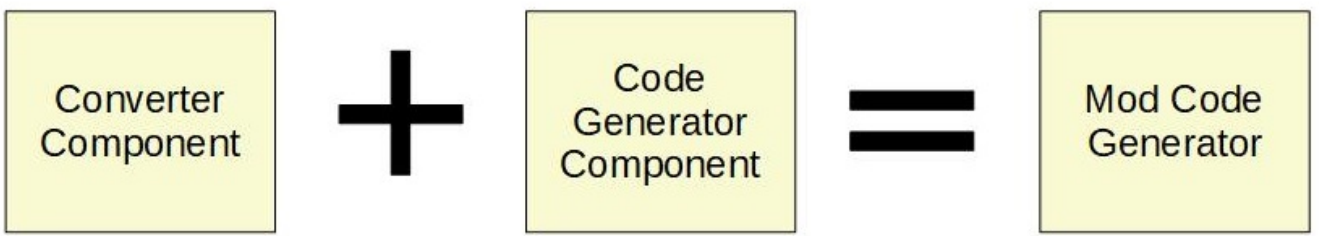
Mod Code Generator Introduction

What is the Mod Code Generator (MCG)?

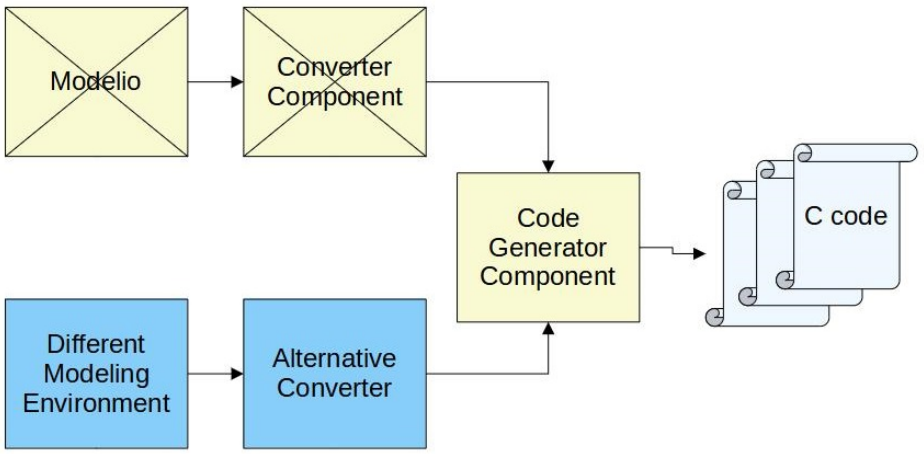
The MCG is a code generator tool. It is written in Python https://www.python.org/. The MCG works in conjunction with Modelio https://www.modelio.org/, that provides modeling environment and graphical user interface used to create a model. The modeling environment is stand-alone program and it is not a part of the MCG, see list of features under https://www.modelio.org/about-modelio/features.html/.

The MCG program consists of two main components (subprograms):

- the Converter Component (CC), which is responsible for conversion of a model create within the modeling environment into configuration file,
- the Code Generator Component (CGC), which is responsible for 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, 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.

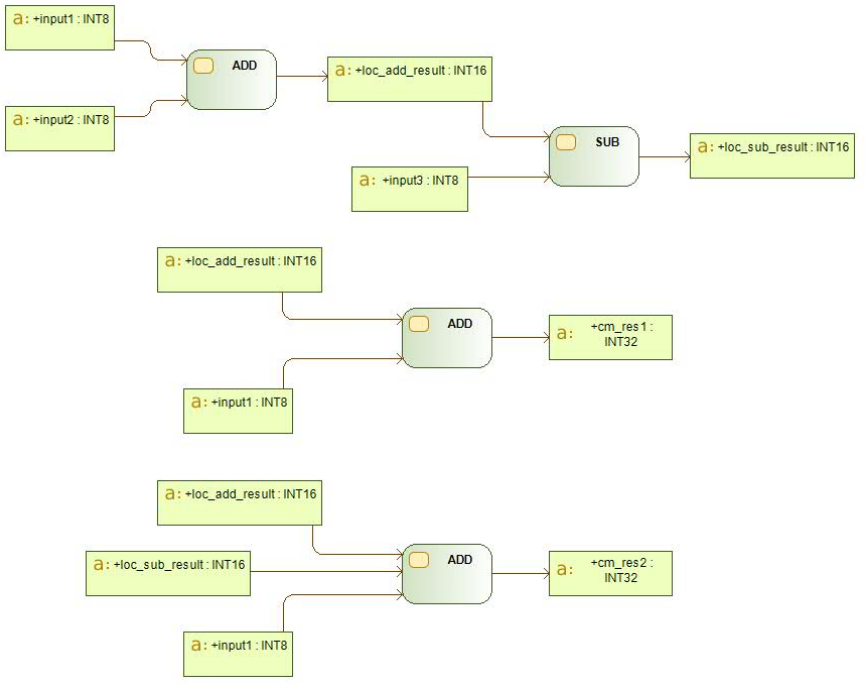
The modeling environment

How the modeling environment is used?

The modeling environment provides graphical user interface standing between an end user and the MCG. The modeling environment allows to build graphical representation of a model and define 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.

```
26        <LINK relation="RepresentedAttribute">
27          <REFOBJ>
28            <ID name="input1" mc="Standard.Attribute" uid="aa4605b2-2ee2-4563-97cd-9d230eb5f6fc"/>
29          </REFOBJ>
30        </LINK>
31        <COMP relation="Outgoing">
32          <OBJECT>
33            <ID name="ObjectFlow" mc="Standard.ObjectFlow" uid="8d3a9bb7-432d-41db-a272-4f4399255e41"/>
34            <ATTRIBUTES>
35              <ATT name="TransformationBehavior"></ATT>
36              <ATT name="SelectionBehavior"></ATT>
37              <ATT name="IsMultiCast">false</ATT>
38              <ATT name="IsMultiReceive">false</ATT>
39              <ATT name="Effect">ReadFlow</ATT>
40              <ATT name="Guard"></ATT>
41              <ATT name="Weight"><![CDATA[1]]></ATT>
42              <ATT name="Name"><![CDATA[ObjectFlow]]></ATT>
43              <ATT name="status">1970354901745664</ATT>
44            </ATTRIBUTES>
45            <DEPENDENCIES>
46              <LINK relation="Target">
47                <REFOBJ>
48                  <ID name="ADD" mc="Standard.OpaqueAction" uid="7ea36f0f-22c9-404f-adbb-1b770e2c188c"/>
```



Model package and component elements

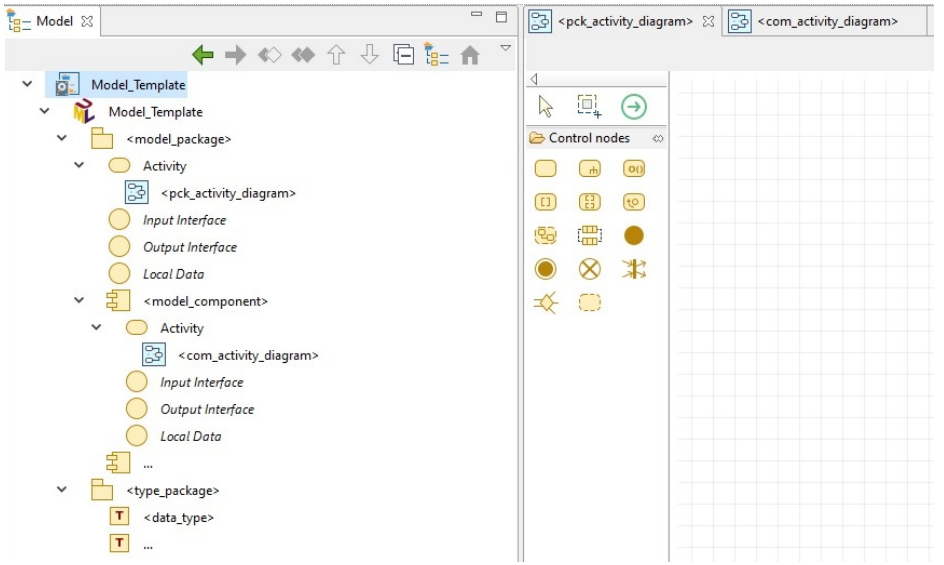
A model should be composed from package and component elements.

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.

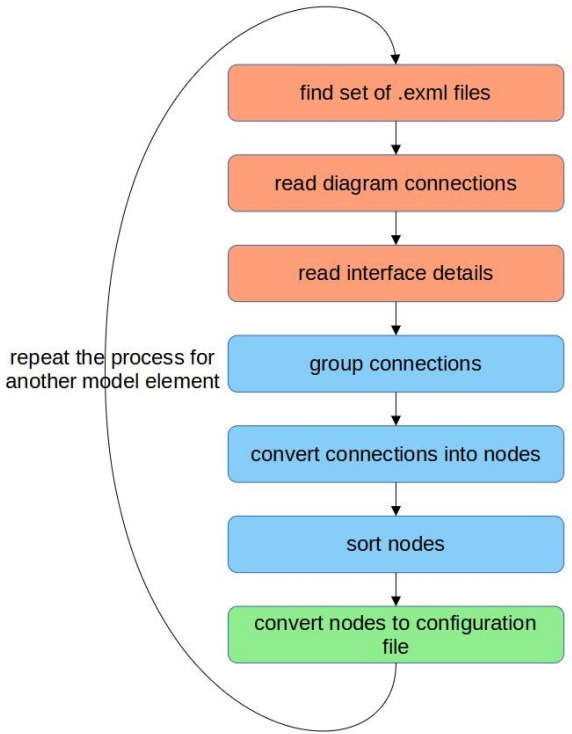
Package activity diagram (optional) is used to define data flow between components. If connections between components are not defined under package activity diagram, then components integration at C level has to be provided manually after code generation.

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



The MCG Converter Component (MCG CC)

The MCG CC workflow

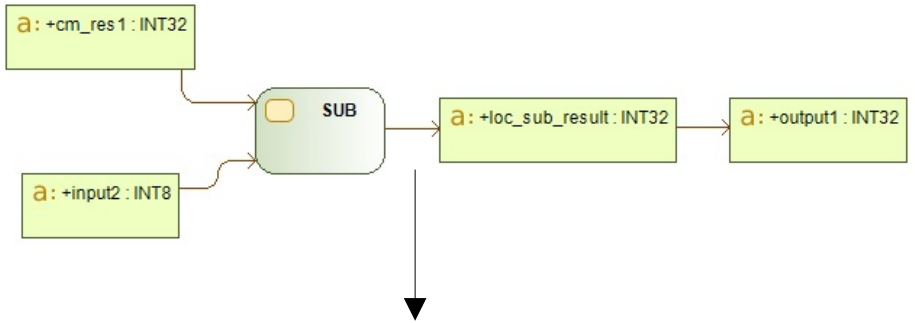


The entire MCG CC workflow is a 3-step process:

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 (for each component and package) that 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 looks 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, that describe each connection's $SOURCE$ and connection's $TARGET$. 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 it adds these 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 appends 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 2248e608-7ae8-414c-b94d-15c69d91ff3c.exml
COMPONENT NAME T_Ratio
COMPONENT INPUT INTERFACE START
type INT16 name temp_diff
type FLOAT32 name temp_compensated
COMPONENT INPUT INTERFACE END
COMPONENT OUTPUT INTERFACE START
type FLOAT32 name temp_ratio
COMPONENT OUTPUT INTERFACE END
COMPONENT LOCAL DATA START
COMPONENT LOCAL DATA END
COMPONENT BODY START
COM Action Interaction DIV 939a2bd1-99d0-42d7-bf22-48a448ac3d42
INS temp_ratio = temp_diff / temp_compensated
COMPONENT BODY END
COMPONENT END
```

An example of configuration file part with package definition

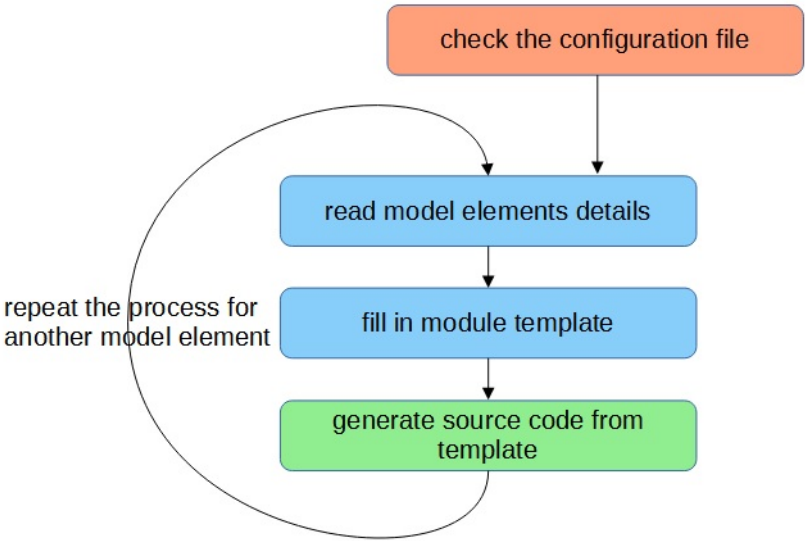
```
PACKAGE START
PACKAGE SOURCE 7ce08d04-fc5c-4521-b8fb-29f92568ea2c.exml
PACKAGE NAME Temperature_Ratio
PACKAGE INPUT INTERFACE START
type INT16 name temp1
type INT16 name temp2
type FLOAT32 name compensation_factor
PACKAGE INPUT INTERFACE END
PACKAGE OUTPUT INTERFACE START
type FLOAT32 name temp_ratio
PACKAGE OUTPUT INTERFACE END
PACKAGE LOCAL DATA START
type DATA name Temp_S
type DATA name Temp_D
type DATA name Temp_C
type DATA name Temp_R
PACKAGE LOCAL DATA END
PACKAGE BODY START
COM Component Interaction T_Diff d6ce217b-9996-4c47-b996-13391a808a07
INV Temp_D = T_Diff (Input Interface)
COM Component Interaction T_Sum e8093495-c9c7-4fc1-b3db-4eaefa794e74
INV Temp_S = T_Sum (Input Interface)
COM Component Interaction T_Comp 207bcfd4-7e51-4b71-8c67-19abfbae30a2
INV Temp_C = T_Comp (Input Interface, Temp_S)
COM Component Interaction T_Ratio 02cc9031-c7f4-487f-941f-6ea71425883a
INV Temp_R = T_Ratio (Temp_D, Temp_C)
ASI Output Interface = (Temp_R)
PACKAGE 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)

The MCG CGC workflow



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

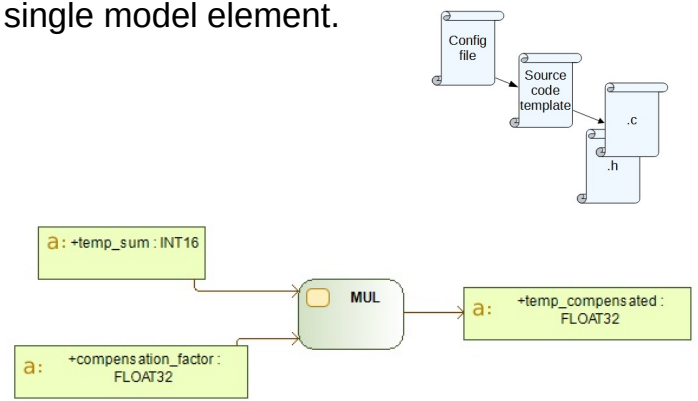
1. CHECKING – check the configuration file that contains information about interfaces of model element and set of instructions which represent data interactions
2. READING – read details of model element from the configuration file and fill in a template of source code item
3. GENERATION – generate source code from the template of source code item

The reading & generation processes are repeated for each model element (for each component and package) until end of the configuration file.

The template and code generation

The MCG CGC reads configuration data of given model element (component, package) and then fills in the template of source code item with data acquired from the configuration file.

Once template is filled with data, the MCG CGC generates from the template both the source code .c and header .h files that represent a single model element.




```c
/*
 *   Generated with Mod Code Generator (MCG) Code Generator Component (CGC)
 *   on 24 Jul 2022, 18:43:10
 *
 *   This is source file of T_Comp module.
 *   The module was generated from file 88df3c99-4250-4c56-9a74-9a3243cc82d8.exml
 */

#include "T_Comp.h"
#include "basic_data_types.h"

// This is definition of module function
T_Comp_output_type T_Comp(T_Comp_input_type *T_Comp_input){

    // Input Interface
    INT16 temp_sum = T_Comp_input->temp_sum;
    FLOAT32 compensation_factor = T_Comp_input->compensation_factor;

    // Local Data

    // Output Interface
    FLOAT32 temp_compensated;
    T_Comp_output_type T_Comp_output;

    // Action Interaction MUL 393b9ba1-3c02-428a-a49c-ba7a9dd4c2bf;
    temp_compensated = temp_sum * compensation_factor;

    // Collect output data
    T_Comp_output.temp_compensated = temp_compensated;

    // Return output data
    return T_Comp_output;
}

/*
 * END OF MODULE
 */
```