Modularization and transpilation of crop models using Crop2ML: a use case with SAMARA

Oriane Braud

August, 2023

Contents

Preamble

I have done my internship at CIRAD, the French Agricultural Research Centre for International Development (Centre International de Recherche en Agronomie pour le Développement). Founded in 1984 by merging former French tropical agricultural research organizations, this research center is an EPIC (Public Establishment of Industrial and Commercial interest), under the authority of the Ministry of Higher Education, Research and Innovation and the Ministry for Europe and Foreign Affairs. As such, it supports French science and diplomacy operations, by working for the sustainable development of tropical and Mediterranean regions. Its aim is to use research and innovation to design resilient farming and food systems for a more sustainable and inclusive world. Its expertise supports the entire range of stakeholders, from producers to public policymakers.

I worked more precisely at AGAP institute (Genetic Improvement and Adaptation of mediterranean and tropical Plants), within the PhenoMEn team (Phenotyping and Modeling of plants in their agro-climatic Environment) [1].

In this team, plant ecophysiology is a key discipline to identify and prioritize the traits and associated processes that explain the variability of plant performance at different organizational levels (tissue, organ, plant, crop) and functional levels (biochemistry, physiology, growth and development, etc.). This performance variability depends on the Genotype, Environment and Cropping System (GxExC). The complexity of the biological systems implies the acquisition of massive and heterogeneous data, at various spatio-temporal scales, by implementing adapted phenotyping methodologies. The analysis of these data, in order to evaluate and predict optimal combinations of traits and cultural practices in current or future agro-climatic scenarios, can only be done with the support of applied mathematics and computer science: modeling (statistical, mechanistic, predictive or explanatory) and modeling support tools, in particular software platforms and model exploration algorithms.

[1] https://umr-agap.cirad.fr/nos-recherches/equipes-scientifiques/phenotypage-et-modelisation-des-plantes-dans-leur-environnement-agro-climatique/contexte-et-enjeux

1 Introduction

Crop simulation models are dynamic computational models that simulate the development and growth of a crop in relation to environmental conditions (e.g. air temperature, soil water, evaporative demand, atmospheric CO2 concentration) and management practices (e.g. sowing date, irrigation, N fertilizer application, crop residue). Crop models have several purposes, from explaining complex processes happening in agricultural systems, to simulating possible situations (i.e. Genotype x Environment x Management (GxExM) interactions) and even predicting yields. Thus, crop models are crucial tools to aggregate an ever-growing knowledge of crop functioning, and provide informed support for decisions and policies.

Crop models are often developed in modeling platforms (i.e. environments in which crops models are executed), to ensure their future extension and to ease coupling with other models, like a soil model and a crop management event scheduler [11].

The crop modeling community has been active for about fifty years, resulting in a wide diversity of models in use. So, there is a growing need for systematic model inter-comparison, maintenance and improvement at the process level, in order to avoid a too large variability between the outputs of the models [17].

That's why, since 2010, the Agricultural Model Inter-comparison and Improvement Project[2] (AgMIP) community of experts has been improving and harmonizing tools and protocols to analyse and model agricultural systems, in order to be able to assess and predict the impacts of climate change and other factors on agriculture, food security, and poverty at local to global scales more accurately. This project strives for the interconnection of different research fields, linking agriculture with other topics like land use, nutrition, shocks, and others, resulting in the active participation of about 1,000 agricultural modelers and stakeholders worldwide.

AgMIP is establishing research standards, with for example the same assumptions across regions and models, and developing a rigorous process to evaluate agricultural models. As its name suggests, the AgMIP program relies on the inter-comparison of models, which is crucial for improving their components.

[2] https://agmip.org/

However, the diversity of implementation languages, software design and architectural constraints of crop models makes it difficult to exchange model components and code (i.e. algorithms) between platforms/models, which is impeding progress in this area. Further, it is crucial to foster collaboration among ecophysiologists, crop modelers, software engineers and model users to facilitate the integration of new knowledge in plant and soil sciences into crop models (i.e. exchange the knowledge rather than building black-box models), as well as to increase capabilities and responsiveness to stakeholder' needs.
To this end, several leading groups in the field have recently established the Agricultural Model Exchange Initiative (AMEI) [17, 31]. The AMEI is an open initiative that aims to tackle various challenges related to exchanging model units at different levels of granularity (from individual processes to whole plant) between modeling frameworks. The initiative's objectives include: (i) defining standards to describe unit and composite model exchange format using a declarative representation; (ii) implementing unit tests with invariants and shared standard parametrizations; and (iii) developing a web-platform that serves as a hub for publishing, documenting, and exchanging the model units [17].
Following FAIR principles (Findable, Accessible, Interoperable, Reusable) of open science, the AMEI designed a centralized framework, named Crop2ML, for exchanging and reusing model components between crop modeling platforms [18, 20].

Crop2ML has been developed recently, and has never been tested for a whole crop model. Therefore, no detailed methodology for its use has been provided and tested yet. In this work, I propose a methodology that covers: (i) the implementation of a model in a high-level language according to Crop2ML standards; (ii) the transpilation of this model using Crop2ML; and (iii) the modularization of a crop model, adapted for its reusability and exchangeability. I tested this methodology on a use case: crop model SAMARA.

2 Models

2.1 Crop2ML

Crop2ML (Crop Meta Modeling Language) is a centralized platform-independent framework for exchanging and reusing process-based crop model components between modeling platforms. A software framework is an abstraction that provides a standard way to build and deploy applications. Crop2ML has been developed by the AMEI, within the PhD thesis of Cyrille Midingoyi [18, 20, 19].
Crop2ML offers a unified and abstracted description of model components (meta-information and algorithms), to overcome the limitations of modeling platforms. Through an automatic transformation system, it converts high-level models, as often developed by agronomists, into platform-compliant components.

CyML, an intermediate language

Bidirectional source-to-source transformation systems between multiple languages require using intermediate languages and representation [27]. Crop2ML defines the CyML language, which is a subset of the Cython language to express the main constructs of the common computer languages (e.g. C, C++, Fortran, Java, C#, Python) that are used for crop model implementation in crop modeling platforms (DSSAT, BioMA, Record, SIMPLACE, OpenAlea, etc.). CyML is a minimal domain language for the description of associated algorithms of crop models [18], hence, for instance, it does not handle complex data structures, such as trees, dictionaries, etc.

Transpilations using CyMLTx and CyMLT

The bidirectional source-to-source transformation (i.e. transpilations) happens in two steps, using two tools: CyMLTx [19] and CyMLT [20].
CyMLTx generates automatically Crop2ML model components from existing platform-specific crop model components written in different languages. Conversely, CyMLT transforms Crop2ML models into model components in different languages and is targeted either at a specific platform (equivalent executable Python, java, C#, C++ components and packages usable from existing crop simulation platform), or to components with no

dependency to a specific platform (only a language is targeted).
Both tranpilations rely on a similar transformation workflow. CyMLTx transformation workflow is described in Figure 1. The transformation system integrates model documentation of the source code, that corresponds to the model specifications of Crop2ML models.

Modular approach

Crop2ML is based on a declarative architecture of modular model representation to describe the biophysical processes and their transformation to model components that conform to crop modeling platforms. The levels of granularity of modeling processes correspond to Crop2ML concepts such as ModelUnit and ModelComposite (respectively referred as "unit model" and "composite model" in the following text). Currently, Crop2ML can only handle two levels of granularity, i.e. one level of composition. A composite model is considered as a directed graph of unit models (i.e. a sequential order of the sub-models), connected by their inputs and outputs to manage model complexity [20].

Crop2ML tools and website

Crop2ML framework includes several tools to enable project development.
Pycrop2ML is an open, modular, and extensible library developed in Python that implements all the steps of Crop2ML model life cycle. PyCrop2ML can be executed via a command line interface, even by users with no knowledge of the Python language [20].
CropMStudio is a JupyterLab environment for Crop2ML model life cycle management. It makes the transpilations as easy as pushing a button.
Crop2ML has its own website[3], that contains a web-repository, CropMRepository, enabling registration, search and discovery of CropML models.
Note that Crop2ML does not provide a modeling solution that would compete with crop model simulation platforms.

2.2 SAMARA

SAMARA (Simulator of crop trait Assembly, MAnagement Response and Adaptation) [14, 15, 23, 24] is a deterministic mono-crop model, developed by CIRAD, specifically

[3] https://crop2ml.org/

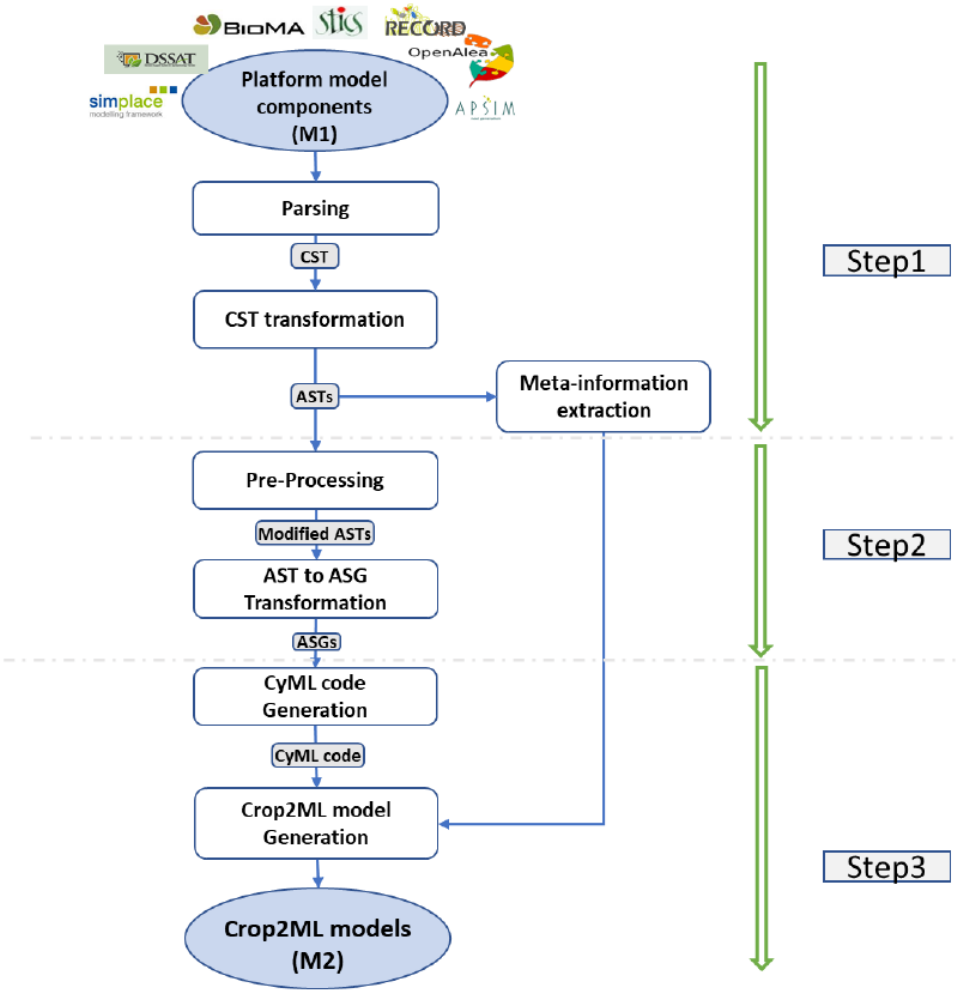

Figure 1: Schema of the main steps of the CyMLTx transformation workflow for Crop2ML model generation from platform model components. Green arrows show the three main steps of the transformation process: (Step 1) parsing of the code of model components using the ANother Tool for Language Recognition (ANTLR) parser generator [26]), then generation of the Concrete Syntax tree (CST) and extraction of meta-information from the Abstract Syntax Trees (ASTs) of the source code; (Step 2) pre-processing of the ASTs and generation of the Abstract Semantic Graphs (ASGs) of the source of the algorithms, initialization and auxiliary functions, for a language-independent representation; and (Step 3) transformation of the ASGs into the CyML language and merging with model specifications to generate a Crop2ML model component. Image from Midingoyi et al. (2023) [19].

designed for rice and sorghum cereal crops. It operates at a daily time step and requires daily agro-meteorological weather data, including variables such as hours of sunshine, temperature, humidity, wind speed, and rainfall. Additionally, hydrological topsoil properties such as volumetric water content at different soil moisture levels, percolation rate under flooded conditions, and soil depth limit for root growth are essential inputs for the model. SAMARA is implemented in C++ in the Artis platform, which also hosts other crop models like EcoMeristem [16] and XPalm [25], and is available on Windows. I worked with SAMARA V2.1.

Features of SAMARA

SAMARA distinguishes itself from other agronomy-scale crop models in its treatment of assimilate partitioning among plant sinks, incorporating more detailed morphology and phenology [6]. Unlike models that only consider carbon assimilation as the limiting factor, SAMARA also accounts for organ sink capacity, which refers to the potential size and number of plant organs that can grow and respire on a given day. This demand-supply interaction, measured by the state variable IC (Index of internal Competition), influences various morphogenetic and physiological processes, including tiller initiation or senescence, leaf size, internode elongation, root growth, and panicle dimensioning.

SAMARA considers the availability of water resources and stresses such as drought, logging, submergence, and thermal stresses. It does not incorporate mineral nutrition (i.e. N, P, K) into its calculations. However, it allows for simulating phenotypic plasticity, adaptive or non-adaptive, and different adaptation strategies based on resource utilization. It considers various crop management options, including transplanting versus direct seeding, flexible water management practices like stress cycles, alternate wetting and drying, deficit irrigation, and mulching.
SAMARA's conceptual model is provided in Figure 2, as an overview of the main processes interactions in SAMARA crop model.

Purpose of SAMARA

SAMARA's primary purpose is to facilitate the study and evaluation of in-silico plant type concepts (i.e. ideotypes) under different climatic, soil, and management conditions. The model serves as a valuable tool for pre-breeding research, aiding in the characteri-

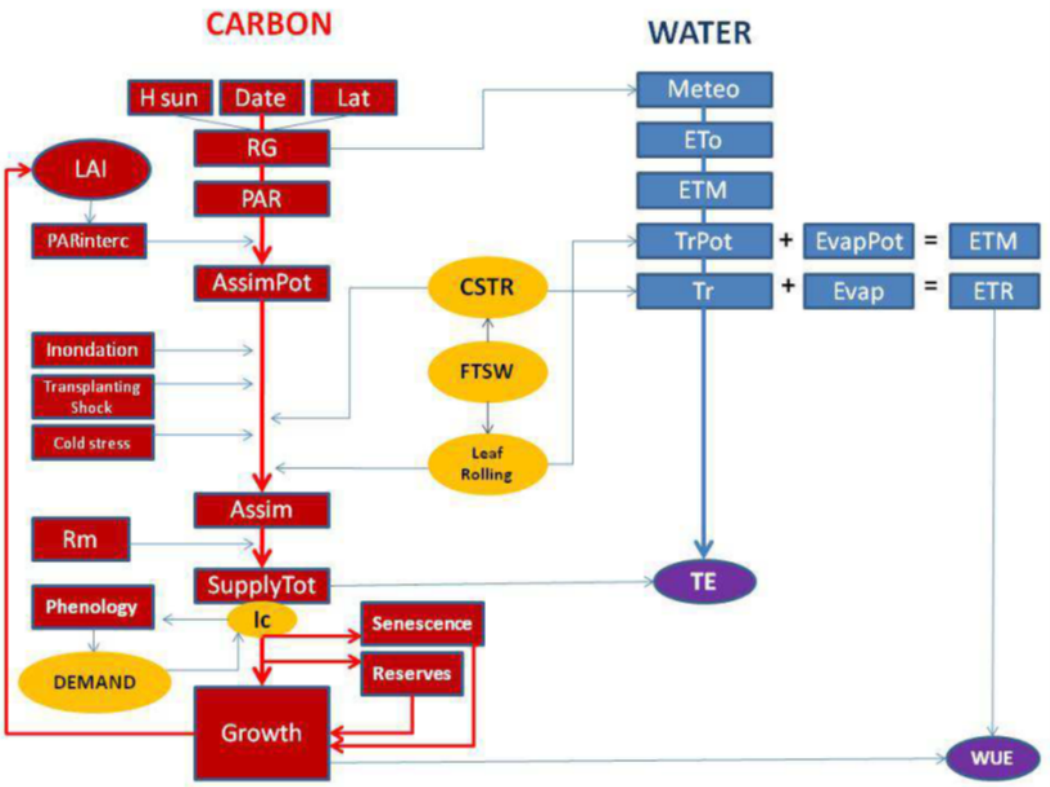

Figure 2: SAMARA's conceptual model. Image by Michael Dingkuhn.

zation of target environments and the development of in-silico ideotypes [23]. However, for applications such as agronomic decision support or mapping climatic yield potential, SAMARA may be considered over-parameterized, given the large number of parameters, some of which are difficult to calibrate due to their indirect measurability [14].

3 Method

The following method is my contribution to the AMEI.

3.1 Transpiling a crop model towards and from Crop2ML

This section describes the method to document, format and transpile from a language or crop modeling platform to another a crop model using Crop2ML.

3.1.1 Tailoring a crop model to Crop2ML standards

Documentation

Documentation is a crucial step in model development life cycle. It is essential for any kind of reuse, by the same person some time later or by another user, whatever his/her background. Explicit documentation is required for an efficient maintenance and (re)usability of any model.

The documentation of models, functions, variables and parameters (i.e. model specifications) has to follow a specific syntax to be understood by Crop2ML. All the specifications are described on Crop2ML Read the Docs[4]. For each unit model, the documentation must be written as a docstring, stating the name of the unit model, its version, its time step, its description (including the title, the author(s), the reference, the institution for which the authors work, an extended, and optionally a short, description), the inputs (parameters and variables, with their name, description, category, type, unit, etc.). All those model specifications must be organized in a hierarchy of indentations, with dashes and asterisks, as shown in Figure 3.

Code formatting

Tags have to be added to mark up the code area for CyMLTx (e.g. model beginning tag '`%%CyML Model Begin%%`' in Figure 3).

Data types have to be precised, whatever the initial language, in order to meet the requirements of all languages (in particular languages with explicit type expression) the model could be transpiled towards.

Some more adjustments may have to be made, depending on the specificities of the initial model. For example, global variables have to become *inout* variables in the unit and composite models. Moreover, Crop2ML only supports some basic libraries, for which there are equivalents in every language. Nevertheless, crop models usually don't need more complex libraries.

The implementation of unit and composites models is documented on Crop2ML Read the Docs[5].

[4] https://crop2ml.readthedocs.io/en/latest/user/specifications/specM2.html
[5] https://crop2ml.readthedocs.io/en/latest/user/index_spec.html

```python
# coding: utf8
from math import exp, sqrt, pow

#%%CyML Model Begin%%
def model_Assim(PARIntercepte: float, PAR: float, Conversion: float,
                                TMax: float, TMin: float, TBase: float, TOpt1: float, DayLength: float,
                                StressCold: float, CO2Exp: float, Ca: float, CO2Cp: float,
                                SlaMin: float, Sla: float, CoeffAssimSla: float, cstr: float, ASScstr: float,
                                CoeffCO2Assim: float):

    '''
    - Name: Assim
    - Version: 2.1
    - Time step: 1
    - Description:
                * Title: Assim model
                * Author: Michael Dingkuhn
                * Reference: https://doi.org/10.1016/j.fcr.2016.04.036
                * Institution: CIRAD
                * ExtendedDescription: This module calculates potential canopy level assimilation (AssimPot, kg/ha/d)
                * ShortDescription: None
    - inputs:
                * name: PARIntercepte
                            ** description : PAR intercepted by crop.
                            ** inputtype : variable
                            ** variablecategory : state
                            ** datatype : DOUBLE
                            ** default : 0.0
                            ** min : 0.0
                            ** max : 15.0
                            ** unit : MJ/d/m**2
```

Figure 3: Formatting of the first lines of a source code (here, source code is written in Python language) according to Crop2ML requirements.

3.1.2 Replicating a crop model in different languages

Transpilation process through Crop2ML

Once the model has been formatted accordingly, in any language supported by Crop2ML, it can be transpiled. There are two stages of transpilation: (i) from initial language (or platform) to Crop2ML; and (ii) from Crop2ML to another language (or platform). The documentation and comments of a model are parsed and transformed during the transpilation to Crop2ML. Generated CyML code is written in PYX files. Meta-information about the model (unit and composite models) is stored in XML files. A PNG image of the workflow of each composite model (i.e. the sequential execution of the unit models within it, a connection between two unit models corresponding to an exchange of variables) is generated only by the transpilation to any language or platform. A tree view of the structure of a Crop2ML model component package is available in [20] (Fig. 8).

Transpilations can be done using the user interface CropMStudio, or with command lines (example in §4.1) in a `conda` environment with the following specifications:

```
$ conda install -c amei -c openalea3 -c conda-forge pycropml
```

Any transpilation process can be assimilated to an identity function, such that, the models before and after transpilation can give the same outputs for any given inputs, i.e. a simulation through one model must be reproducible with the other one. Only the language and the implementation of the model has to change by transpiling, not its functionality and behavior.

One-to-many transpilations

Crop2ML supports 7 different languages (CyML, Python, R, C++, C#, Fortran, Java) and 8 different platforms (OpenAlea, DSSAT, BioMA, APSIM, SIMPLACE, Record, SiriusQuality, and STICS), which means that transpiling through a centralized framework like Crop2ML enables to replace $\frac{(8)(8-1)}{2} = 28$ different converters or wrappers.

3.2 Modularizing a crop model

3.2.1 Modularity & Model reuse

Definition of modularity

Modularity is the degree to which a system's components may be separated into modules, that can be recombined. The concept of modularity implies interdependence within and independence across modules [2]. In other words, the goal of modularizing a system is to form loosely coupled and self-contained parts of it.
Here, the system I considered is a set of connected unit models, that I aimed to modularize in order to form composite models. I have used the terms "modules" and "composite models" interchangeably.

Modularity applied to crop models

When modularizing a crop model (or a sub-set of a crop model) with the aim of exchanging and reusing its modules, a first concern should be about the biological and functional coherence of the modules, i.e. each module corresponds to a given plant or environmental

process. But soon, two other issues emerge: (i) the granularity of the modularization, i.e. how many modules should be formed and of which size and hierarchy; and (ii) the distribution of the model variables among the modules should be as optimized as possible, so that unit models that are in different modules exchange as few variables as possible. Indeed, crop models often work with large amounts of variables (e.g. around 300 for SAMARA), which, moreover, are not necessarily the same among the different crop models.

State of the art

Some models (e.g. DSSAT, Monica, APSIM, etc.) are already built with a notion of modularity. In the literature, some suggestions, provided by experts, of ways to consider separately the different plant processes can be found. They are shown in Table 1.

Wery (2005) [34]	Hay and Porter (2006) [9]	Adam (2010) [1]
- Phenology (vegetative and reproductive development) - Leaf area expansion - Production of assimilates - Partitioning of assimilates - Nitrogen dynamics - Transpiration	- Phenological development - Leaf canopy development - Biomass production - Biomass partitioning over the plant organs	- Phenology (vegetative and reproductive development) - Light interception (leaf area expansion, leaf canopy development) - Dry matter production - Partitioning / allocation (development of sink and assimilate partitioning) - Production level (e.g. water stress)

Table 1: Modularity of plant processes according to crop model experts

One can first observe that there is a certain consensus about isolating the phenology of the plant, which can be viewed as the timer of the plant development, as well as the plant and canopy development. However, the latter is strongly linked to the other processes, as light interception, along with water uptake, condition assimilation and is regulated by biomass production and stresses. And the challenge of modularizing a crop model mainly lies in those other processes.

3.2.2 Quantitative Graph Theory approach for Modularization

Mathematical tools are often helpful to get an objective view, even though those tools are chosen by the user and therefore biased. They can still serve to support decision. In the quest of modularizing crop models, I chose to use a quantitative graph theory approach in order to optimize the community structure of the crop model, i.e. its separation into modules.

Choice of graph

A process-based crop model can indeed be seen as a graph, whose vertices are its unit models, inputs and outputs are ports associated with vertices, and edges are directed from one output port of a vertex (e.g. a model) to the input port of another one to pass information. I considered the graphs associated to crop model as directed graphs, because the order in which the unit models are executed in the crop model can't be ignored. There are nevertheless several ways to formalize more precisely the connections between the unit models, depending on which aspect of the model is worth being emphasized. A port graph is very detailed and stores every port-to-port connections between the vertices (i.e. all variables exchanged between the unit models), whereas a workflow of a crop model only links two unit models if they exchange at least one variable. The graph I wanted to modularize was a hybrid, because it only records the number of connections between the vertices (i.e. the number of variables exchanged between the unit models). Indeed, when it comes to modularizing a crop model, which is a kind model involving many variables (e.g. about 300 variables in SAMARA), it can be interesting to consider the number of variables exchanged by the different unit models. However, a port graph would be harder to manipulate.
I have referred to this last directed weighted graph as "the graph of unit models". The mathematical definitions of all those graphs are provided below.

Definitions

Port graph of the model - The port graph associated to a crop model is a directed graph $G = (V, P_{in}, P_{out}, E)_{(\phi_{in},\phi_{out})}$ where:
$V = \{v_1, \ldots, v_n\}, n \in \mathbb{N}$ is a finite set of vertices (i.e. nodes), here the unit models in their order of execution in a simulation;

$P_{out} = \{(p_{s_1}, a_{s_1}), \ldots, (p_{s_n}, a_{s_n})\}, n \in \mathbb{N}$ and $P_{in} = \{(p_{t_1}, a_{t_1}), \ldots, (p_{t_n}, a_{t_n})\}, n \in \mathbb{N}$ are finite sets of tuples of, respectively, output and input ports, and their attributes, here the names of the variables;
$\phi_{in} : P_{in} \longrightarrow V$ and $\phi_{out} : P_{out} \longrightarrow V$ are applications associating respectively input and output ports to vertices; here it corresponds to associating input and outputs variables with a unit model; and
$E = \{(s_x, (p_{s_x}, a_{s_x})), (t_y, (p_{t_y}, a_{t_y})) \mid s_x, t_y \in V, x < y, (p_{s_x}, a_{s_x}) \in P_{out}, (p_{t_y}, a_{t_y}) \in P_{in}, a_{s_x} = a_{t_y}\}$ is a finite set of edges connecting output ports to input ports of vertices with a higher index (considering the order of the unit models in a simulation) when they have their attributes are equal; here it means that an edge is created when, among the output variables of one unit model and the input variables of another one executed after, two variables share the same name.

Workflow of the model - The workflow of a crop model corresponds to the daily sequential execution of the unit models, hence it can be formalized as a directed unweighted graph $G = (V, E')$ where:
$E' = \{(s_x, t_y) \mid s_x, t_y \in V, x < y, \exists (s_x, (p_{s_x}, a_{s_x})), (t_y, (p_{t_y}, a_{t_y})) \in E\}$ is a finite set of directed graph edges connecting vertices; two vertices are connected if there exists at least one port-to-port edge between the two same vertices in the port graph.

Graph of unit models - A directed weighted graph $G = (V, E'')$ can then be obtained from the port graph, such that:
$E'' = \{((s_x, t_y), w) \mid (s_x, t_y) \in E', w = \sum_{(s_i,(p_{s_i},a_{s_i})),(t_j,(p_{t_j},a_{t_j}))\in E} \delta((s_i, t_j) = (s_x, t_y))\}$ is a finite set of directed weighted graph edges connecting vertices, where the weight of an edge corresponds to the number of port-to-port edges between two vertices in the port graph.

Pre-processing

This was an assertive choice, guided by expert knowledge, to perform a pre-processing in order to work with a less complex network.
First, I formed some composite models manually, when some kind of consensus has been reached (e.g. for phenology, cf §3.2.1). In practice, the unit models concerned were then

removed from the weighted graph of unit models (the nodes and edges linked to them), in order to perform community detection algorithms only on unit models that are hard to classify.
Another pre-processing maneuver has been to quotient the weighted graph of unit models by forming groups that would correspond to systems of equations, considering another abstraction level between the unit and composite models. In the end, the goal was to form communities of systems of equations.

Exploring different graph community detection algorithms

To compute modules from the graph of unit models, I used community detection algorithms using different methods, of which I wanted to compare the outcome (a review of community detection algorithms can be found in [12]).
There was no issue of time or space complexity in the choice of the algorithms since I dealt with a rather small network. For format compatibility purposes, I worked with NetworkX graph (from Python library NetworkX), and all the algorithms chosen were either available in NetworkX or used this library (as for generalized k-means on graphs[6] [8]). The algorithms also had to work for directed weighted graphs.
The details of the chosen algorithms are displayed in Table 2.

Hierarchical community detection algorithms, such as Clauset-Newman-Moore [4] [32] [22] and Girvan-Newman [7] algorithms, operate by building a hierarchy of community structures. There are two main strategies to get this outcome: (i) agglomerative, i.e. a bottom-up approach where each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy; and (ii) divisive, i.e. a top-down approach where all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy. The merges and splits are often determined by optimizing a metric in a greedy manner (i.e. making the locally optimal choice at each stage).
Generalized K-means on graph is a k-means-like community detection algorithm, adapted to graphs, that utilizes centrality measures (e.g. PageRank, harmonic centrality, etc) [8].

[6] https://github.com/mhajij/Generalized_K-means_on_Graphs_Using_PageRank/tree/main

Algorithm	Method	Metric (optimization)	Parameter & chosen range
Clauset-Newman-Moore[7] [4] [32] [22]	Greedy Hierarchical Agglomerative	Modularity (maximization)	Resolution γ (> 1 for smaller communities) $[1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4]$
Girvan-Newman[8] [7]	Greedy Hierarchical Divisive	Edge betweenness centrality (edges with the largest one progressively removed)	Nb of communities $[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$
Generalized K-means on graphs[9] [8]	K-means-like greedy clustering from randomly picked nodes	Any centrality measure available in NetworkX	Nb of communities $[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$

Table 2: Main features of the community detection algorithms used to form communities from the reduced version of the directed weighted graph of the unit models.

Comparing the community structures using a metric

In order to compare and thus quantify (i.e. objectivate) the pertinence of the community structures generated by the community detection algorithms, I wanted to rely on a metric. So, according to the criteria mentioned §3.3, a metric that would suit this situation was one taking into account the number of variables exchanged between the composite models, that I wanted to minimize. I called this new metric "model nearness". An easy way to implement it was to take the complement to 1 of the edge coverage of a community structure. Indeed, the coverage of a community structure is the ratio of the number of intra-community edges to the total number of edges in the graph:

$$\text{coverage} = \frac{\text{nb of intra-community edges}}{\text{total nb of edges}}$$

, hence:

$$\text{Model nearness} = 1 - \text{coverage} = \frac{\text{nb of inter-community edges}}{\text{total nb of edges}}$$

The number of inter-community edges corresponds to the number of variables exchanged between the composite models. Thus, this is model nearness that I aimed to minimize while comparing the different community structures.

Sensitivity analysis

Model nearness strongly depends on the number of communities. The community structures generated by the community detection algorithms or formed manually with expert knowledge were only comparable if they had a similar number of communities. That's why I wanted to capture model nearness sensitivity to the number of communities.
Note that, in order to overcome the randomness of the generalized K-means algorithm, I ran the algorithm many times (e.g. 1000 repetitions) and averaged the model nearness for each value of the parameter (e.g. from 3 to 12 communities). This of course did not allow for an analysis of the pertinence of the communities formed for a given value of the parameter.

3.3 Software engineering method

3.3.1 Software engineering tools used for this project

As part of this work, I developed a new version of crop model SAMARA, called XSamara, which stands for "eXtended Samara".
GitHub project - This project is public and I made it available on GitHub[10]. The use of GitHub, as the most popular web-based version control and collaboration platform for software developers, made the collaborative development of XSamara easier and more efficient by being able to maintain a record of modifications made to the code, allow code review, and provide feedback via pull requests and issues.
License - XSamara is under the open source license Cecill-C.
Documentation - XSamara crop model is provided with a Sphinx documentation and tests on GitHub. The composite models are also separately available on CropMRepository[11], with their own documentation.
The goal of these choices was to gain in visibility and to make the transfer of the model and the methodology easier.

SAMARA's original source code - The C++ script of the earlier version of SAMARA can be found on GitHub[12]. From there, I drew the code of the unit mod-

[10] https://github.com/AgriculturalModelExchangeInitiative/XSamara/
[11] https://crop2ml.org/Repository
[12] https://github.com/PAMDeveloper/rSamara/

els, that I translated to Python, documented and then transpiled.

PyCrop2ML - Cloning PyCrop2ML repository[13] and creating a `conda` environment with the necessary requirements were necessary in order to use Crop2ML software. While progressing in the development of XSamara using PyCrop2ML, I interacted with PyCrop2ML developers by reporting issues on GitHub.

IDE - I used Visual Studio Code as an IDE.

Python packages - Most of the code I produced was in Python, for both the transpilation and the modularization part. Python was indeed interesting for the latter thanks to Python libraries, in particular I used NetworkX for graph analysis.

Graph visualization software - Cytoscape [33] was used to create the layout of Figure 7, and VisuAlea [30, 29] enabled the visualization of the port graph associated with XSamara, obtained by transpilation to OpenAlea platform.

3.3.2 Modeling workflow for end user

The aim was to create a user-friendly modeling workflow that could allow any non-computer-scientist to easily make their crop model or model component available in Crop2ML and reusable on any platform. This workflow is conceptualized in Figure 4.
This workflow does not explain how to integrate foreign model components to crop models. This step might imply many adjustments that are proper to each case.

4 Results & Discussion

4.1 SAMARA to XSamara

Rewriting of the unit models of SAMARA in Python

First, I wrote all the 84 unit models of SAMARA V2.1 from C++ to Python. The goal of this first transformation was: (i) to get to know this crop model, understand the hypotheses made and the physiological processes taken into account; (ii) to make sure

[13] https://github.com/AgriculturalModelExchangeInitiative/PyCrop2ML/

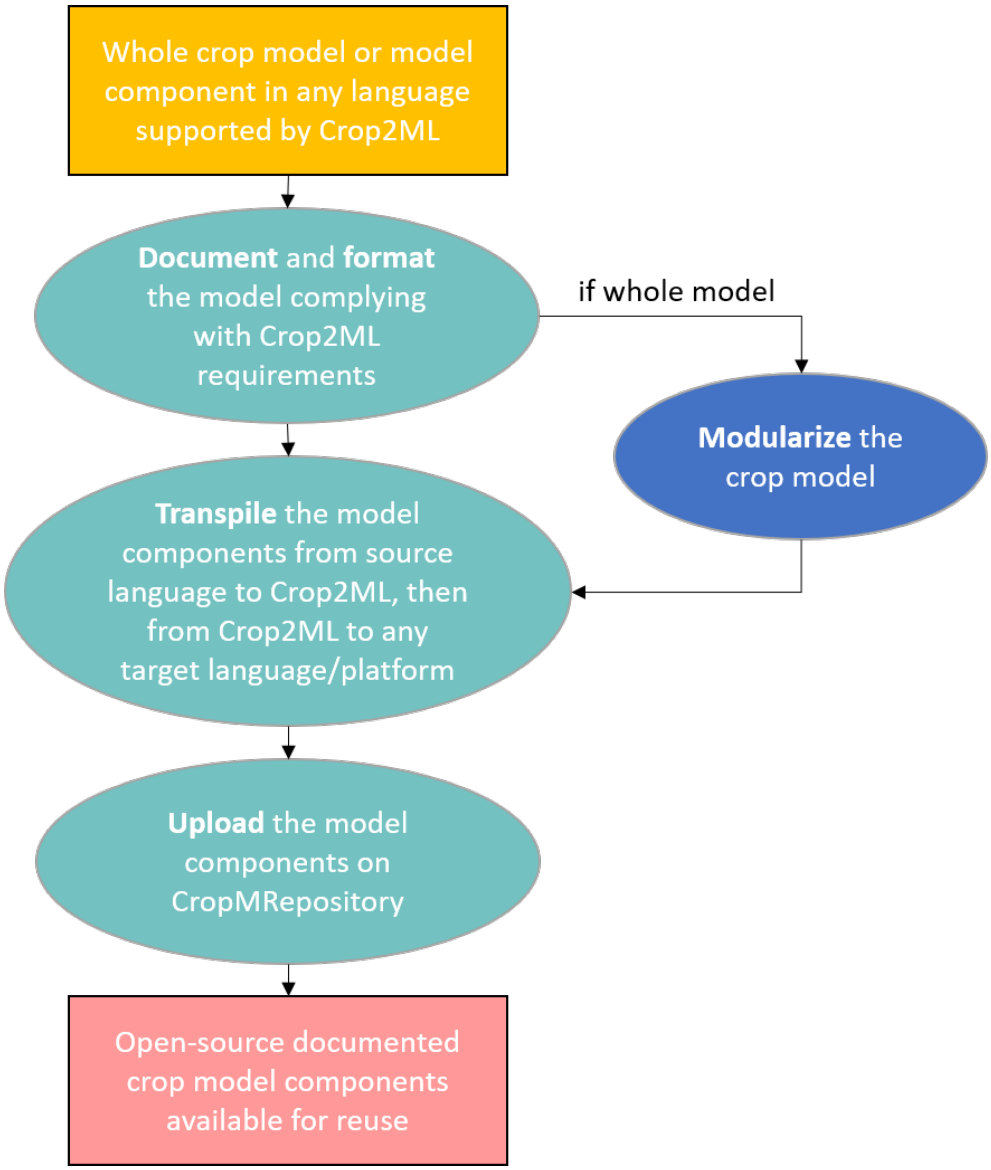

Figure 4: Modeling workflow for end user.

not to forget any feature of the model for which the transformation with Crop2ML would be hazardous; and (iii) to start the whole process from a high-level language in order to understand the issues that the researchers could possibly encounter using Crop2ML. Unit tests were performed for each unit models to ensure that the translated code was functional. This step was purely methodological, as I didn't know SAMARA and its implementation.

Documentation of the unit models

A documentation of the SAMARA model is available online[14]. From this, I had to write a formatted documentation in each unit model (syntax explained in §3.3.1.), so that it is readable by Crop2ML.
I created a database (i.e. an XLSX file) recording all the variables and parameters of the model (approximately 400 in total), in order to get organized while documenting the code. For each variable, their whole characteristics were specified: description, category, type, unit, default, min and max values, as well as the unit models they appear in, as input (in), output (out) or both (inout). The information found in this database is the same as the one provided in the documentation generated on https://crop2ml.org/ with the XML files.

Transpilations of the unit models

Once the files were correctly formatted (cf Code formatting in §3.1), they could undergo transpilation using Crop2ML. At this stage of the project, I considered the model as unmodularized, i.e. I only implemented one composite model XSamara.
I transpiled from Python to Crop2ML the code automatically using CyMLTx. I transpiled the unit models and the XSamara component from Python to Crop2ML automatically, using CyMLTx, with the following command line:

```
$ C:\XSamara> cyml -c samara . py
```

This transpilation created in a `crop2ml` directory: XML files for the metadata and PYX files for the CyML code in a algo\pyx\ directory.

Afterwards, I transpiled the unit models and the XSamara component from Crop2ML to any target language or platform available in Crop2ML automatically, using CyMLT, with the following command line:

```
$ C:\> cyml -p XSamara XSamara target_language_or_platform
```

The first XSamara corresponds to the project directory, and the second one to the name of the directories created to store the output files. This transpilation created in

[14] https://umr-agap.cirad.fr/nos-recherches/equipes-scientifiques/modele-samara, documents on the left

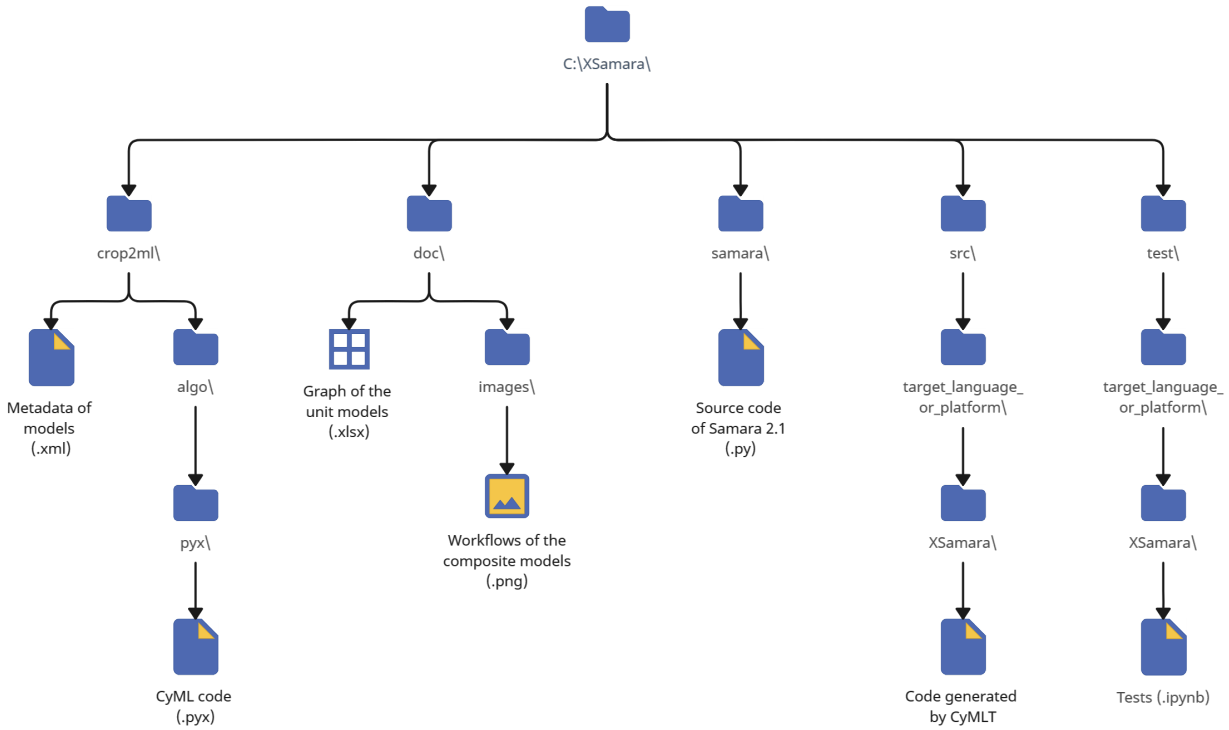

Figure 5: Tree view of the file architecture of XSamara project.

C:\XSamara\: (i) the generated code is stored in a directory src\target_language\XSamara\; (ii) tests in test\target_language\XSamara\; (iii) PNG figures of the generated workflows of the composite models in doc\images; and, only for a transpilation to OpenAlea platform, (iv) the graph of the whole model (i.e. an edge list with weights, as an XLSX files) in doc\. The file architecture of XSamara is provided in Figure 5.

Validation of code functional integrity after transpilation process

I had to make sure that the code provided the same results after either its rewriting to Python and its transpilation with Crop2ML. For this purpose, I transpiled SAMARA to C++, in order to execute this new version with the same environment as SAMARA V2.1. Some adjustments have been made because the C++ code obtained with CyMLT had a different way to manage the variables, mainly. Then, I could check whether simulations with SAMARA V2.1 and with the C++ version of XSamara, starting with the same input values, give the same outputs. The trace of the simulation has been validated with the help of a Qt application, specially designed for SAMARA.

4.2 Modularization

Pre-processing

The initial SAMARA model contained 84 unit models. There were two ways to reduce the number of unit models to include in the graph of unit models given to the community detection algorithms: (i) by merging together unit models when it was pertinent (e.g. when they were involved in the same precise plant process, or when they contained only one equation); and (ii) by forming manually, with expert knowledge, some modules whenever suitable (as described Pre-processing in §3.4): Phenology (vegetative and reproductive development, influenced by photo-thermal time), Temperature-induced Stresses and Sterility, Water Stresses (drought and logging), Rice-specific (transplanting, irrigation), and Sorghum-specific (mortality). In the end, I only had to deal with 49 unit models.

Moreover, I quotiented (i.e. partitioned) the graph of unit model minimally, not to constrain to much the modularization, grouping only the unit models related to organ structural demand on one side, and those related to organ structural growth on another side, in order to make sure that the unit models in each group stay together during modularization.

Sensitivity analysis

Model nearness increased with the number of communities, whatever the community detection method applied, as can be observed on Figure 6. It seems logical since the more communities there are, the more variables are exchanged in all.

The behavior of Girvan-Newman algorithm was singular, as it made some plateaus (i.e. small increase in model nearness with the separation of a community). It means that only a few cuts in the graph made model nearness highly increase, i.e. separated groups of unit models that share a consequent number of variables.

The first "leap" from 3 (model nearness = 0.08) to 4 communities (model nearness = 0.23) corresponded to the separation of a "plant architecture and organ structural demand" community from the other plant processes, such as C assimilation and supply, IC, dry matter production, and leaf area factors (LAI and SLA) computation. Indeed, many

variables are shared among these two blocks, as organ structural demand depends on the IC and C supply, organ growth on the demand, and leaf area factors on plant architecture. The second "leap" from 8 (model nearness = 0.28) to 9 communities (model nearness = 0.38) marked the separation of leaf area factors (LAI and SLA) computation from the large community comprising C assimilation and supply, IC and dry matter production. Between those "leaps", the plateaus corresponded most of the time to the creation of communities containing only one unit model (e.g. communities number 5, 6, and 7 correspond respectively to models that compute the actual radiation use efficiency (RUE), the maximal root speed, or the actual root depth). Yet, separating single unit models often had less meaning, even though it helps minimizing model nearness. Without those cuts in the graph, one could easily imagine that the difference of model nearness between the community structures obtained with Girvan-Newman algorithm and the other ones would be much smaller.

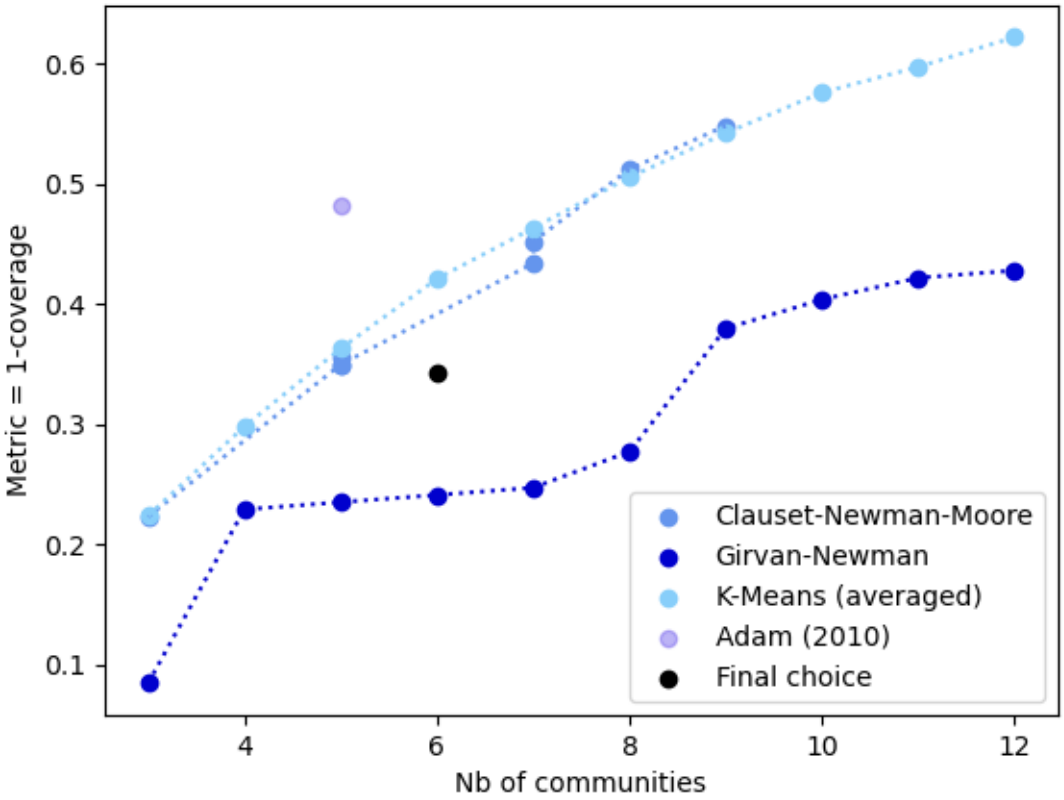

Figure 6: Graph of Model nearness = f(nb of communities)

Analysis of the community structures formed by the community detection algorithms

As observed in the sensitivity analysis, looking only to minimize model nearness may not be sufficient to form "optimal" communities, as it biases toward low-granularity community structures. Yet, opting for a very low granularity, i.e. big communities, cannot be considered as a well done modularization, as it does not really solve the black-box issue of crop models. Indeed, when asked to form only 3 communities, Clauset-Newman-Moore (CNM) and Girvan-Newman (GN) algorithms formed one for soil water dynamics, one for light interception (and C assimilation and respiration with CNM) and one for all the other plant processes.
Thus, instead of wanting to fix a threshold for model nearness, under which the community structure would be automatically considered as suitable, it would be more adapted to first fix the desired level of granularity (that would correspond here to a certain range of number of communities, as only one level of granularity can be implemented with Crop2ML) and then try to find trade-off between: (i) a relatively low value for model nearness; and (ii) the biological coherence of the community structure.

Regarding the general aspect of the community structures formed by the community detection algorithms, one can first observe that the generalized K-means algorithm did not provide relevant community structures overall. Indeed, as this algorithms starts by picking random unit models as heads of the future communities, regardless of their "proximity" in the graph, the clustering might for example split communities that are very close. So, it can't be used on its own.
Therefore, let's focus on the results of the two other algorithms. Similar communities emerged with both algorithms (disregarding the issue of single unit models with GN mentioned in the previous paragraph). CNM algorithm gave particularly promising community structures, up to a resolution of 1.6 (5 communities formed, model nearness = 0.35), above which some small meaningless communities were formed.

Post-processing and choice of community structure for XSamara

For the modularization of XSamara, I wanted to form about 5 or 6 communities. The most relevant community structures that would correspond to this constraint (equivalent

to a level of granularity) were: (i) the one obtained with CNM algorithm by setting a resolution of 1.6, giving 5 communities for a model nearness of 0.35; and (ii) the one got with GN algorithm by setting the number of communities to 9 (which would be equivalent to 6 without the single-model communities) for a model nearness of 0.38. The community structures are detailed in Table 3. I attributed names to the communities or groups of communities for better comprehension. The unit models in *italic* are the ones that have been manually changed during post-processing, guided by expert choice, to obtain the community structure retained for implementation in XSamara, showed in Figure 7.

Algorithm	CNM	GN
Nb of communities	5	9
Model nearness	0.35	0.38
Soil water dynamics	SurfaceEvaporation	SurfaceEvaporation
	FTSW	FTSW
	SoilSurfWaterAfterET	SoilSurfWaterAfterET
	MaxRootSpeed	RunoffAndDrainage-
	RunoffAndDrainage-	-SoilWaterReservoirs
	-SoilWaterReservoirs	FilledSoilWaterReservoirs
	FilledSoilWaterReservoirs	WaterLoggingUpland
	WaterLoggingUpland	UpdateSoilWaterReservoirs
	UpdateSoilWaterReservoirs	AvailableWater4Roots
	AvailableWater4Roots	*CarbonAndWaterBalances*
	ActualRootDepth	
	PotET	
	SeparationPotET	
	Transpiration	
	PotAndActualET	
		MaxRootSpeed
		ActualRootDepth
Evapo-transpiration		PotET
		SeparationPotET
		Transpiration
		PotAndActualET

Plant architecture & organ structural demand	LeafAppearance	LeafAppearance
	PlantHeightAndWidth	PlantHeightAndWidth
	PotLeafLength	PotLeafLength
	TotNbOfTillers	TotNbOfTillers
	TotNbOfLeavesPerHill	TotNbOfLeavesPerHill
	DeadTiller	DeadTiller
	DemandLeaf	DemandLeaf
	DemandSheath	DemandSheath
	DemandRoot	DemandRoot
	DemandInternode	DemandInternode
	DemandPanicleStruct	DemandPanicleStruct
Light interception & C assimilation	DeadLeaf	DeadLeaf
	TotalDryMatter	*TotalDryMatter*
	LAI	LAI
	LightInterception	SLAMitch
	SLAMitch	
	IncidentPAR	IncidentPAR
	InterceptedPAR	InterceptedPAR
	ActualRUE	LightInterception
	Assim	
	CarbonAndWaterBalances	
		ActualRUE

Dry matter production & IC	MaintenanceRespiration	*Assim*
	TotalSupply	MaintenanceRespiration
	ICVegetative	TotalSupply
	GrowthLeaf	ICVegetative
	GrowthSheath	GrowthLeaf
	GrowthRoot	GrowthSheath
	GrowthInternodeStruct	GrowthRoot
	GrowthPanicleStruct	GrowthInternodeStruct
	PaniclePriorityPhase4	GrowthPanicleStruct
	TotalGrowthStruct	PaniclePriorityPhase4
	ICReproductive	TotalGrowthStruct
	PanicleGrainFilling	ICReproductive
	GrowthInternodeReserve	PanicleGrainFilling
	GrowthRootWithExcess-	GrowthInternodeReserve
	-Assim	GrowthRootWithExcessAssim

Table 3: Most relevant community structures generated by community detection aglorithms Clauset-Newman-Moore (CNM) and Girvan-Newman (GN).

The model nearness computed for this community structure of 6 communities was of 0.34. This was lower than for both community structures, obtained by algorithms, from which this chosen community structure was inspired.

In comparison, the community structure of 5 communities suggested in Adam et al. (2010) [1], so only formed with expert knowledge and obviously biologically relevant, had a value of model nearness of 0.48. The algorithms performed better locally (i.e. for a neighboring number of communities), whatever the community detection method they used. Even though the algorithms chosen do not directly take model nearness into account in their community detection method, they integrate in their way the concept of modularity defined in §3.2.1, and model nearness focused on the "independence across modules" aspect. However, their greediness doesn't allow for a global optimum to be reached.

Therefore, post-processing a result obtained with algorithms thanks to expert input seems to be a nice combination. Indeed, it enables to optimize both model nearness (both thanks

to the algorithms and also by manual post-processing) and the biological pertinence of the community structure.

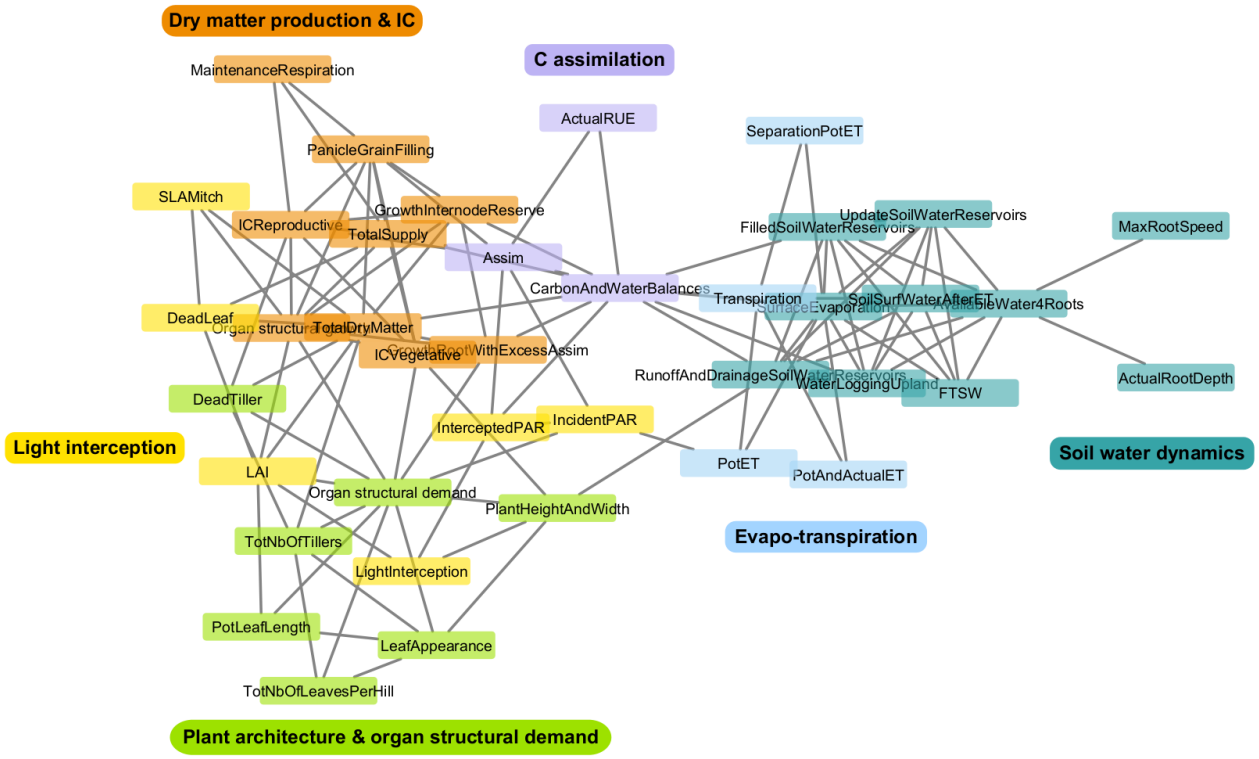

Figure 7: Sub-set of the community structure retained for XSamara after post-processing. Displayed using Cytoscape, with a force-directed layout.

4.3 Transfer to users

XSamara crop model and its documentation

XSamara crop model and its Sphinx documentation, as an HTML page, are available in the XSamara GitHub project[15].

Composite models from XSamara

The separate composite models from XSamara have been uploaded and are now available, on CropMRepository[16]. Each composite model and the unit models within have their own

[15] https://github.com/AgriculturalModelExchangeInitiative/XSamara/
[16] https://crop2ml.org/Repository

documentation on CropMRepository, and can also be downloaded.

5 Limitations and perspectives

5.1 On a widespread use of Crop2ML

Crop2ML is a rather young framework (created around 2020), so there are still some adjustments to be made. I already addressed some GitHub issues that will be treated later by the developers. The tool in itself is very promising, but using it implies to change the way crop models are conceived.

The whole process of documenting, formatting the code, modularizing the model, etc. can be time- and labor-intensive, which could explain the reluctance of some researchers to adopt this practice. Nevertheless, one should consider that this is the "price to pay" anyway in order to build a long-lasting, multiple-user, user-friendly, easy-to-maintain model.

Exchanging and reusing model components requires collaboration between the scientific teams in different research units, universities or even countries, as well as the willingness to share their model as an open source piece of work. Enabling a wider (i.e. international) use of this framework, in a field where every research unit has a platform of preference, seem to have more to do with their own politics than with technical issues (even though there are still some).

Transpilation sprints have already been organized internationally by the AMEI, for some crop model components (e.g. soil temperature and energy balance). This helps with comparing the different approaches for some special processes, and also finding equivalences (thanks to a common formalism), as well as building partnership between the research teams around the world. Model components exchanges between crop modeling platforms are currently being tested.

5.2 On modularizing crop models

Using community detection algorithms for modularizing a crop models has proved to be interesting to guide towards an optimized modularization, but not sufficient.

Relying on a metric enables to objectively compare community structures (automatically

or manually formed). Model nearness was particularly interesting in our case, as it implies, in practice, a lower number of adjustments to be made to make any model components work together. One could imagine implementing a new community detection algorithm whose metric to optimize would be model nearness.
Nonetheless, ultimately, a determining approach was adding expert-driven manual input, which appears essential for achieving meaningful and reusable modules. This sentiment aligns with the observation made by Miller and Elgard (1998), who aptly stated, "modularity balances standardization and rationalization with customization and flexibility" [21].
As part of the "customization and flexibility" aspects of modularity, any desired level of granularity can be chosen by the user, and the comparison of several community structures using a metric is only possible for a given level of granularity. However, in the current version of Crop2ML framework, it is only possible to implement one level of granularity (i.e. it is currently impossible to form composites of composites).

Simulations with a crop model are performed with a daily time-step, so that, every day of a simulation, all unit model are executed in a certain order, updating the variables in a certain order. At the end of the day, the output variables are given as input for the next day. Even though the order of execution of the unit models (i.e. crop processes) is most of the time immutable during a given day, some unit models are dedicated to the computations of variables that could, theoretically, be done either at the end of one day, or at the beginning of the next one. However, all the graphs described in §3.2.2, and in particular the simulation workflow, only represent what happens in one day of simulation. In other words, those are not cyclic graphs as they don't take into account the exchange of variables between the unit models from one day to another, and it biases the connections between the models (for any level of granularity), and thus model nearness.

5.3 On semantic standardization

Crop models often involve a large amount of variables and parameters (e.g. around 400 for Samara), and different ones across models. Thus, there is a particular concern whereas to how to deal with them while reusing and exchanging model components. Indeed, even though many of them might refer to the same measures and concepts, equivalents have

to be found despite different names. It is easier to refer to the same concepts with a same name. So, there is a need of semantic standardization among crop models [28].
One possible solution could be to follow the semantic standard of the ICASA dictionary[17] [3]. This project is still developing and the ICASA dictionary does not contain all the imaginable variables and parameters used in crop models yet.

6 Conclusion

In this work, I : (i) implemented the unit models of SAMARA V2.1 crop model in a high-level language (Python); (ii) documented and formatted them, complying with Crop2ML requirements; (iii) created a new version of SAMARA crop model that I called XSamara, with some additional changes; (iv) set up and tested a method to modularize crop models based on graph theory; (v) modularized XSamara; (vi) transpiled XSamara (as a whole and modularized) from Python to Crop2ML, and from Crop2ML to C++, OpenAlea platform, etc.; (vii) uploaded the modules of XSamara on CropMRepository, giving them visibility and a proper web documentation; (viii) generated a Sphinx documentation for XSamara; and (ix) provided a modeling workflow for the end user.

Crop2ML has the potential to become a major tool in the crop modeling community (the AgMIP in particular), as it facilitates model components exchange and reuse, and thus inter-comparison and improvement. An available easy-to-use workflow for modularizing and transpiling crop models using Crop2ML will benefit its wide use. Nevertheless, there are still efforts to make in the crop modeling community for exchange and reuse of model components to become common practices in crop model development.

Acknowledgments

I used ChatGPT for rephrasing certain parts (mostly in §1 and 2.2).
I would like to thank Christophe Pradal and Myriam Adam for their valuable overall

[17] https://docs.google.com/spreadsheets/d/1MYx1ukUsCAM1pcixbVQSu49NU-LfXg-Dtt-ncLBzGAM/pub?output=html

internship supervision and expert input; Cyrille Mindingoyi for dealing with the issues I reported on PyCrop2ML GitHub; Gregory Beurier for answering my questions about the C++ code of SAMARA; Michael Dingkuhn for answering my questions about the eco-physiology behind SAMARA; Pierre Martre for insights about ICASA; Romain Fernandez for proof-reading and helpful conversations; Frederic Boudon for providing me with the opportunity to present my work at an inter-team seminar on gramineae modeling; Laura Lescroart for encouraging me to present my work at the Agapiades; and in particular Christine Granier and the other interns and PhD students for emotional support at coffee breaks.

References

[1] Myriam Adam, Frank Ewert, Peter A Leffelaar, Marc Corbeels, Herman Van Keulen, and Jacques Wery. Crospal, software that uses agronomic expert knowledge to assist modules selection for crop growth simulation. *Environmental Modelling & Software*, 25(8):946–955, 2010.

[2] Carliss Young Baldwin and Kim B Clark. *Design rules: The power of modularity*, volume 1. MIT press, 2000.

[3] J Bouma and JW Jones. An international collaborative network for agricultural systems applications (icasa). *Agricultural Systems*, 70(2-3):355–368, 2001.

[4] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004.

[5] Jean-Claude Combres, Benoît Pallas, Lauriane Rouan, Isabelle Mialet-Serra, Jean-Pierre Caliman, Serge Braconnier, Jean-Christophe Soulié, and Michael Dingkuhn. Simulation of inflorescence dynamics in oil palm and estimation of environment-sensitive phenological phases: a model based analysis. *Functional Plant Biology*, 40(3):263–279, 2012.

[6] Michael Dingkuhn, Kouressy Mamoutou, Michel Vaksmann, B. Clerget, and Jacques Chantereau. A model of sorghum photoperiodism using the concept of threshold-

lowering during prolonged appetence. *European Journal of Agronomy*, 28:74–89, 02 2008.

[7] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[8] Mustafa Hajij, Eyad Said, and Robert Todd. Generalized K-means for Metric Space Clustering Using PageRank. In Panagiotis D. Ritsos and Kai Xu, editors, *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association, 2020.

[9] RK Hay and JR Porter. The physiology of crop yield. 2006.

[10] Alexandre Bryan Heinemann, Michael Dingkuhn, Delphine Luquet, Jean Claude Combres, and Scott Chapman. Characterization of drought stress environments for upland rice and maize in central brazil. *Euphytica*, 162(3):395–410, 2008.

[11] Gerrit Hoogenboom, Eric Justes, Christophe Pradal, Marie Launay, Senthold Asseng, Frank Ewert, and Pierre Martre. icropm 2020: Crop modeling for the future. *The Journal of Agricultural Science*, 158(10):791–793, 2020.

[12] Muhammad Aqib Javed, Muhammad Shahzad Younis, Siddique Latif, Junaid Qadir, and Adeel Baig. Community detection in networks: A multidisciplinary review. *Journal of Network and Computer Applications*, 108:87–111, 2018.

[13] Mamoutou Kouressy, Michael Dingkuhn, Michel Vaksmann, and Alexandre Bryan Heinemann. Adaptation to diverse semi-arid environments of sorghum genotypes having different plant type and sensitivity to photoperiod. *Agricultural and Forest Meteorology*, 148(3):357–371, 2008.

[14] Uttam Kumar, Ma Rebecca Laza, Jean-Christophe Soulié, Richard Pasco, Kharla V.S. Mendez, and Michael Dingkuhn. Compensatory phenotypic plasticity in irrigated rice: Sequential formation of yield components and simulation with samara model. *Field Crops Research*, 193:164–177, 2016.

[15] Laza, Michael Dingkuhn, U. Kumar, MR, and Richard Pasco. Samara: A crop model for simulating rice phenotypic plasticity. 2016.

[16] Delphine Luquet, Michael Dingkuhn, HaeKoo Kim, Ludovic Tambour, and Anne Clement-Vidal. Ecomeristem, a model of morphogenesis and competition among sinks in rice. 1. concept, validation and sensitivity analysis. *Functional Plant Biology*, 33(4):309–323, 2006.

[17] Pierre Martre, Marcello Donatelli, Christophe Pradal, Andreas Enders, Cyrille Ahmed Midingoyi, Ioannis Athanasiadis, Davide Fumagalli, Dean Holzworth, Gerrit Hoogenboom, Cheryl Porter, Hélène Raynal, Andrea E. Rizzoli, and Peter J. Thorburn. The agricultural model exchange initiative. In *Abstracts of the 7th AgMIP Global Workshop*, volume 7, pages 17–18, San José, Costa Rica, 2018. IICA.

[18] Cyrille Ahmed Midingoyi, Christophe Pradal, Ioannis N. Athanasiadis, Marcello Donatelli, Andreas Enders, Davide Fumagalli, Frédérick Garcia, Dean Holzworth, Gerrit Hoogenboom, Cheryl Porter, Hélène Raynal, Peter Thorburn, and Pierre Martre. Reuse of process-based models: automatic transformation into many programming languages and simulation platforms. *In Silico Plants*, 2020.

[19] Cyrille Ahmed Midingoyi, Christophe Pradal, Andreas Enders, Davide Fumagalli, Patrice Lecharpentier, Hélène Raynal, Marcello Donatelli, Ioannis N. Athanasiadis, Cheryl Porter, Gerrit Hoogenboom, Dean Holzworth, and Pierre Martre. Crop modeling frameworks interoperability through bidirectional source code transformation. in press in Environmental Modelling & Software, 2023.

[20] Cyrille Ahmed Midingoyi, Christophe Pradal, Andreas Enders, Davide Fumagalli, Hélène Raynal, Marcello Donatelli, Ioannis N. Athanasiadis, Cheryl Porter, Gerrit Hoogenboom, Dean Holzworth, Frédérick Garcia, Peter Thorburn, and Pierre Martre. Crop2ml: An open-source multi-language modeling framework for the exchange and reuse of crop model components. *Environmental Modelling & Software*, 142:105055, 2021.

[21] Thomas D. Miller and Per Elgard. Defining modules, modularity and modularization. In *Proceedings of the 13th IPS research seminar, Fuglsoe*. Aalborg Universiy Fuglsoe, 1998.

[22] Mark E.J. Newman. Analysis of weighted networks. *Physical review E*, 70(5):056131, 2004.

[23] Philippe Oriol, Myriam Adam, Grégory Aguilar, Richard Pasco, Jean-Christophe Soulie, and Michael Dingkuhn. Samara: between a functional structural plant model and an agronomic model. 2013.

[24] Philippe Oriol, Myriam Adam, Grégory Aguilar, Jean-Christophe Soulie, Richard Pascol, Delphine Luquet, Serge Braconnier, and Michael Dingkuhn. An overview of samara crop model and some applications on multipurpose sorghum. 2014.

[25] Benoît Pallas, Anne Clément-Vidal, Maria-Camila Rebolledo, Jean-Christophe Soulié, and Delphine Luquet. Using plant growth modeling to analyze c source–sink relations under drought: inter-and intraspecific comparison. *Frontiers in plant science*, 4:437, 2013.

[26] Terence Parr. The definitive antlr 4 reference. *The Definitive ANTLR 4 Reference*, pages 1–326, 2013.

[27] David Plaisted. Source-to-source translation and software engineering. *Journal of Software Engineering and Applications*, 06:30–40, 01 2013.

[28] Cheryl H. Porter, Chris Villalobos, Dean Holzworth, Roger Nelson, Jeffrey W. White, Ioannis N. Athanasiadis, Sander Janssen, Dominique Ripoche, Julien Cufi, Dirk Raes, Meng Zhang, Rob Knapen, Ritvik Sahajpal, Kenneth Boote, and James W. Jones. Harmonization and translation of crop modeling data to ensure interoperability. *Environmental Modelling & Software*, 62:495–508, 2014.

[29] Christophe Pradal, Daniel Barbeau, Thomas Cokelaer, and Eric Moscardi. VisuAlea, Towards a Scientific Modelling Environment using Visual Programming. EuroSciPy 2010, 2010.

[30] Christophe Pradal, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier, and Christophe Godin. Openalea: a visual programming and component-based software platform for plant modelling. *Functional Plant Biology*, 35:751–760, 2008.

[31] Helene Raynal, Andreas Enders, Pierre Martres, Ioannis Athanasiadis, Marcello Donatelli, Dean Holzworth, and Christophe Pradal. Agricultural Model Exchange Initiative (AMEI). In *9th International Congress on Environmental Modelling and Software*, Fort Colins, United States, June 2018.

[32] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical review E*, 74(1):016110, 2006.

[33] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[34] Jacques Wery. Differential effects of soil water deficit on the basic plant functions and their significance to analyse crop responses to water deficit in indeterminate plants. *Australian Journal of Agricultural Research*, 56(11):1201–1209, 2005.