**CyMLTx Requirements**

**Use case**

A user provides an autonomous crop model component source code. This component can be defined by a set of simpler independent units (routines, functions, methods) called sequentially. Units can be tested independently of others. By using CyMLTx, the result of this component transformation is a Crop2ML model consisting of:

* Model Specification (Meta-information of units and composite)
* Model Algorithms (Computation part) of units
* Model Initialization for the states variables (if required)
* Auxiliary Functions (if require).

The specification is generated in an XML format validated by the Crop2ML DTDs while Algorithms, Initialization, and Functions are generated in CyML language.

**General Requirements:**

The source code should follow these guidelines (**in bold the high constraints**):

1. Define explicitly the inputs and outputs of each unit of the component.
2. If the platform does not provide a pattern allowing retrieval of the unit components, their inputs, and outputs meta-information, the computation, and initialization parts, it could be done by using CyMLT annotations (Tableau 1).
3. !!! Important: **The invocation of a unit with the same values of inputs should produce the same values of outputs**. This requirement avoids using global variables and a mechanism that preserves local variables or arrays values (maintains its value from one call to the next) in the routine. Likewise, a unit should not depend on any computational state other than its inputs and outputs.
4. Domain classes (composite variables) can be used. It requires explicitly providing meta-information on all the fields of the domain classes. **The names of the fields of different domain classes should be different.**
5. State variables initialization part must depend only on parameters, and exogenous (driven) variables.
6. **Auxiliary functions should be implemented in the file of the unit component calling them**. External utilities are not yet supported.
7. **Make the code case-sensitive**.
8. The main language constructs in the algorithm, initialization, and external functions parts are:
   * Types: integer, real, string, logical, list (dynamic array), array (fixed size)
   * Assignment
   * Branching
   * Looping (for (do), while)
   * Binary operations
   * Usual mathematical functions (trigonometric functions, max, min, log, exp, ...)
   * Use for loops for operations between two arrays or on each item of an array variable

Table: 1. CyMLT annotations

|  |  |
| --- | --- |
| Annotations | Part of source code |
| %%CyML Model Begin%%  %%CyML Model End%% | ModelUnit |
| %%CyML Initialization Begin%%  %%CyML Initialization End%% | Initialization |
| %%CyML Rate Begin%%  %%CyML Rate End%% | Rate (if separated from state) |
| %%CyML State Begin%%  %%CyML State End%% | State (if separated from rate) |
| %%CyML Compute Begin%%  %%CyML Compute End%% | Algorithm (including rate and/or state calculation) |
| %%CyML Ignore Begin%%  %%CyML Ignore End%% | Ignored statements |
| %%CyML Description Begin%%  %%CyML Description End%% | Meta-information |
| %%CyML ModelComposition Begin%%  %%CyML ModelComposition End%% | ModelComposition |

8- Pattern used for inputs and outputs meta-information:

Meta-information are included between “%%CyML Description Begin%%” and “%%CyML Description End%%” annotations.

Annotations and meta-information started with the symbol of a single comment statement of language.

Name (item)? description (units) (default, [mini, maxi]) category type

**Examples:**

!%%CyML Description Begin%%

! ABD Average bulk density for soil profile (g [soil] / cm3 [soil])

!     (10.5, [0.5 - 100.0]) state variable

! TLL Total soil water in the profile at the lower limit of

!     plant-extractable water (cm) (, [0.0 - 10000]) exogenous variable

!%%CyML Description End%%

#%%CyML Description Begin%%

# ABD(L)      Average bulk density for soil profile (g [soil] / cm3 [soil])

#            (10.5, [0.5 - 100.0])  state variable

#%%CyML Description End%%

The complete pattern is:



Fig. 1: Pattern of meta-information

**Fortran, DSSAT, and STICS requirements (+ the general requirements)**

1. Arguments

1.1 Use INTENT option: IN, OUT and INOUT

1.2. Or put a comment next to arguments that represent inputs (!Input), and outputs (!output). If a subroutine argument can receive a value that is changed during computations its intent is INOUT, then for that case “InOut” will be put.

1.3. Variables without the “INTENT” keyword are considered local variables. Their values should not be saved and used for the next routine call.

2. “SAVE” statement preserves the values of the local variables for the next invocation that breaks **Requirement 3**. To change that manually, this local variable should be defined as subroutine arguments.

3. Exhaustive use of the "IMPLICIT NONE" directive to detect bad variable usage

4. Use “allocatable” arrays to mimic List type (array with variable size)

5. Fortran array indices start at 1 by default, rather than at 0 as in CyML. The user needs to keep this default rule rather than fix the start from 1. CyMLTx will manage this difference.

**Example: DSSAT-EPIC Soil Temperature component.**

This component is provided with a CMakeLists file that indicates 5 files:

* STEMP\_EPIC.for: It is the main component that describes the initialization and rate calculation part of the component
* ASKEE\_DSSAT\_EPIC.for: The main program to run the component
* ModuleDefs.for: A module where some variables are described including control variables
* OPSTEMP.for: A routine to write Outputs values
* DATES.for: A routine to format date.

STEMP\_EPIC routine was tagged with the ModelUnit Tag as well as the initialization and rates calculations parts. Some parts of the code could be ignored with the "ignore" tag such as calling OPSTEMP.for, DATES.for. STEMP\_EPIC call SOILT\_EPIC (defined in STEMP\_EPIC.for) which is the only important function required for reusing this component. Control variable could be ignored since initialization and compute parts will be generated independently. The choice is left to whoever wants to reuse to combine the initialization and the algorithm. The declaration of some variables that are not useful for the rest of the code could be ignored such as DOY, YEAR, DYNAMIC.

After analyzing the component, few changes are needed:

NDays, TDL, WetDay, CUMPDT, and TMA are considered as local variables in the original code and are not defined as subroutine arguments. The code is expressive and limits the number of arguments since that SAVE statement is used and their values can be preserved. We define them as subroutine arguments since they are initialized in the Initialization part and their values are used in the computation part. As in Requirement 6, the computation part will take initialized variables as arguments. Likewise, in the rate calculation, the previous values of these variables are required, which allows considering them as arguments with INOUT INTENT. ST variable has INOUT intent.

Regarding the called routine SOILT\_EPIC, it uses a local variable X2\_PREV which is used in an assignment without being assigned and which is then assigned with X2\_AVG. So, for the first invocation of SOILT\_EPIC, X2\_PREV takes the default value 0.0 and it will assign X2\_AVG. For the next invocation, the value of X2\_PREV is the previous value of X2\_AVG, which breaks Genereral Requirement 3. X2\_PREV is defined explicitly as an argument to satisfy this requirement.

Complete the result with a unit test based on the ASKEE\_DSSAT\_EPIC file.

**Example STICS Soil Temperature component**

This component contains the first format of the model description proposed. This format was kept for STICS components transformation.

Some operations on arrays are changed to satisfy the general requirement 8

**BioMA requirements**

**Simplace requirements**