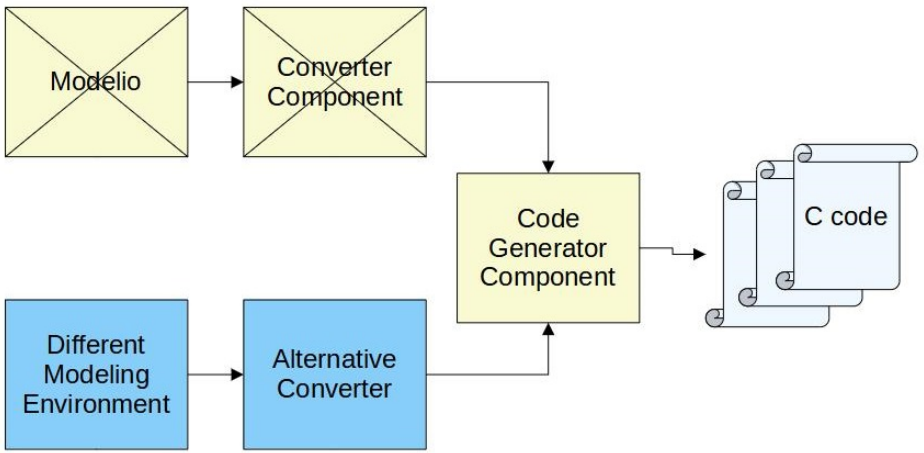
The MCG is divided into two components:

Converter Component (CC), responsible for conversion of a model created within Modelio environment into a configuration file.

Code Generator Component (CGC), responsible for C code generation from the configuration file.



Why MCG CC and CGC are two separate tools?



The MCG CC is suited to work in conjunction with the Modelio environment and its main task is to generate the configuration file for the MCG CGC.

A tool separation gives possibility to replace the MCG CC with alternative converter tool and generate code from different modeling environment than the Modelio, as long as the alternative converter tool will generate the configuration file in desired format and syntax for the MCG CGC.

If preferred, the MCG CGC can be used in a stand-alone mode. An end user can write the configuration file in desired format and syntax and input it into the MCG CGC for code generation.

The Modelio environment

What role does the Modelio environment play?

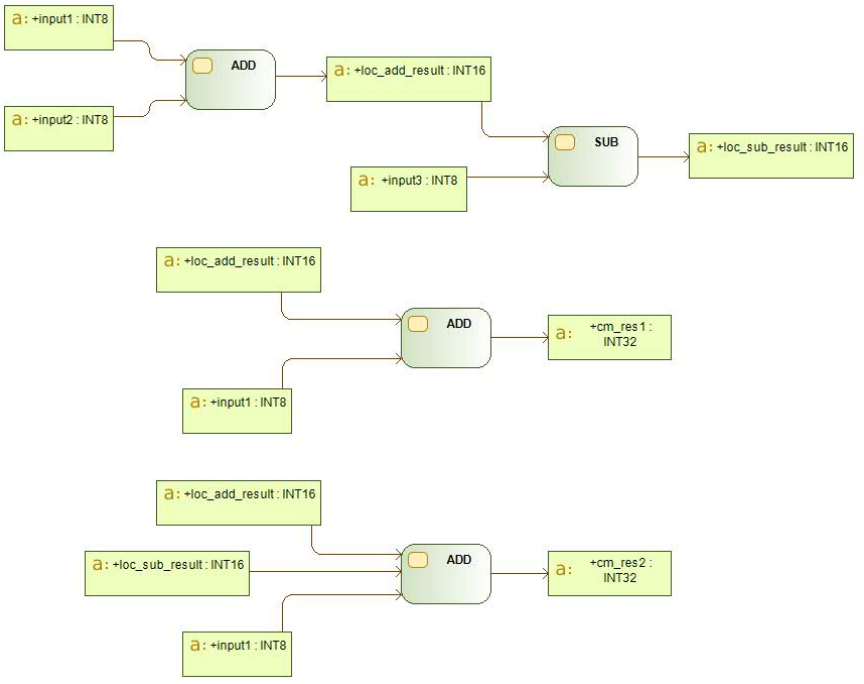
- The MCG CC and MCG CGC are command line tools
- The Modelio provides modeling environment and graphical user interface for the MCG
- It is stand-alone program and it is not a part of the MCG
- See list of features under https://www.modelio.org/about-modelio/features.html/

How the Modelio environment is used?

The Modelio provides modeling environment and graphical user interface standing between an end user and the MCG. The modeling environment allows to build graphical representation of a model and express its data flow and data interactions.

The model data is saved in form of .exml files, which further provide input information about model elements and diagram connections to the MCG CC.

```
<LINK relation="RepresentedAttribute">
  <REFOBJ>
    <ID name="input1" mc="Standard.Attribute" uid="aa4605b2-2ee2-4563-97cd-9d230eb5f6fc"/>
  </REFOBJ>
</LINK>
<COMP relation="Outgoing">
  <OBJECT>
    <ID name="ObjectFlow" mc="Standard.ObjectFlow" uid="8d3a9bb7-432d-41db-a272-4f4399255e41"/>
    <ATTRIBUTES>
      <ATT name="TransformationBehavior"></ATT>
      <ATT name="SelectionBehavior"></ATT>
      <ATT name="IsMultiCast">false</ATT>
      <ATT name="IsMultiReceive">false</ATT>
      <ATT name="Effect">ReadFlow</ATT>
      <ATT name="Guard"></ATT>
      <ATT name="Weight"><![CDATA[1]]></ATT>
      <ATT name="Name"><![CDATA[ObjectFlow]]></ATT>
      <ATT name="status">1970354901745664</ATT>
    </ATTRIBUTES>
    <DEPENDENCIES>
      <LINK relation="Target">
        <REFOBJ>
          <ID name="ADD" mc="Standard.OpaqueAction" uid="7ea36f0f-22c9-404f-adbb-1b770e2c188c"/>
```



Model package and component elements

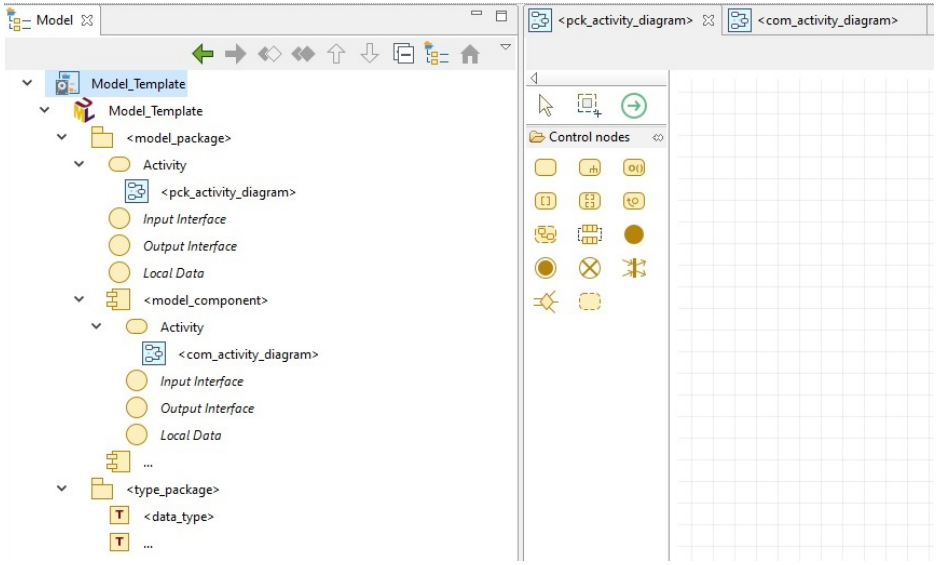
A model created within Modelio environment should be composed from package and component elements.

Both component and package should have interface definition along with activity diagram, which will be converted later from .exml format into configuration file.

Component activity diagram is used to define data interactions (like arithmetic operations) and is treated as a separate C module.

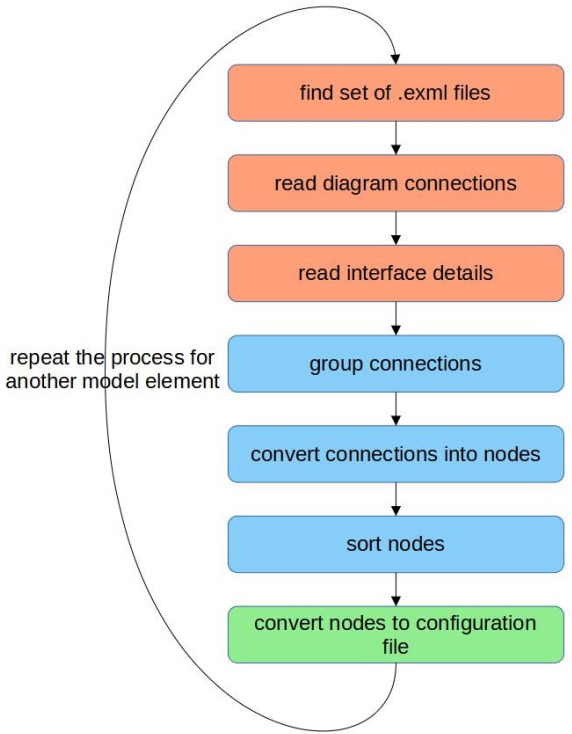
Package activity diagram (optional) is used to define data flow between components. If connections between components are not defined under package activity diagram, then components integration at C level has to be provided manually after code generation.

Type package is used only to define allowed data types for model interfaces.



The MCG Converter Component (MCG CC)

The MCG CC workflow

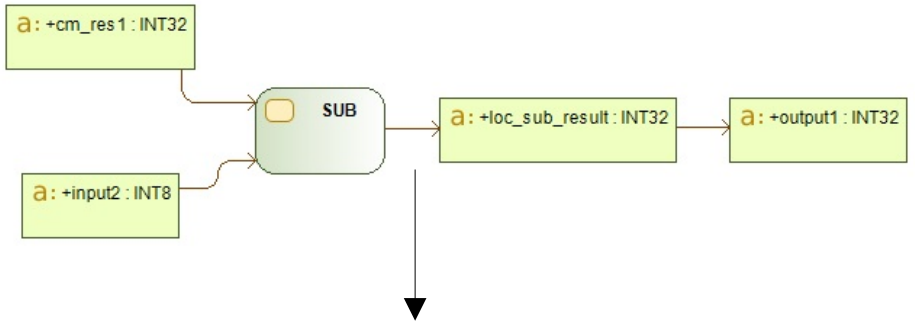


The entire MCG CC workflow is a 3-step process:

1. READING – find set of .exml files, which describe one model element, then read from them information about connections between diagram elements and interface details
2. SORTING – group all connections per diagram interaction type, then convert connections into nodes and sort nodes at the end of that process step
3. CONVERSION – convert nodes into the configuration file

The entire process is repeated for each model element, which is for each component and package, as long as they have appended activity diagram.

Model connections and nodes



- $SOURCE$: $FIRST$ cm_res1 $TARGET$: SUB 6d076cfa-6ce0-4bd4-bde4-e327457e8691
- $SOURCE$: input2 $TARGET$: SUB 6d076cfa-6ce0-4bd4-bde4-e327457e8691
- $SOURCE$: output1 $TARGET$: $EMPTY$
- $SOURCE$: loc_sub_result $TARGET$: output1
- $SOURCE$: SUB 6d076cfa-6ce0-4bd4-bde4-e327457e8691 $TARGET$: loc_sub_result

The MCG CC searches for set of .exmls files, which describe one model element and then reads from these files information about connections between diagram elements and details of model element interface.

Basing on above details the MCG CC creates list of connections, which describe each connection $SOURCE$ and connection $TARGET$. All connections are then sorted into groups of interactions.

Grouped connections finally are converted into nodes format. A node describes one, specific instance of data interaction (like SUBtraction operation) from activity diagram, along with interaction inputs and output, as exmaple:

$INPUTS$: cm_res1 input2 $INTERACTION$: SUB 6d076cfa-6ce0-4bd4-bde4-e327457e8691 $OUTPUT$: loc_sub_result

How activity diagram nodes are sorted?

- $INPUTS$: input1 input2 $INTERACTION$: ADD 7ea36f0f-22c9-404f-adbb-1b770e2c188c $OUTPUT$: loc_add_result
- $INPUTS$: loc_sub_result input1 $INTERACTION$: ADD 9b1cef49-19f9-47a5-9247-e94b09d8b9a3 $OUTPUT$: cm_res1
- $INPUTS$: loc_add_result loc_sub_result input1 $INTERACTION$: ADD 8abfc438-1b8d-4771-a445-53855a80ea10 $OUTPUT$: cm_res2
- $INPUTS$: loc_add_result input3 $INTERACTION$: SUB 56cfd233-ec84-405e-8de8-22caf60c369d $OUTPUT$: loc_sub_result

- $INPUTS$: input1 input2 $INTERACTION$: ADD 7ea36f0f-22c9-404f-adbb-1b770e2c188c $OUTPUT$: loc_add_result
- $INPUTS$: loc_add_result input3 $INTERACTION$: SUB 56cfd233-ec84-405e-8de8-22caf60c369d $OUTPUT$: loc_sub_result
- $INPUTS$: loc_sub_result input1 $INTERACTION$: ADD 9b1cef49-19f9-47a5-9247-e94b09d8b9a3 $OUTPUT$: cm_res1
- $INPUTS$: loc_add_result loc_sub_result input1 $INTERACTION$: ADD 8abfc438-1b8d-4771-a445-53855a80ea10 $OUTPUT$: cm_res2

Once nodes are defined, the MCG CC will sort them:

1. At beginning the MCG CC looks for nodes, which use only input interface data (no dependency from local data) and then add such nodes at beginning of sorted nodes list.
2. After that the MCG CC looks for nodes, which use either input interface data or local data computed by previously sorted nodes, then append them to the list.
3. The MCG CC repeats the process from (2) for the rest of unsorted nodes till all are sorted.

Conversion into configuration file

An example of configuration file part with component definition

```
COMPONENT START
COMPONENT SOURCE 48bfdb13-00ff-4337-a0e1-c9c4a7ec221a.exml
COMPONENT NAME Computation_Block
INPUT INTERFACE START
type INT8 name input1
type INT8 name input2
type INT8 name input3
INPUT INTERFACE END
OUTPUT INTERFACE START
type INT32 name cm_res1
type INT32 name cm_res2
OUTPUT INTERFACE END
LOCAL DATA START
type INT16 name loc_add_result
type INT16 name loc_sub_result
LOCAL DATA END
BODY START
COM Action Interaction ADD 7ea36f0f-22c9-404f-adbb-1b770e2c188c
INS loc_add_result = input1 + input2
COM Action Interaction SUB 56cfd233-ec84-405e-8de8-22caf60c369d
INS loc_sub_result = loc_add_result - input3
COM Action Interaction ADD 9b1cef49-19f9-47a5-9247-e94b09d8b9a3
INS cm_res1 = loc_sub_result + input1
COM Action Interaction ADD 8abfc438-1b8d-4771-a445-53855a80ea10
INS cm_res2 = loc_add_result + loc_sub_result + input1
BODY END
COMPONENT END
```

An example of configuration file part with package definition

```
PACKAGE START
PACKAGE SOURCE 4a56b20e-97bd-4806-b1eb-79500f1b201f.exml
PACKAGE NAME Simple_Calc
INPUT INTERFACE START
type INT8 name input1
type INT8 name input2
type INT8 name input3
INPUT INTERFACE END
OUTPUT INTERFACE START
type INT32 name output1
type INT32 name output2
type INT32 name cm_res2
type INT32 name cm_res1
OUTPUT INTERFACE END
LOCAL DATA START
type DATA name com_bl_output
type DATA name lw_side_output
type DATA name hi_side_output
LOCAL DATA END
BODY START
COM Component Interaction Computation_Block e531e463-60f0-49ad-aca8-270c6d97f1d4
INV com_bl_output = Computation_Block (Input Interface)
COM Component Interaction Low_Side_Process 2bbdc80d-c296-431a-8e4e-e5a73359292e
INV lw_side_output = Low_Side_Process (com_bl_output, Input Interface)
COM Component Interaction High_Side_Process 05c7935b-ec8f-4ddb-89f7-b115651b0a9d
INV hi_side_output = High_Side_Process (Input Interface, com_bl_output)
ASI Output Interface = (com_bl_output, lw_side_output, hi_side_output)
BODY END
PACKAGE END
```

Interface data and sorted nodes, which were sourced from .exml files are later converted into configuration file format by the MCG CC.

The configuration file is treated as input data to the MCG CGC process and it contains information about interfaces of model element and set of instructions which represent data interactions from activity diagram.

The MCG Code Generator Component (MCG CGC)