Environmental Modelling and Software 142 (2021) 105055

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft

Crop2ML: An open-source multi-language modeling framework for the exchange and reuse of crop model components

Cyrille Ahmed Midingoyi [a], Christophe Pradal [b,c,**], Andreas Enders [d], Davide Fumagalli [e], Hélène Raynal [f], Marcello Donatelli [g], Ioannis N. Athanasiadis [h], Cheryl Porter [i], Gerrit Hoogenboom [i,j], Dean Holzworth [k], Frédérick Garcia [l], Peter Thorburn [m], Pierre Martre [a,*]

[a] LEPSE, Univ Montpellier, INRAE, Institut Agro, Montpellier, France
[b] CIRAD, UMR AGAP Institut, F-34398, Montpellier, France
[c] LIRMM, Univ Montpellier, Inria, CNRS, Montpellier, France
[d] Institute of Crop Science and Resource Conservation (INRES), University of Bonn, Bonn, Germany
[e] Institute for Environment and Sustainability, Joint Research Centre, European Commission, Ispra, Italy
[f] AGIR, INRAE, INP Toulouse, Castanet-Tolosan, France
[g] Research Centre for Agriculture and Environment, CREA-AA, Bologna, Italy
[h] Wageningen University, Wageningen, the Netherlands
[i] Agricultural & Biological Engineering, University of Florida, Gainesville, USA
[j] Institute for Sustainable Food Systems, University of Florida, Gainesville, USA
[k] CSIRO Agriculture and Food, Toowoomba, Australia
[l] MIAT, INRAE, Castanet-Tolosan, France
[m] CSIRO Agriculture and Food, Brisbane, Australia

ARTICLE INFO

Keywords:
Crop model
Crop2ML
Component-based software
Model exchange and reuse

ABSTRACT

Process-based crop models are popular tools to analyze and simulate the response of agricultural systems to weather, agronomic, or genetic factors. They are often developed in modeling platforms to ensure their future extension and to couple different crop models with a soil model and a crop management event scheduler. The intercomparison and improvement of crop simulation models is difficult due to the lack of efficient methods for exchanging biophysical processes between modeling platforms. We developed Crop2ML, a modeling framework that enables the description and the assembly of crop model components independently of the formalism of modeling platforms and the exchange of components between platforms. 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. Here, we present Crop2ML framework and describe the mechanisms of import and export between Crop2ML and modeling platforms.

1. Introduction

The wide range of crop process-based models (PBM) reflects the evolution of our knowledge of the soil-plant-atmosphere system and the rich historical development for more than five decades (reviewed in Jones et al., 2017; Muller and Martre 2019). The high diversity of PBM is due to their multiple applications and the complexity of the system influenced by several factors, e.g. weather, soil, crop management (Basso et al., 2013) and genotypic factors (Wang et al., 2019). Most of the PBM are continuous models, formalized using ordinary differential equations, but are implemented as discrete time simulation models using finite difference equations. They are commonly decomposed into simpler biophysical functions (e.g. phenology, morphogenesis, resource acquisition, pests and diseases impact) often implemented by recurrent

* Corresponding author. INRAE – UMR LEPSE. 2 place Viala, 34 060, Montpellier, France.
** Corresponding author. CIRAD – UMR AGAP, Avenue Agropolis, 34 398, Montferrier-sur-Lez, France.
E-mail addresses: cyrille.midingoyi@inrae.fr (C.A. Midingoyi), christophe.pradal@cirad.fr (C. Pradal), aenders@uni-bonn.de (A. Enders), davide.fumagalli@ext.ec.europa.eu (D. Fumagalli), helene.raynal@inrae.fr (H. Raynal), marcello.donatelli@crea.gov.it (M. Donatelli), ioannis.athanasiadis@wur.nl (I.N. Athanasiadis), cporter@ufl.edu (C. Porter), gerrit@ufl.edu (G. Hoogenboom), dean.holzworth@csiro.au (D. Holzworth), frederick.garcia@inrae.fr (F. Garcia), peter.thorburn@csiro.au (P. Thorburn), pierre.martre@inrae.fr (P. Martre).

https://doi.org/10.1016/j.envsoft.2021.105055
Accepted 15 April 2021
Available online 30 April 2021
1364-8152/© 2021 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

equations with control flows. Another common characteristic is that PBM simulate plant growth and development at the scale of the canopy or average plant level without spatial dependence with a daily or sub-daily time step.

PBM are often implemented in modeling and simulation platforms at a higher level of abstraction to facilitate model development (Rizzoli et al., 2008). These platforms offer not only scalable, modular, and robust modelling solutions but also the ability to analyze, evaluate, reuse and combine models. The diversity of PBM led the crop modeling community to compare their performance and to improve them by aggregating modelers' knowledge or by introducing improvements provided from diverse research groups under the umbrella of large international collaborative projects such as the Agricultural Model Intercomparison and Improvement Project (AgMIP; Rosenzweig et al., 2013). Studies conducted in the context of model intercomparison and improvement exercises (e.g. Asseng et al., 2013; Wang et al., 2017) pointed out the large uncertainty of PBM simulations and have analyzed the sources of uncertainty or the processes involved. These intercomparison results showed the potential and limits of PBM and highlighted the need to analyze models at the process level, but also to exchange model components describing specific processes between simulation platforms (e.g. Donatelli et al., 2014; Wang et al., 2017). The uncertainty of a PBM component may be related to its validity domain, inputs, parameters, structure, and the underlying scientific hypotheses (Walker et al., 2003). Epistemic uncertainty may arise from incomplete or lack of knowledge of these sources. The uncertainty of PBM results from the aggregation of the uncertainty of each of its component (Refsgaard et al., 2007). A framework that would allow the exchange of model components between different platforms would give crop modelers the ability to test alternative hypotheses in the same model, thus helping to reduce epistemic uncertainty.

Although most crop simulation platforms provide modular approaches and reuse techniques, there is little exchange of PBM components between them despite theoretical and application interests. PBM components often contain source code developed in different programming languages and are tightly coupled to the platforms. Therefore, model components are not seamlessly reusable outside the modeling platforms in which they have been developed without recoding or wrapping them (Holzworth et al., 2014; Rizzoli et al., 2008). Re-implementing a component in several platforms is a tedious and cumbersome task and requires a minimum knowledge of the different platforms. The wrapping solution treats components as black boxes taking little or no advantage of the framework (Rizzoli et al., 2008) or as white boxes but with a high-level of complexity (Fernique and Pradal, 2018; Pradal et al., 2008). Other reuse approaches in environmental modeling have been explored. Declarative modeling can provide portability and facilitate integration between independent, uncoordinated models (Athanasiadis and Villa, 2013). However, model specifications are seldom separate from implementation details. Model builders rely often directly on implementation that hides the scientific content of a model (i.e. its Algorithm) and its structure. Moreover, the publication of PBM components in scientific journals does not provide sufficient description associated with the modeled processes, which is a fundamental criterion for reuse (Pradal et al., 2013). This raises the problem of reproducibility and reliability of scientific results that are strongly linked to the platforms in which the models have been implemented and tested (Cohen-Boulakia et al., 2017; Hinsen, 2016).

Visual domain-specific languages such as Simile (Muetzelfeldt and Massheder 2003) or Stella (Richmond, 1985) provide a rich graphical interface to build models but become difficult to use for complex models and require many widgets to represent graphically nested control flows. Multiscale modelling and simulation frameworks (Marshall-Colon et al., 2017; Pradal et al., 2015) propose model interface designs which enables communication of multi-language components as black box components. Other declarative modelling languages are also used in the Systems Biology community who have developed declarative open standard such as SBML (Hucka et al., 2010), CELLML (Cuellar et al., 2003), or NEUROML (Le Franc et al., 2012) to describe biological models. However, crop modelers generally use procedural modelling rather than a mathematical formalism like differential or reaction equations as it is commonly done in System biology.

An alternative to the problem of PBM component reuse between PBM platforms is the use of a centralized framework that enables the development of PBM components regardless of the modeling platforms (Fig. 1). We followed this approach and developed a modeling framework called Crop2ML (Crop Modelling Meta Language) that separates the structure of a model component from its implementation. Given that the wrapping solution was excluded because of the lack of transparency and high maintenance cost and that Crop2ML does not aim at replacing existing modeling platforms or at simulating components within large modeling solutions (crop models), we created a solution that generates components, from a metalanguage, for specific PBM platforms. It provides a centralized PBM components repository to store model components in a standard format to facilitate their access and reuse. This reuse approach is supported by the Agricultural Modeling Exchange Initiative (AMEI), which brings together some of the most widely used crop modelling and simulation platforms, including the Agricultural Production Systems sIMulator (APSIM, Holzworth et al., 2018), the Biophysical Model Applications (BioMA; Donatelli et al., 2010), the Decision Support System for Agrotechnology Transfer (DSSAT; Jones et al., 2003; Hoogenboom et al., 2019), OpenAlea (Pradal et al., 2015), the REnovation and COORDination of agroecosystems modelling (RECORD; Bergez et al., 2013), and the Scientific Impact assessment and Modeling Platform for Advanced Crop and Ecosystem management (Simplace; Gaiser et al., 2013) and other crop models such as STICS (Brisson et al., 2010) or *SiriusQuality* (Martre et al., 2006). Here, we first present the main components of Crop2ML framework. Then we describe the mechanisms of importing and exporting between Crop2ML and PBM platforms. We then discuss our approach and present some perspectives.

2. Crop2ML: a centralized framework for crop model components development and sharing

Crop2ML is a framework for crop model component development, exchange, and reuse between PBM platforms. It is designed following FAIR principles for research software (Lamprecht et al., 2019) to provide:

- *Simplicity*: Model specifications are defined using a declarative language (eXtensible Markup Language [XML]; Bray et al., 2008) with generic concepts shared between PBM platforms and model algorithms are encoded using a minimal language.
- *Transparency*: Models are shared as documented components in a well-defined format (Crop2ML format).
- *Flexibility*: Model units are composed with a shared abstract representation of model structure.
- *Findability*: Model specifications include rich metadata and are assigned a globally unique and persistent identifier for each released version.
- *Reusability*: Model components are transformed into PBM platform-compliant code to support efficient interoperability.
- *Reproducibility*: Model components can be executed and tested regardless of the PBM platforms.
- *Modularity*: Three levels of modularity of models are defined: (single) model units, composite models and package. Package contains model units and composite as well as data. It provides the flexibility to make different compositions based on these models.

We used the principles of Lamprecht et al. (2019) for assessing the FAIR-ness of Crop2ML framework (Supplementary data Table C1).

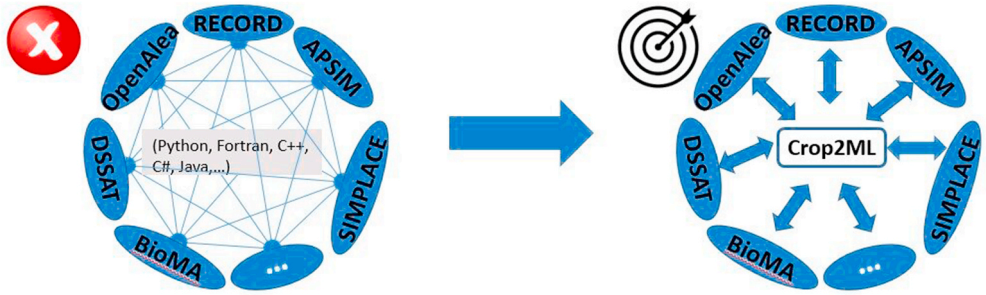


Fig. 1. From a combinatorial to a centralized exchange framework. The schema illustrates the reduction of import export links between platforms in a centralized (right) versus combinatorial exchange framework.

2.1. Design and concepts of Crop2ML model specification

Software modularity is one of the main criteria of reuse. Jones et al. (2001) proposed key elements for modular model structure, which is an essential first step to enhance collaborative modelling effort. Crop2ML follows and extends these principals. In most PBM, the system is decomposed into compartments such as plant parts or soil layers that interact. For each compartment, different processes are described and assembled in components to simulate the response of the compartment. These processes can be subdivided into discrete, explanatory, independent biophysical sub-processes, which could be individually modeled (ModelUnit) or composed (ModelComposite). A modular model structure requires making an objective decomposition of the system to avoid coarse granularity models, which limit reusability. A ModelUnit should not encapsulate alternative assumptions and formalisms, making it easier to test them. In addition, the management of input and output data, such as data access, logging, and file generation, must be managed separately from the implementation of model component. These design principles foster the reuse of components, which are intended to be integrated and simulated with a large variety of input data formats in different PBM platforms. Moreover, to emphasis modularity, the temporal integration loop must be removed from the model process implementation. This makes it possible to reuse the same process with different modeling formalisms or simulation frameworks that manage temporal dynamics of the simulation differently (e.g. different numerical integration techniques).

Crop2ML provides a level of abstraction that enables a shared representation of model components between PBM platforms. A *ModelUnit* is defined with the following descriptive elements (Fig. 2a):

- a model description;
- a list of inputs;
- a list of outputs;
- an initialization step of the state variables;
- a link pointing to the source of the model Algorithm;
- a list of usual mathematical functions;
- a set of unit tests with parameterization shared between modeling platforms.

A *ModelComposite* includes the same elements as a *ModelUnit*. In addition, it contains a list of *Models* and the links between them. (Fig. 2b). However, if control structures are necessary to express the behavior of a *ModelComposite*, the Algorithm can be explicitly provided.

The Crop2ML model specification is based on XML Language. XML is

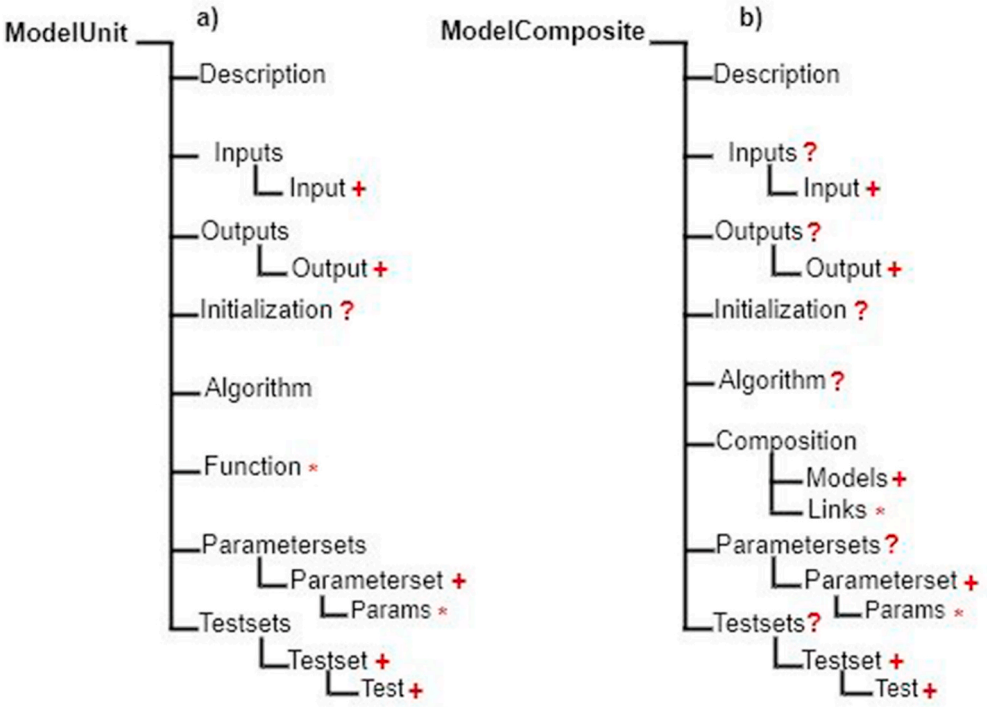


Fig. 2. Crop2ML concepts for model specifications: *ModelUnit* (a) and *ModelComposite* (b). "+", one or more elements; "*", zero or more elements; "?", zero or one element.

a widely used declarative metalanguage for describing or structuring data in a portable format with some descriptive elements. XML format is used in several PBM platforms for template parametrization and model simulation configuration (e.g. APSIM, BioMA, RECORD, Simplace, *SiriusQuality*). This reinforces our choice on this format since the transformation between different XML documents or in any language is relatively straightforward, allows using XML as a bridge between heterogeneous structures and it facilitates collaborative development. Moreover, the use of XML and a formal description of model specifications and their associated metadata facilitate machine readability and model exchange. In the following sections, we describe the concepts of Crop2ML model specifications.

2.1.1. Description

The core description of a Crop2ML model contains the name of the model, an identifier that ensures the provenance of the model and a version number (Fig. 3). The identifier of the model is specified to keep the property of the component. Since PBM are dynamic models, the time step is an important factor that is specified to allow a multi temporal-scale composition. In addition, other elements are described to provide rich metadata, including author names and affiliations, citable and findable references (e.g. doi) and a brief description of the model. The description also includes usage licenses compatible with the model dependencies.

2.1.2. Inputs – outputs

In Crop2ML, a component takes parameter and variable values as inputs and produces variable values as outputs. A variable is a quantity which is given by the context of the experiment (input data) or calculated by the model (output data), while the value of a parameter is an input that can be specified by the modeler within a defined interval. Variables and parameters are distinguished with *input type* attributes and are categorized with *variable category* and *parameter category* attributes, respectively (Table 1).

Crop2ML currently supports four basic types: integer, double, strings and logical. It also supports two collection types: lists and arrays, which contain a sequence of elements of basic types. They are explicitly specified in a *datatype* attribute, similar to the VarInfo type (Donatelli and Rizzoli, 2008). It also provides a common representation of date/-time. The domain of validity of each variable is specified by *min* and *max* attributes. A measurement unit can also be associated to the variables and parameters. Fig. 4 gives an example of inputs and outputs specifications.

Table 1
Category, definition, and example of variables and parameters in Crop2ML.

Input Type	Category	Definition	Example
Variable	State	Characterizes the behavior of a component	Leaf area index, weight of a plant part, canopy temperature
	Rate	Defines the change of one state variable	Transpiration rate, leaf growth rate
	Auxiliary	Intermediate variable computed by an auxiliary function	Dry matter partitioning, shoot number
	Exogenous	Driven variables that do not depend on other variables of the system or component	Mean air temperature, wind speed
Parameter	Constant	Absolute constant	Boltzmann constant
	Soil	Soil parameter	N mineralization constant, maximum rootable soil depth
	Species	Crop parameter with fixed value for a species	Maximum respiration rate
	Genotypic	Crop parameter that can take different values for different genotypes (cultivars)	Phyllochron, grain filling duration

2.1.3. Initialization

State variables of Crop2ML *ModelUnits* and *ModelComposites* are initialized at the start of a simulation and are specified with an *Initialization* element. This element is optional, and the default values of state variables are used if it is omitted. Initialization may also be a function that assigns initial values to state variables. In this case, the *Initialization* element contains the path to the source code of the initialization function.

2.1.4. Algorithm

Algorithm elements link the model specifications with the model algorithm (Fig. 5). A model algorithm describes the behavior of a component in terms of a sequence of inputs, successive rules or actions, conditions and a flow of instructions from inputs to outputs including mathematical expressions. A model algorithm can be implemented in different programming languages. However, Crop2ML proposes to encode the model algorithm in a shared language, CyML (Midingoyi et al., 2020). The CyML source code is the common representation for model algorithm shared by the supported languages and platforms (see Section 2.2.).

```
<ModelUnit modelid="SQ.EnergyBalance.DiffusionLimitedEvaporation"
           name="DiffusionLimitedEvaporation"
           timestep="1"
           version="1.0">
    <Description>
        <Title>DiffusionLimitedEvaporation Model</Title>
        <Authors>Pierre Martre</Authors>
        <Institution>INRA Montpellier</Institution>
        <Reference> From the wheat simulation model SiriusQuality
                    Published in 2009
                    doi:http://dx.doi.org/10.1016/j.eja.2006.04.007
        </Reference>
        <Abstract>the evaporation from the diffusion limited soil when the surface has
            dried sufficiently to provide a significant barrier to water vapor diffusion
        </Abstract>
        <URI> http://www1.clermont.inra.fr/siriusquality/?page_id=547 </URI>
    </Description>
...
</ModelUnit>
```

Fig. 3. Example of a Crop2ML ModelUnit core description.

```
<Inputs>
    <Input name="deficitOnTopLayers" description="deficit On TopLayers"
        variablecategory="auxiliary"  datatype="DOUBLE" default="5341"
        min="0" max="10000" unit="g/(m**2*d)" inputtype="variable"/>
    <Input name="soilDiffusionConstant" description="soil Diffusion Constant"
        parametercategory="soil" datatype="DOUBLE" default="4.2" min="0" max="10"
        unit="dimensionless" inputtype="parameter"/>
</Inputs>
<Outputs>
    <Output name="diffusionLimitedEvaporation"
            description="the evaporation from the diffusion limited soil "
            variablecategory="rate" datatype="DOUBLE" min="0" max="5000"
            unit="g/(m**2*d)"/>
</Outputs>
```

Fig. 4. Example of input and output specifications of a Crop2ML model.

```
<Algorithm  language="Cyml" filename="algo/pyx/diffusionlimitedevaporation.pyx" />
```

Fig. 5. Example of a link to an Algorithm file.

2.1.5. Function

A function is a utility routine that can be called from the model Algorithm or from other functions. It reduces the code length and improves the readability of the encoded algorithm. If a model needs an external function, this function must be declared in the model specification by referencing the path where the function is implemented. A function can also be used for model adaptations such as temporal aggregation or integration, unit conversion to link model components without changing their algorithms. Crop2ML provides a shared library of mathematical functions in different languages such as standard functions, interpolation, or upper and lower bound functions. Modelers can use these functions in their own algorithm, implemented in the CyML language.

2.1.6. Parameter sets and test sets

A Crop2ML model specification includes one or more sets of model parameterizations used for different unit tests (Fig. 6). A parameterization is a set of values assigned to an input parameter of a model. It is described by a name and a description. A unit test in Crop2ML is described in the *Testsets* element and allows comparing estimated and expected outputs values. Several unit tests can be specified. They are described by their *name*, their *description* and the name of *parameters set* associated to them. Each test provides a list of values assigned to each variable and the expected values of the model outputs. A numerical precision could be associated with the *output* of the test to check its validity.

2.1.7. Model links

Model links are specified in a *ModelComposite* and depict how *ModelUnits* or *ModelComposites* are interconnected. A *ModelComposite* is a port graph (Andrei and Kirchner, 2009) that defines a dataflow where nodes are *ModelUnits,* and ports are inputs and outputs of the *ModelUnits*. Edges are oriented links connecting output ports of a source *ModelUnit* to the input ports of a target *ModelUnit* (Fig. 7). Three types of links must be specified: *InternalLink* is the connection between an input of one sub-model and the output of another sub-model, *InputLink* is the connection between an input port of a sub-model and an input port of the composite model, and *OutputLink* is the connection between a *ModelUnit* or *ModelComposite* output port, that can be either a *ModelUnit* or *ModelComposite*, and a *ModelComposite* output port. These connections show the hierarchical structure of a *ModelComposite*. This modeling approach enhances reusability and has been used with success (Wyatt, 1990).

2.2. CyML: the common modelling language of biophysical processes in crop models

We defined a set of common features resulting from the intersection of the programming languages supported by PBM platforms to propose a shared modelling language. A design choice was to define a subset of an existing language that can provide these common features. We needed a widely used high-level language with a low learning curve so that modelers with basic programming skills could efficiently use it. The transformation of a language with dynamic typing can make code

```
<Parametersets>
    <Parameterset name="set1" description="some values in there" >
        <Param name="soilDiffusionConstant">4.2</Param>
    </Parameterset>
</Parametersets>
<Testsets>
    <Testset name="first" parameterset = "set1" description="some values in there" >
        <Test name ="test1">
            <InputValue name="deficitOnTopLayers">5341</InputValue>
            <OutputValue name="diffusionLimitedEvaporation" precision ="3">6605.505</OutputValue>
        </Test>
    </Testset>
</Testsets>
```

Fig. 6. Example of parameterization and unit tests in Crop2ML.

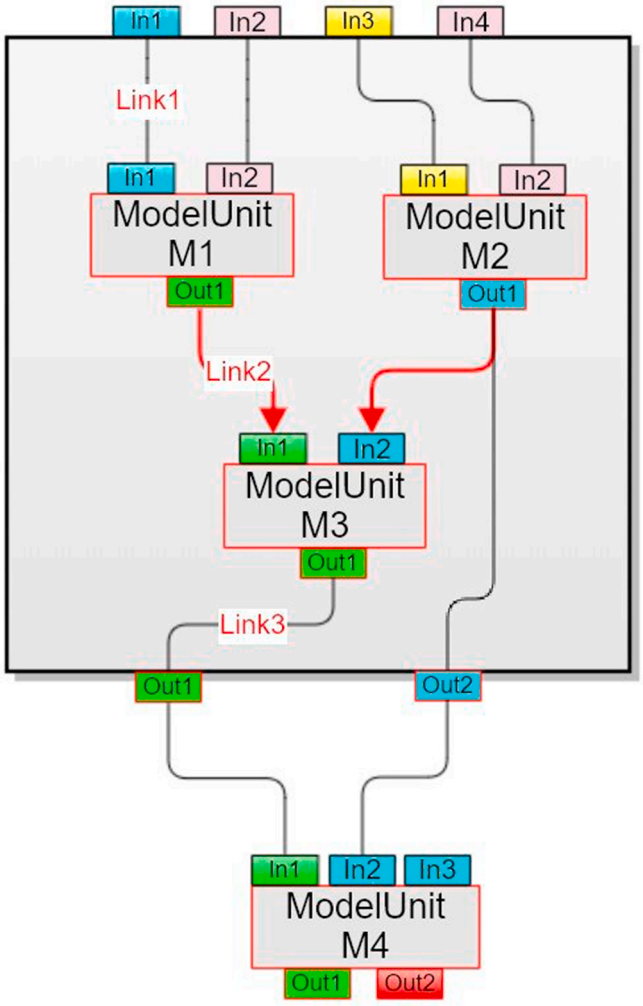


Fig. 7. Graph of a ModelComposite. Three ModelUnits (M1 to M3) are connected to form a first level of composition, which is linked to a fourth ModelUnit (M4). Link1 is an InputLink, Link2 is an InternalLink, and Link3 is an OutputLink. OutputLink is specified to clearly define the outputs of the ModelComposite, which necessarily include all state variables. Each model component has input ports In1, In2, …, and output ports Out1, Out2, …, where 1, 2, …, are internal (local) port numbers.

transformation into programming languages with static typing ambiguous. Therefore, we choose Cython, a high-level language that combines the expressive power of Python language with explicit type declaration of C language (Behnel et al., 2011). It is compiled directly in efficient C code, which improves runtime speed and makes it possible to interact with C, C++ and Fortran source code. However, not all Cython syntax can be directly transformed in all target languages. For instance, the yield statement and anonymous functions are not supported by Fortran. Therefore, we defined CyML (Cython Meta Language), a sub-set of Cython to address the implementation of the model Algorithm (Midingoyi et al., 2020).

We use CyML as a pivot language between various platform languages, which can be mapped to their syntax and semantics. The structure and syntax of CyML, as well as its transformation system to various languages and platforms is detailed in Midingoyi et al. (2020). In brief, CyML supports datatypes defined in the model specification and provides standard mathematical functions and operators. In addition to local variable declaration and assignment statements, control structures are used in the flow of instructions described by the encoded algorithms. These include conditional statements (if, elif and else) to check if a condition is satisfied before addressing part of an Algorithm, sequential statement (for loop) with an incremental index on a data collection, and a repetitive statement (while) used to repeat part of an algorithm while a condition is satisfied. These structures can be nested. To support modular designs and the reuse of *ModelUnits* and functions, CyML provides import mechanisms, which assumes that imported *ModelUnits* or functions are referenced.

Crop2ML framework provides a source-to-source transformation system (CyMLT) which converts CyML source code into procedural (Fortran, Python, C++), object-oriented (Java, C#, C++, Python) and scripting or functional (R, Python) languages (Midingoyi et al., 2020). CyMLT implementation relies on the transformation of the abstract syntax tree (AST) generated from the syntax analysis of the CyML code. The AST is transformed to a self-contained representation of the source code called Abstract Semantic Graph, which is independent of the source language. CyMLT proposes a unique approach to transform the Abstract Semantic Graph into readable source code in many different languages. The generated code is independent from the transformation system and can be run outside the Crop2ML framework. The transformation system integrates model documentation based on the model specification into generated code.

2.3. Crop2ML model package

In the context of large projects and collaborative work, it is useful to define some requirements or standards to facilitate common exchange. Crop2ML provides a logical, standardized but flexible support to facilitate model sharing between modeling platforms through the definition of a directory structure (Fig. 8). This template includes a folder that contains model description and associated algorithms, a repository of source code for each language and modeling platforms. It also includes a folder containing input data for a *ModelComposite* simulation, and a folder containing the unit tests. To save time and avoid omission of mandatory files or folders during package creation, we created a cookiecutter (Roy, 2017) template that automatically generates Crop2ML package templates (https://crop2ml.readthedocs.io/en/latest/user/package.html). It increases model reusability by automatically generating a project that follows shared guidelines. Any *ModelUnit* or *ModelComposite* can be extracted as a stand-alone model from an existing package, tested, reused, or integrated in other *ModelComposite* or package. The notion of package-dependency increases the modularity of Crop2ML and avoids model duplicity.

2.4. Crop2ML model lifecycle management

Crop2ML aims at collaborative model development that supports the entire model lifecycle, including model creation, editing, verification, validation, transformation, composition, and documentation. Therefore, we developed tools and services to support all the steps of a Crop2ML model lifecycle.

2.4.1. Model analysis

Crop2ML models conform to a specific Document Type Definition (DTD) that describes Crop2ML concepts. Model analysis verifies if the model specifications are a well-formed XML document validated by Crop2ML DTD. The analysis of a *ModelComposite* consists of checking model composability through port datatypes and units. Most XML editors can check the validity of an XML document against a DTD but the Crop2ML software environment (see Section 3.2) ensures this.

2.4.2. Model validation

Crop2ML model components can be validated by executing unit tests. It consists of using the parameter and variable values from the model specification to produce unit tests in different languages. Unit tests are generated in Jupyter notebook format, a document format for publishing source codes and reproducible computational workflows that could be executed in the appropriate kernel in Crop2ML software

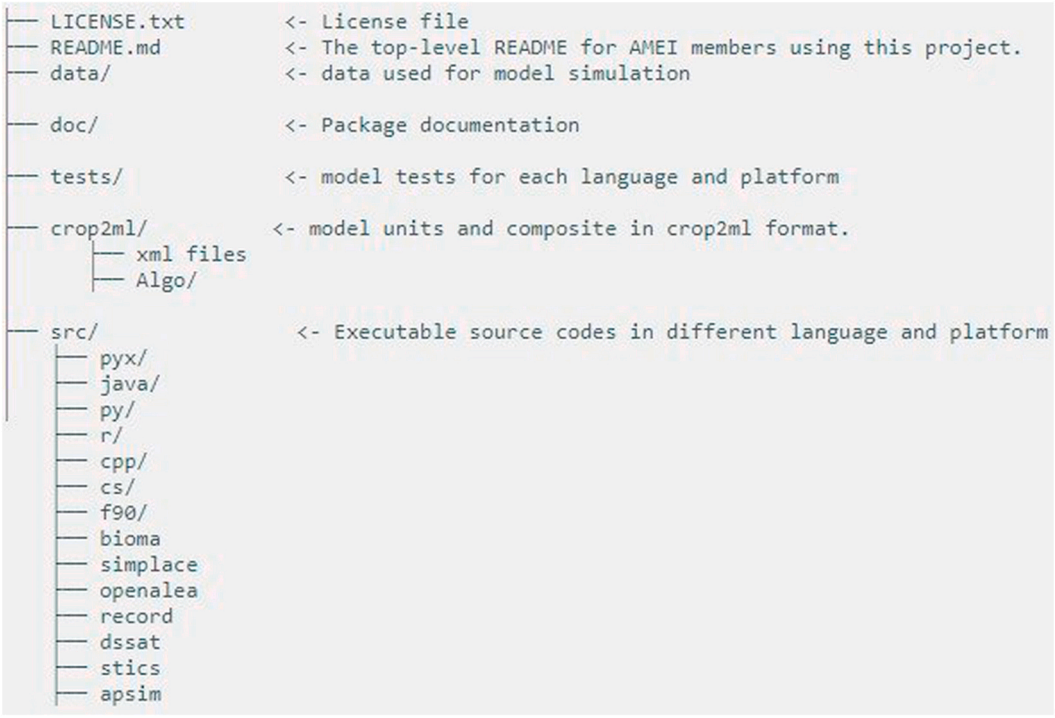


Fig. 8. Tree view of the structure of a Crop2ML model component package.

environment. This format is useful for code and documentation publishing and real-time collaboration when running on a remote server (Kluyver et al., 2016). Unit tests may also be associated with a model publication.

2.4.3. Model transformation

The success of Crop2ML model reuse through a white box approach comes from its ability to generate model components that conform to platform requirements. The transformation of a model component from a platform to another one goes through Crop2ML model representation. It relies on a system of transformation to and from Crop2ML and the platforms.

For some PBM platforms, meta-information of model components are described inside their implementation as documentation. For other platforms meta-information are encoded in a textual or visual programming language. CyMLT generates from Crop2ML model either appropriate documentation or variables and parameters specifications based on the artifacts of the target platforms. In addition, CyMLT generates model component algorithms in various languages. Given a model component provided by a platform, meta-information are extracted by identifying Crop2ML concepts inside the component to generate Crop2ML model meta-information. Moreover, algorithms in CyML are produced to obtain a complete Crop2ML model.

2.4.4. Model documentation

Sharing model knowledge requires detailed information on the model. Crop2ML generates model documentation from the model specification. From the relationships between the *ModelUnits* of a *ModelComposite*, the diagram flow of the *ModelComposite* is generated. It may constitute part of the model documentation and gives a first description of the model component. This allows groups of modelers to understand the model structure and evaluate the component.

3. Crop2ML software environment and tools

3.1. PyCrop2ML: a Python library for Crop2ML

Pycrop2ML is an open, modular, and extensible library developed in Python that implements all the steps of Crop2ML model lifecycle. It is designed to support the current Crop2ML model specifications but can easily be adapted to support future versions. Pycrop2ML can be integrated into other software projects as a plug-in. It allows:

- Verifying a Crop2ML model: This is ensured through a model parser based on the Crop2ML DTD.
- Transforming a Crop2ML *ModelUnit* to source code: PyCrop2ML integrates CyMLT that generates model components that conform to PBM platform requirements.
- Transforming a CyML source code to various languages: Regardless of Crop2ML model specifications, any CyML source code can also be transformed into the target languages. This source code can be used as auxiliary functions for Crop2ML model development.
- Transforming source code to Jupyter notebook format: Each *ModelUnit* source code generated can be translated as a cell of Jupyter notebook, as well as, each unit test, allowing its execution in Crop2ML JupyterLab environment.
- Transforming a Crop2ML *ModelComposite*: A Crop2ML *ModelComposite* provided as a directed graph can be transformed to source code as a sequential order of the submodels.
- Visualizing a *ModelComposite*: Pycrop2ML provides a function to visualize a *ModelComposite* with the links between *ModelUnits* (Fig. 9).

PyCrop2ML is written in Python and can be executed via a command-line interface, inputting either a Crop2ML package or CyML source code, as well as the target language or platform for transformation. Users with no knowledge of the Python language can easily run PyCrop2ML via the command line. The PyCrop2ML library incorporates three crop model components as model examples that can be used to test the different functionalities.

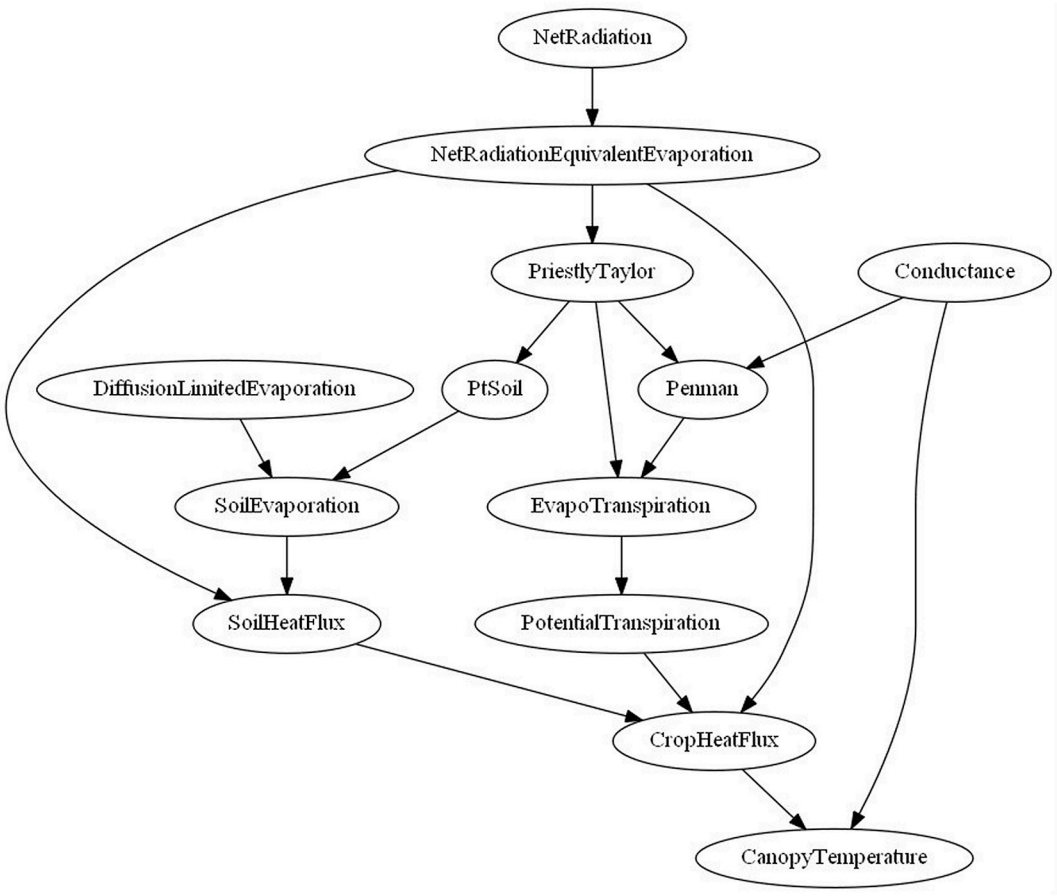


Fig. 9. Visualization of energy balance ModelComposite provided from SiriusQuality wheat model developed with the BioMA platform. Ellipses are ModelUnits and arrows represent the link between two ModelUnits.

3.2. CropMStudio: A JupyterLab environment for Crop2ML model life cycle management

Crop2ML model specifications can be created or edited using any XML editor. However, to fulfil our objective of collaborative model development accessible to modelers with no specific programming skills, we developed a user-friendly interface based on the PyCrop2ML package to manage the lifecycle of Crop2ML model components (Fig. 10). Since Crop2ML models are transformed in different languages, it is useful to execute the unit tests in a single environment. Our solution, named CropMStudio, uses the JupyterLab environment (https://jupyterlab.readthedocs.io), an open-source web application that allows working with code in different languages through different language backends kernels. We installed Python, Java, C#, C++, R and Fortran

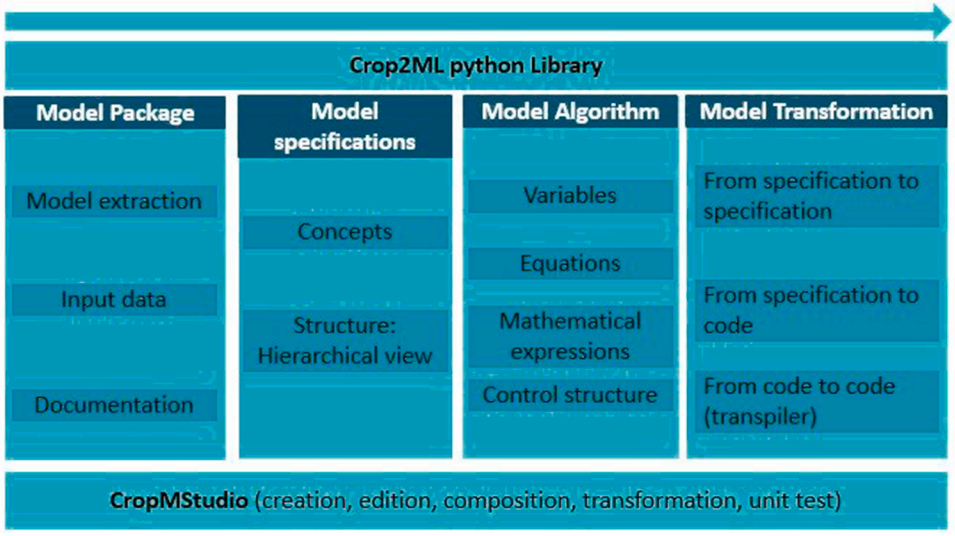


Fig. 10. Schematic representation of the Crop2ML framework showing Crop2ML model lifecycle from the creation of a package to model transformation.

kernels to execute *ModelUnit* tests. The current version of CropMStudio can be accessed through a web browser and run locally like a desktop application. Another motivation to use JupyterLab is to make publication results reproducible in a shared environment based on the capacity to produce interactive and readable code documents (Kluyver et al., 2016).

4. Interoperability between various simulation platforms

The interoperability between simulation platforms is based on two transformation processes (import and export) via Crop2ML. The import process consists of transforming any platform model component to Crop2ML model. The export process consists of transforming Crop2ML models to any platform. Detailed descriptions of the import/export mechanisms in five widely used platforms with different architectures (BioMA, DSSAT, Record, OpenAlea, SIMPLACE) are provided in Supplementary data (Appendix C). Table 2 summarizes the interoperability of model components between these platforms. Platforms are based on various programming languages, which requires the definition of transformation rules between CyML and various languages including C# (BioMA), Java (Simplace), C++ (Record), Python (OpenAlea) and Fortran (DSSAT) in both directions. We identified the levels of granularity of modeling processes that correspond to Crop2ML concepts such as *ModelUnit* and *ModelComposite* in each platform. We also considered how documentation or model specifications are described in these platforms.

The export process, from Crop2ML to platforms, is automatically done in BioMA, OpenAlea and Simplace. The modularity principle in BioMA matches Crop2ML, which allows associating simple and composite BioMA strategies with Crop2ML *ModelUnit* and *ModelComposite*, respectively. Moreover, all the Crop2ML elements are well translated into the VarInfo type attributes (Donatelli and Rizzoli, 2008), and Crop2ML model algorithms are transformed to a method of a strategy class that takes generated domain classes as inputs. OpenAlea relies on two families of approaches: component-based architecture and scientific workflows. Thus, Crop2ML exports *ModelUnits* as OpenAlea components and *ModelComposite* as OpenAlea workflows. ModelComposite can thus be visualized and edited using VisuAlea, the visual programming environment in OpenAlea. Widgets of *ModelUnit* are automatically generated based on the type of inputs that is mapped to an OpenAlea interface. Simplace is based on the concept of software units, called SimComponents as the smallest building blocks that map with *ModelUnits*. *ModelComposite* are converted into a combination of SimComponents (SimComponentGroup). Variables and parameters descriptions are automatically included in the SimComponents descriptive part.

In DSSAT and Record the export process is many automatic but some aspects need to be done manually. In DSSAT, Crop2ML transformation system generates a submodule in Fortran 90 for each *ModelUnit*. It also generates a sequence of submodules calls for composite models. One issue that makes this transformation not completely automatic is that Crop2ML does not manage the handling of input and output files. Therefore, it requires to manually add the input and output methods into the generated submodules. The concepts of atomic and coupled models in Record are mapped with those of Crop2ML. Thus, atomic model classes are generated in C++ to correspond to *ModelUnits*. However, the configuration and simulation file (VPZ) representing the *ModelComposite* is manually completed with further information such as the description of simulation result files.

The import process (from simulation platforms to Crop2ML) is only partially automatic. Platform tools produce automatically the meta-information in Crop2ML format but algorithms are manually converted into the CyML language that leads to a semi-automatic transformation. A complete automatic transformation would require the implementation of source-to-source transformation from platforms' language into CyML. In BioMA, VarInfo attributes are extracted from BioMA strategies to produce Crop2ML model meta-information. The process of automatically retrieving the *estimate* method to produce model Algorithm in CyML is not implemented yet. The description of component in OpenAlea is very generic compared to Crop2ML concepts. Although OpenAlea is mainly built for Functional Structural Plant Modeling (FSPM) application, there is no plant domain specific description associated with inputs and outputs such as units, categories of variables and parameters. Thus, the generation of model description in Crop2ML is partial. It requires further description of components that can be provided in documentation or by extending OpenAlea concepts. Like BioMA, the SimComponent specific descriptors in Simplace allows generating *ModelUnits* meta-information. The *process* method (algorithm) is currently translated manually in CyML. Links between the different SimComponents (Unit) stored in the SimComponentGroup (Composition) are automatically exported to the Crop2ML structure. However, there is a loss of information since when a ModelUnit is activated or ignored it is not transferred to the Crop2ML structure. In DSSAT, unlike in the other platforms, the description of physiological processes is provided as documentation in submodules and it is not fully complete with respect to Crop2ML specifications. Inputs and outputs variables and their descriptions, units can be clearly identified, based on systematic platform guidelines. DSSAT submodules contain specific platform variables such as control variables that need to be removed to produce CyML model algorithms. In Record, as in DSSAT, there is no explicit specification of a model. The documentation of a model within its associated C++ class is used to generate partial *ModelUnit* meta-information. The parsing of the VPZ file, that contains the structure of composite models in Record, is used to generate a *ModelComposite*. However, it is not possible to represent retroaction loops in Crop2ML as it is done in Record with coupled models.

In order to illustrate Crop2ML concepts and transformation results, a phenology and an energy balance models are used. Phenology, the timing of crop development is the heart of most crop growth models and is an essential component of most crop modeling platforms. The energy balance model involves interconnected components that allows estimating canopy temperature, evapotranspiration, and heat transfer between the canopy and the air. These processes are implemented as BioMA standalone components (Manceau and Martre, 2018) of the wheat PBM *SiriusQuality* (He et al., 2012; Martre et al., 2006). The two

Table 2
Import and Export processes between Crop2ML and PBM platforms. A, automatic; P, partially automatic; M, manual.

PBM Platforms	From Crop2ML to PBM platforms				From PBM platforms to Crop2ML			
	A	P	M	To be completed	A	P	M	To be automatized
BioMA								Source transformation from C# to CyML
DSSAT				Integration of I/O				Source transformation from Fortran to CyML Remove I/O More variable description
Record				Complete VPZ file				Source transformation from C++ to CyML More variable description
OpenAlea								Source transformation from Python to CyML More variable description
SIMPLACE								Source transformation from SIMLPACE Java to CyML

Table 3
Declaration of the inputs and Algorithm of a Crop2ML ModelUnit of the Penman-Monteith evapotranspiration model and the equivalent source code generated by CyMLT for Record, BioMA, and DSSAT. The declaration of a single variable is given as an example.

Framework or platform/language	Variable declaration	Algorithm

Crop2ML/CyML	`<Input name="rhoDensityAir" description="Density of air" parametercategory="constant" datatype="DOUBLE" min="0" max="10000" default="1.225" unit="kg/m**3" inputtype="parameter"/>`	`def model_penman(float evapoTranspirationPriestlyTaylor=449.367, float hslope=0.584, float VPDair=2.19, float psychrometricConstant=0.66, float Alpha=1.5, float lambdaV=2.454, float rhoDensityAir=1.225, float specificHeatCapacityAir=0.00101, float conductance=598.685):`
		`    """`
		
		`    """`
		`    cdef float evapoTranspirationPenman`
		`    evapoTranspirationPenman = evapoTranspirationPriestlyTaylor / Alpha + 1000.0 * ((rhoDensityAir * specificHeatCapacityAir * VPDair * conductance) / (lambdaV * (hslope + psychrometricConstant)))`
		`    return  evapoTranspirationPenman`

Record/C++	`//Model parameters`
	`/**`
	`* @brief Density of air (kg/m**3)`
	`*/`
	`double rhoDensityAir;`
	`//...`
		`virtual void compute(const vd::Time& /*time*/)`
		`{`
		`    evapoTranspirationPenman = evapoTranspirationPriestlyTaylor() / Alpha + (1000.0d * (rhoDensityAir * specificHeatCapacityAir * VPDair() * conductance() / (lambdaV * (hslope() + psychrometricConstant))));`
		`}`

BioMA/C#	`VarInfo v4 = new VarInfo();`
	`v4.DefaultValue = 1.225;`
	`v4.Description = "Density of air";`
	`v4.Id = 0;`
	`v4.MaxValue = 10000;`
	`v4.MinValue = 0;`
	`v4.Name = "rhoDensityAir";`
	`v4.Size = 1;`
	`v4.Units = "kg/m**3";`
		`private void  CalculateModel(SiriusQualityEnergybalance.EnergybalanceState s, SiriusQualityEnergybalance.EnergybalanceState s1, SiriusQualityEnergybalance.EnergybalanceRate r, SiriusQualityEnergybalance.EnergybalanceAuxiliary a, SiriusQualityEnergybalance.EnergybalanceExogenous ex)`
		`{`
		`    double evapoTranspirationPriestlyTaylor = r.evapoTranspirationPriestlyTaylor;`
		`    double hslope = a.hslope;`
		`    double VPDair = a.VPDair;`
		`    double conductance = s.conductance;`
		`    double evapoTranspirationPenman;`
		`    evapoTranspirationPenman = evapoTranspirationPriestlyTaylor / Alpha + (1000.0d * (rhoDensityAir * specificHeatCapacityAir * VPDair * conductance / (lambdaV * (hslope + psychrometricConstant))));`
		`    r.evapoTranspirationPenman = evapoTranspirationPenman;`
		`}`

DSSAT/Fortran		

(*continued on next page*)

Table 3 (*continued*)

Framework or platform/language	Variable declaration	Algorithm
	* name: rhoDensityAir ** description : Density of air ** datatype : DOUBLE ** min : 0 ** max : 10000 ** default : 1.225 ** unit : kg/m**3	SUBROUTINE model_penman(evapoTranspirationPriestlyTaylor, ...) REAL, INTENT(IN) :: evapoTranspirationPriestlyTaylor ... evapoTranspirationPenman = evapoTranspirationPriestlyTaylor / Alpha + & (1000.0 * (rhoDensityAir * specificHeatCapacityAir * VPDair * & conductance / (lambdaV * (hslope + psychrometricConstant)))) END SUBROUTINE model_penman

components were converted into Crop2ML packages, and then automatically translated into different languages and model components that conform to different PBM platforms. These packages are presented in Appendixes A and B. In Table 3 we illustrate how to represent a parameter and an Algorithm in a Crop2ML ModelUnit and its translation with CyMLT in Record, BioMA, and DSSAT. The implementations of the model differ between the platforms. For instance, DSSAT defines a subroutine with all the variables as argument, Record defines a class method (compute) with the variables as attributes of the class and uses specific operator "()" to manage temporal variables, while BioMA defines a class method (CalculateModel) that takes as argument data structures implementing each category of variables (state, rate, auxiliary, exogenous). The aim of model transformation is to provide to the platforms alternative model components that could easily replace their corresponding components to analyze the effects of new hypotheses into their modeling solutions.

The sequence of ModelUnits that compose a Crop2ML ModelComposite is formally modeled as a directed acyclic graph. This means that there is no feedback loop or retroaction at a given time step, instead they are usually represented by a cycle in the *ModelComposite*. Alternatively, a state variable can be defined explicitly as two variables with respect to the current and the previous time. Thus, a composite model may take as input a state variable at previous time and a state variable at current time as output, making implicitly a loop with respect to time advance. Another way to represent feedback inside a time step is to associate an explicit Algorithm to the ModelComposite that defines how to run it. However, this feature is not supported by two simulation platforms (OpenAlea and RECORD).

5. Discussion

The Crop2ML framework enables a user to exchange and reuse biophysical components between various PBM platforms through shared declarative specifications. The use of a minimal language to describe the model Algorithm once and the transformation system facilitates reuse of models' components. *ModelUnits* and *ModelComposite* can be accessed and composed following a white box approach. Therefore, the Crop2ML approach greatly increases the ability of modelers to share their algorithms. The protocol will allow modelers to borrow components easily and will facilitate their intercomparison and improvement in different PBM platforms.

5.1. How does Crop2ML address model reuse compared to other initiatives?

Some initiatives addressed model reuse by providing multi-scale and multi-language integrative frameworks such as *Crops in silico* (Marshall-Colon et al., 2017) the Open Modeling Foundation OpenMI (Buahin and Horsburgh, 2018). These frameworks can compose and simulate heterogeneous models provided by different frameworks through a communication interface. The model components are often wrapped and are represented as black-box components. All state variables are not always exposed as model outputs, which may limit their integration in an existing modeling solution. Therefore, these frameworks enhance model reuse in their own environment but they do not address reusability with other PBM platforms. Many existing PBM platforms do not support the coupling of models written in multiple languages (e.g. BioMA, APSIM next generation).

Donatelli and Rizzoli (2008) proposed a design pattern for platform-independent model components to enhance modularity and to facilitate model reuse in several PBM platforms via simple wrappers. However, this approach fixes the structure of the components. The lack of specification or meta-information makes the reuse of model components between platforms difficult. Even in component-based systems, explicit information about the component itself and its inputs and outputs (types, units and boundary conditions) are required to ensure a

syntactic composability and to meet the specificities of the platforms. Moreover, the knowledge of the structure underlying the source code of a component is also required to systematically extract model information (variables and algorithms) for their transformation and integration in different platforms. We thus argue that model component reuse is improved if it is supported by model specification. Crop2ML defines an abstract representation of model design shared by PBM platforms through some shared concepts enriching or extending those proposed by Athanasiadis et al. (2011) with other attributes and a formal and shared description of unit tests. We included unit tests in Crop2ML specifications to ensure model transformation validation and some imperative constructs for model dynamics.

Several initiatives have used declarative modeling to describe model specifications and address model reuse issues. The approach proposed by Villa et al. (2006) is similar to ours but it is limited to models where the dynamics of the modeled processes is represented by simple mathematical expressions without control structures, which does not match crop modeling context. Hucka et al. (2003) used MathML (Ausbrooks et al., 2003) to express interactions between variables through mathematical formalisms well defined in the systems biology community. This approach is similar to that of Rizzoli et al. (2008) and is useful when processes are governed by differential equations. However, in the PBM context, simulation platforms use algorithms to describe processes rather than mathematical formalisms with differential equations. Moreover, in PBM, variables that drive the system are temporal series that change the behavior of the system at discrete time. This does not require finding a general solution of recurrent equations used in crop models but rather estimating at each time step the state variables of the system.

Automated model transformation is a core aspect of model-driven development (Cuadrado and Molina, 2007). It uses Model-Driven Engineering (MDE) principles based on metamodeling concepts. Crop2ML is in line with MDE. It defines structured concepts representing its metamodel, with which all Crop2ML models are conform, and a model transformation to generate PBM platforms' components. Model Driven Architecture (Brown, 2004) is a framework of MDE that provides several standard languages (e.g., ATL, QVT, ETL, Henshin, VIATRA, and Stratego) for model transformation (Jouault and Kurtev, 2006; Kurtev et al., 2006). Crop2ML is based on a transformation process through a set of refinement of models and code with some extensible rules defined as templates in Python. Most MDE approaches allow model to model or model to code transformation where a model represents the specification in our case. However, the use of transformation language standards was inappropriate in our context to unify transformation process towards many languages with different paradigms (Bucchiarone et al., 2020). Crop2ML produces code in a target language but also adapts the code to fit with PBM platform specificities. To our knowledge, model transformation languages in MDE do not support code generation in multiple languages with extended features in the same environment.

5.2. Connecting Crop2ML to PBM platforms

Given that Crop2ML datatypes do not handle complex data structures other than arrays and lists, some compromises or transformation should be made to the import-export process on the platform side with respect to handling other data structures used in platforms. As an example, BioMA provides the Dictionary data type that is a set of keys associated with values to represent either input or output variables. This data type is not handled by Crop2ML, and by most PBM platforms. As an alternative, Dictionaries can be expressed in Crop2ML as two list datatype variables that represent keys and values of the dictionary.

The simulation Algorithm defining the feedback loop is explicitly described as control flow in some platforms (e.g. BioMA) but this is not the case in other platforms (e.g. Record, where the VPZ file representing the simulation model file is handled by the simulation engine VLE). Different simulation engines are based on different models of computation used by the platforms such as dataflow (e.g. OpenAlea), DEVS simulation (e.g. Record), control flow (e.g. BioMA, DSSAT, and Simplace). These models of computation are used to coordinate the execution of the model. The current version of Crop2ML framework does not take into account the specificities of simulation engines and addresses components which can be sequentially composed.

The Crop2ML transformation system is designed to support the specificities of the target PBM platforms. However, the semantic of a Crop2ML model is based on shared concepts to describe at a high level a biophysical process by a discrete-time model. There is no semantic reason to support the description of each instance of the concepts. For example, since we have not defined a convention to name process variables, the integration of a Crop2ML component into a PBM modeling solution requires adapting the name of its variables. In the future, we could annotate Crop2ML models to add semantic information to make semantic links between any Crop2ML model variables or parameters with those of model components of PBM platforms. This will also allow a semantic composability of Crop2ML models instead of a syntactic composability that analyzes whether the pair of variables to be linked are compatible. However, this would require the crop modeling community to agree on shared semantics and ontologies of crop model variables and parameter representations. Until now this has been a real

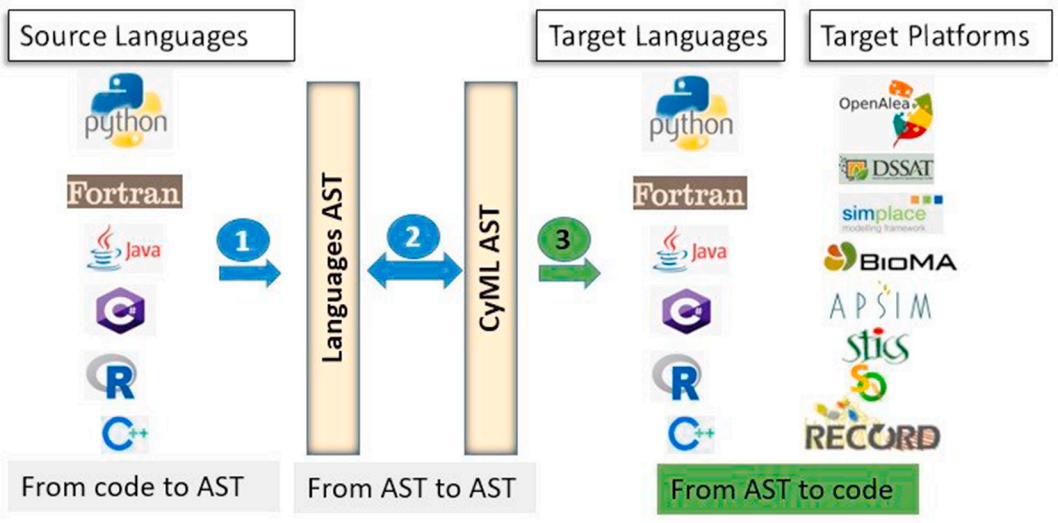


Fig. 11. Schema illustrating CyML transformation extensibility to support bidirectional source transformation.

challenge as the crop modeling community has not be too keen on adapting standards (White et al., 2013). In addition, to facilitate the exchange and reuse of model components, semantic descriptions of model variables and parameters would facilitate the linking of crop models to plant phenomics data (Neveu et al., 2018).

We were able to achieve fully-automatic export of Crop2ML model to several PBM platforms. The import process into Crop2ML is more mixed regarding the overall differences between PBM platforms. It is much easier to start with concepts shared and reused by PBM platforms than to start from divergent views of model representations to achieve a particular result. Some PBM platforms need to extend their concepts for model specification or to provide a rich model documentation in order to produce complete Crop2ML model specifications. This reveals the need of a good level of abstraction to represent a model in various PBM platforms. The higher the level of abstraction, the further the description moves away from the platforms and the less easy it is to understand. On the other hand, if the level of abstraction is too low, it is not always possible to represent all features of the models present in the platforms.

5.3. Future developments

A common model repository infrastructure is essential for efficient model exchange (Glont et al., 2018; Lloyd et al., 2008). Currently, Crop2ML model components are stored in Github repositories. We aim to provide a Crop2ML model repository to store models in a shared format to make them easily accessible and reusable by the plant and crop modeling community. This repository should aim at hosting alternative biophysical processes. It will help modelers to operate on multiple model components, compare processes, or evaluate the impact of the integration of alternative models of biophysical processes in crop models. The success of the Crop2ML repository requires that the community gives access to their models by feeding the repository, which will be curated by the AMEI consortium to avoid error propagation.

Crop2ML has some limitations, which can be addressed in the next versions, either by extending the model specifications with shared concepts or by adapting the target PBM platforms to Crop2ML specification and language. It is an ongoing, long-term activity, to satisfy platform requirements and facilitate Crop2ML model life-cycle management to make Crop2ML a standard for the plant and crop modeling community.

The transformation of a model component of a PBM platform into a Crop2ML package requires rewriting the model algorithms in the CyML language. This limit is currently being addressed by extending the CyML transpiler to a bidirectional transpiler. Thereby, PBM platforms could provide model algorithms in the language they use and the extended CyMLT will transform them in CyML and target languages used by other PBM platforms (Fig. 11). This is a two-step process. First, the model algorithms in the language of the source PBM platform will be parsed and an AST will be generated. Second, the rules for transforming this AST into the CyML AST will be applied. The second step will reuse the CyML transformation tool developed by Midingoyi et al. (2020) to produce model algorithms compatible with other languages and PBM platforms.

Other future developments of Crop2ML include:

- Enhance Crop2ML model repositories with model annotation to link publications to models for reproducibility;
- Add unit checks and conversions in Crop2ML to improve model validity;
- Define a methodology to link Crop2ML with plant structure representation for multiscale viewing and analysis;
- Define and implement an ontology of crop model variable and parameters to allow better Crop2ML model interpretation and improve transformation between PBM platforms and the integration of model component in complex modeling solutions.
- Extend Crop2MLab prototype by including bidirectional transformation and the creation of a web interface on a remote server in order to give users the possibility to handle Crop2ML model lifecycle without local installation.

6. Conclusion

At the interface between modeling and software engineering, this paper addresses plant and crop model component reuse by proposing the Crop2ML framework. Despite all the differences between PBM platforms, some common features can be identified that enabled model representation regardless of the platforms' specificities. Crop2ML provides structured concepts to support the definition of *ModelUnit* and *ModelComposite* and allows their transformation to make them compatible with PBM platforms at implementation level. Therefore, Crop2ML defines a new unified crop model representation that considers the abstraction of PBM component features in several PBM platforms. Moreover, Crop2ML uses a domain specific language to describe biophysical processes and auxiliary functions to represent model dynamics based on a subset of the Cython language, which can then be automatically transformed into different target languages. Crop2ML proposes an open framework to manage all the steps of model lifecycle.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

C.M. was supported through a PhD scholarship from the French National Research Agency under the Investments for the Future Program, referred as ANR-16-CONV-0004 and INRAE Divisions AgroEcoSystem and NUM. P.M. acknowledges the support of INRAE Division AgroEcoSystem through the *Modélisation du fonctionnemnet des Peuplements Cultivés* (MFPC) network. The authors thank Dr. Loic Manceau (INRA, UMR LEPSE) for discussions and its help to translate the wheat phenology BioMA component of *SiriusQuality* in Crop2ML.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.envsoft.2021.105055.

References

Andrei, O., Kirchner, H., 2009. A port graph calculus for autonomic computing and invariant verification. TERMGRAPH 2009, 5th international workshop on computing with terms and graphs, satellite event of ETAPS 2009. https://hal.inria.fr/inria-00418560.

Asseng, S., Ewert, F., Rosenzweig, C., Jones, J.W., Hatfield, J.L., Ruane, A.C., Boote, K.J., Thorburn, P.J., Rötter, R.P., Cammarano, D., Brisson, N., Basso, B., Martre, P., Aggarwal, P.K., Angulo, C., Bertuzzi, P., Biernath, C., Challinor, A.J., Doltra, J., Wolf, J., 2013. Uncertainty in simulating wheat yields under climate change. Nat. Clim. Change 3 (9), 827–832. https://doi.org/10.1038/nclimate1916.

Athanasiadis, I.N., Rizzoli, A.E., Donatelli, M., Carlini, L., 2011. Enriching environmental software model interfaces through ontology-based tools. Int. J. Appl. Syst. Stud. 4 (1–2), 94–105. https://doi.org/10.1504/IJASS.2011.042205.

Athanasiadis, I.N., Villa, F., 2013. A roadmap to domain specific programming languages for environmental modeling: key requirements and concepts. DSM 2013 - Proc. 2013 ACM Workshop on Domain-Specific Model. 27–32. https://doi.org/10.1145/2541928.2541934.

Ausbrooks, R., Buswell, S., Carlisle, D., Dalmas, S., Devitt, S., Diaz, A., Frou-mentin, M., Hunter, R., Ion, P., Kohlhase, M., Miner, R., Poppelier, N., Smith, B., Soiffer, N., Sutor, R., Watt, S., 2003. Mathematical Markup Language (MathML) version 2 . 0 (second edition). http://www.w3.org/TR/MathML2/.

Basso, B., Cammarano, D., Carfagna, E., 2013. Review of crop yield forecasting methods and early warning systems. The first meeting of the scientific advisory committee of the global strategy to improve agricultural and rural statistics, pp. 1–56. https://doi.org/10.1017/CBO9781107415324.004.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S., Smith, K., 2011. Cython: the best of both worlds. Comput. Sci. Eng. 13 (2), 31–39. https://doi.org/10.1109/MCSE.2010.118.

Bergez, J.E., Chabrier, P., Gary, C., Jeuffroy, M.H., Makowski, D., Quesnel, G., Ramat, E., Raynal, H., Rousse, N., Wallach, D., Debaeke, P., Durand, P., Duru, M., Dury, J., Faverdin, P., Gascuel-Odoux, C., Garcia, F., 2013. An open platform to build, evaluate and simulate integrated models of farming and agro-ecosystems. Environ. Model. Software 39, 39–49. https://doi.org/10.1016/j.envsoft.2012.03.011.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., 2008. Extensible Markup Language (XML) 1.0, fifth ed. http://www.w3.org/TR/REC-xml/.

Brisson, N., Launay, M., Mary, B., Beaudoin, N., 2010. Conceptual Basis, Formalisations and Parameterization of the Stics Crop Model. Editons Quae. http://www.quae.com/en/r1291-conceptual-basis-formalisations-and-parameterization-of-the-stics-crop-model.html.

Brown, A.W., 2004. Model driven architecture: principles and practice. Software Syst. Model 314–327. https://doi.org/10.1007/s10270-004-0061-2.

Buahin, C.A., Horsburgh, J.S., 2018. Advancing the open modeling interface (OpenMI) for integrated water resources modeling. Environ. Model. Software 108 (April), 133–153. https://doi.org/10.1016/j.envsoft.2018.07.015.

Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A., 2020. Grand challenges in model-driven engineering: an analysis of the state of the research. Software Syst. Model 19 (1), 5–13. https://doi.org/10.1007/s10270-019-00773-6.

Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., Hinsen, K., Larmande, P., Bras, Y. Le, Lemoine, F., Mareuil, F., Ménager, H., Pradal, C., Blanchet, C., 2017. Scientific workflows for computational reproducibility in the life sciences: status, challenges and opportunities. Future Generat. Comput. Syst. 75, 284–298. https://doi.org/10.1016/j.future.2017.01.012.

Cuadrado, J.S., Molina, J.G., 2007. Building domain-specific languages for model-driven development. IEEE Software 24 (5), 48–55. https://doi.org/10.1109/MS.2007.135.

Cuellar, A.A., Lloyd, C.M., Nielsen, P.F., Bullivant, D.P., Nickerson, D.P., Hunter, P.J., 2003. An overview of CellML 1.1, a biological model description language. Simulation 79 (12), 740–747. https://doi.org/10.1177/0037549703040939.

Donatelli, M., Bregaglio, S., Confalonieri, R., De Mascellis, R., Acutis, M., 2014. A generic framework for evaluating hybrid models by reuse and composition - a case study on soil temperature simulation. Environ. Model. Software 62, 478–486. https://doi.org/10.1016/j.envsoft.2014.04.011.

Donatelli, M., Rizzoli, A.E., 2008. A design for framework-independent model components of biophysical systems. 4th biennial meeting of international congress on environmental modelling and software: integrating sciences and information technology for environmental assessment and decision making. IEMSs 2008 2 (July), 727–734. http://www.scopus.com/inward/record.url?eid=2-s2.0-70349563625&partnerID=40&md5=251da64e55d9bed564ab136bf25897d7.

Donatelli, M., Russell, G., Rizzoli, A.E., Acutis, M., Adam, M., Athanasiadis, I.N., Balderacchi, M., Bechini, L., Belhouchette, H., Bellocchi, G., Bergez, J.-E., Botta, M., Braudeau, E., Bregaglio, S., Carlini, L., Casellas, E., Celette, F., Ceotto, E., Charron-Moirez, M.H., Zerourou, A., 2010. A component-based framework for simulating agricultural production and externalities. In: Environmental and Agricultural Modeling. Springer Netherlands, pp. 63–108. https://doi.org/10.1007/978-90-481-3619-3_4.

Fernique, P., Pradal, C., 2018. Auto WIG: automatic generation of python bindings for C ++ libraries. PeerJ Comput. Sci. (4), e149. https://doi.org/10.7717/peerj-cs.149, 2018.

Gaiser, T., Perkons, U., Küpper, P.M., Kautz, T., Uteau-Puschmann, D., Ewert, F., Enders, A., Krauss, G., 2013. Modeling biopore effects on root growth and biomass production on soils with pronounced sub-soil clay accumulation. Ecol. Model. 256, 6–15. https://doi.org/10.1016/j.ecolmodel.2013.02.016.

Glont, M., Nguyen, T.V.N., Graesslin, M., Hälke, R., Ali, R., Schramm, J., Wimalaratne, S. M., Kothamachu, V.B., Rodriguez, N., Swat, M.J., Eils, J., Eils, R., Laibe, C., Malik-Sheriff, R.S., Chelliah, V., Le Novère, N., Hermjakob, H., 2018. BioModels: expanding horizons to include more modelling approaches and formats. Nucleic Acids Res. 46 (D1), D1248–D1253. https://doi.org/10.1093/nar/gkx1023.

He, J., Le Gouis, J., Stratonovitch, P., Allard, V., Gaju, O., Heumez, E., Orford, S., Griffiths, S., Snape, J.W., Foulkes, M.J., Semenov, M.A., Martre, P., 2012. Simulation of environmental and genotypic variations of final leaf number and anthesis date for wheat. Eur. J. Agron. 42, 22–33. https://doi.org/10.1016/j.eja.2011.11.002.

Hinsen, K., 2016. Scientific notations for the digital era. Phys. Soc. 1–27.

Holzworth, D.P., Huth, N.I., Fainges, J., Brown, H., Zurcher, E., Cichota, R., Verrall, S., Herrmann, N.I., Zheng, B., Snow, V., 2018. APSIM Next Generation: overcoming challenges in modernising a farming systems model. Environ. Model. Software 103, 43–51. https://doi.org/10.1016/j.envsoft.2018.02.002.

Holzworth, D.P., Snow, V., Janssen, S., Athanasiadis, I.N., Donatelli, M., Hoogenboom, G., White, J.W., Thorburn, P., 2014. Agricultural production systems modelling and software: current status and future prospects. Environ. Model. Software 72, 276–286. https://doi.org/10.1016/j.envsoft.2014.12.013.

Hoogenboom, G., Porter, C.H., Boote, K.J., Shelia, V., Wilkens, P.W., Singh, U., White, J. W., Asseng, S., Lizaso, J.I., Moreno, L.P., Pavan, W., Ogoshi, R., Hunt, L.A., Tsuji, G. Y., Jones, J.W., 2019. The DSSAT crop modeling ecosystem. In: Boote, K.J. (Ed.), Advances in Crop Modeling for a Sustainable Agriculture. Burleigh Dodds Science Publishing, Cambridge, United Kingdom, pp. 173–216. https://doi.org/10.19103/AS.2019.0061.10.

Hucka, M., Bergmann, F., Hoops, S., Keating, S., Sahle, S., Schaff, J., Smith, L., Wilkinson, D., 2010. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. the WorldWide Web. Available via. http://sbml.org/Documents/Specifications.

Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., Arkin, A.P., Bornstein, B.J., Bray, D., Cornish-Bowden, A., Cuellar, A.A., Dronov, S., Gilles, E.D., Ginkel, M., Gor, V., Goryanin, I.I., Hedley, W.J., Hodgman, T.C., Hofmeyr, J.H., Wang, J., 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19 (4), 524–531. https://doi.org/10.1093/bioinformatics/btg015.

Jones, J.W., Antle, J.M., Basso, B., Boote, K.J., Conant, R.T., Foster, I., Godfray, H.C.J., Herrero, M., Howitt, R.E., Janssen, S., Keating, B.A., Munoz-Carpena, R., Porter, C. H., Rosenzweig, C., Wheeler, T.R., 2017. Brief history of agricultural systems modeling. Agric. Syst. 155 (June), 240–254. https://doi.org/10.1016/j.agsy.2016.05.014.

Jones, J.W., Hoogenboom, G., Porter, C.H., Boote, K.J., Batchelor, W.D., Hunt, L.A., Wilkens, P.W., Singh, U., Gijsman, A.J., Ritchie, J.T., 2003. The DSSAT cropping system model. Eur. J. Agron. 18 https://doi.org/10.1016/S1161-0301(02)00107-7. Issues 3–4).

Jones, J.W., Keating, B.A., Porter, C.H., 2001. Approaches to modular model development. Agric. Syst. 70 (2–3), 421–443. https://doi.org/10.1016/S0308-521X(01)00054-3.

Jouault, F., Kurtev, I., 2006. Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference, vol. 3844. Issue OCTOBER, pp. 128–138. https://doi.org/10.1007/11663430_14.

Kluyver, T., Ragan-kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., 2016. Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows. *Positioning and Power in*. Academic Publishing: Players, Agents and Agendas, pp. 87–90. https://doi.org/10.3233/978-1-61499-649-1-87.

Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P., 2006. Model-based DSL Frameworks. Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications. OOPSLA '06, p. 602. https://doi.org/10.1145/1176617.1176632, 2006.

Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., van de Sandt, S., Ison, J., Martinez, P.A., McQuilton, P., Valencia, A., Harrow, J., Psomopoulos, F., Gelpi, J.L., Chue Hong, N., Goble, C., Capella-Gutierrez, S., 2019. Towards FAIR principles for research software. Data Sci. 3 (1), 37–59. https://doi.org/10.3233/ds-190026.

Le Franc, Y., Davison, A.P., Gleeson, P., Imam, F.T., Kriener, B., Larson, S.D., Ray, S., Schwabe, L., Hill, S., De Schutter, E., 2012. Computational Neuroscience Ontology: a new tool to provide semantic meaning to your models. BMC Neurosci. 13 (Suppl. 1), P149. https://doi.org/10.1186/1471-2202-13-S1-P149.

Lloyd, C.M., Lawson, J.R., Hunter, P.J., Nielsen, P.F., 2008. The CellML model repository. Bioinformatics 24 (18), 2122–2123. https://doi.org/10.1093/bioinformatics/btn390.

Manceau, L., Martre, P., 2018. SiriusQuality-BioMa-phenology-component. https://doi.org/10.5281/ZENODO.2478791.

Marshall-Colon, A., Long, S.P., Allen, D.K., Allen, G., Beard, D.A., Benes, B., Von Caemmerer, S., Christensen, A.J., Cox, D.J., Hart, J.C., Hirst, P.M., Kannan, K., Katz, D.S., Lynch, J.P., Millar, A.J., Panneerselvam, B., Price, N.D., Prusinkiewicz, P., Raila, D., Zhu, X.G., 2017. Crops in silico: generating virtual crops using an integrative and multi-scale modeling platform. Front. Plant Sci. 8, 1–7. https://doi.org/10.3389/fpls.2017.00786.

Martre, P., Jamieson, P.D., Semenov, M.A., Zyskowski, R.F., Porter, J.R., Triboi, E., 2006. Modelling protein content and composition in relation to crop nitrogen dynamics for wheat. Eur. J. Agron. 25 (2), 138–154. https://doi.org/10.1016/j.eja.2006.04.007.

Midingoyi, C.A., Pradal, C., Athanasiadis, I.N., Donatelli, M., Enders, A., Fumagalli, D., Garcia, F., Holzworth, D.P., Hoogenboom, G., Porter, C., Raynal, H., Thorburn, P., Martre, P., 2020. Reuse of process-based models: automatic transformation into many programming languages and simulation platforms. In: Silico Plants. https://doi.org/10.1093/insilicoplants/diaa007.

Muetzelfeldt, R., Massheder, J., 2003. The Simile visual modelling environment. Eur. J. Agron. 18 (3–4), 345–358. https://doi.org/10.1016/S1161-0301(02)00112-0.

Muller, B., Martre, P., 2019. Plant and crop simulation models: powerful tools to link physiology, genetics, and phenomics. J. Exp. Bot. 70 (9), 2339–2344. https://doi.org/10.1093/jxb/erz175.

Neveu, P., Tireau, A., Hilgert, N., Nègre, V., Mineau-Cesari, J., Brichet, N., Chapuis, R., Sanchez, I., Pommier, C., Charnomordic, B., Tardieu, F., Cabrera-Bosquet, L., 2018. Dealing with multi-source and multi-scale information in plant phenomics: the ontology-driven Phenotyping Hybrid Information System. New Phytol. 221 (1), 588–601. https://doi.org/10.1111/nph.15385.

Pradal, C., Dufour-Kowalski, S., Boudon, F., Fournier, C., Godin, C., 2008. OpenAlea: a visual programming and component-based software platform for plant modelling. Funct. Plant Biol. 35 (10), 751–760. https://doi.org/10.1071/FP08084.

Pradal, C., Fournier, C., Valduriez, P., Cohen-Boulakia, S., 2015. OpenAlea: scientific workflows combining data analysis and simulation. In: *Proceedings of the 27th International Conference on Scientific and Statistical Database Management* (SSDBM' 15). Association for Computing Machinery, New York, USA, pp. 1–6. https://doi.org/10.1145/2791347.2791365.

Pradal, C., Varoquaux, G., Langtangen, H.P., 2013. Publishing scientific software matters. J. Comput. Sci. 4 (5), 311–312. https://doi.org/10.1016/j.jocs.2013.08.001.

Refsgaard, J.C., van der Sluijs, J.P., Højberg, A.L., Vanrolleghem, P.A., 2007. Uncertainty in the environmental modelling process - a framework and guidance. Environ. Model. Software 22 (11), 1543–1556. https://doi.org/10.1016/j.envsoft.2007.02.004.

Richmond, B.M., 1985. STELLA: Software for Bringing System Dynamics to the Other 98%. Proceedings of the 1985 International System Dynamics Conference, pp. 706–718.

Rizzoli, A.E., Donatelli, M., Athanasiadis, I.N., Villa, F., Huber, D., 2008. Semantic links in integrated modelling frameworks. Math. Comput. Simulat. 78 (2–3), 412–423. https://doi.org/10.1016/j.matcom.2008.01.017.

Rosenzweig, C., Jones, J.W., Hatfield, J.L., Ruane, A.C., Boote, K.J., Thorburn, P., Antle, J.M., Nelson, G.C., Porter, C., Janssen, S., Asseng, S., Basso, B., Ewert, F., Wallach, D., Baigorria, G., Winter, J.M., 2013. The agricultural model intercomparison and improvement project (AgMIP): protocols and pilot studies. Agric. For. Meteorol. 170, 166–182. https://doi.org/10.1016/j.agrformet.2012.09.011.

Roy, A., 2017. Cookiecutter: better project templates — cookiecutter 1.7.2 documentation. https://cookiecutter.readthedocs.io/en/1.7.2/.

Villa, F., Donatelli, M., Rizzoli, A.E., Krause, P., Kralisch, S., Van Evert, F.K., 2006. Declarative Modelling for Architecture Independence and Data/model Integration: A Case Study. *Voinov, A., Jakeman, A.J., Rizzoli, A.E. (Eds). Proceedings of the IEMSs Third Biennial Meeting:" Summit on Environmental Modelling and Software"*. International Environmental Modelling and Software Society, Burlington, USA, pp. 1–6. http://www.iemss.org/iemss2006/sessions/all.html.

Walker, W.E., Harremoës, P., Rotmans, J., van der Sluijs, J.P., van Asselt, M.B.A., Janssen, P.P.M., Krayer von Kraus, M.P., 2003. Defining uncertainty: a conceptual basis for uncertainty management in model-based decision-support. Integrated Assess. 4, 5–17. https://doi.org/10.1076/iaij.4.1.5.16466.

Wang, E., Brown, H.E., Rebetzke, G.J., Zhao, Z., Zheng, B., Chapman, S.C., 2019. Improving process-based crop models to better capture genotype×environment×management interactions. J. Exp. Bot. 70 (9), 2389–2401. https://doi.org/10.1093/jxb/erz092.

Wang, E., Martre, P., Zhao, Z., Ewert, F., Maiorano, A., Rötter, R.P., Kimball, B.A., Ottman, M.J., Wall, G.W., White, J.W., Reynolds, M.P., Alderman, P.D., Aggarwal, P. K., Anothai, J., Basso, B., Biernath, C., Cammarano, D., Challinor, A.J., De Sanctis, G., Asseng, S., 2017. The uncertainty of crop yield projections is reduced by improved temperature response functions. Nat. Plants 3 (July). https://doi.org/10.1038/nplants.2017.102.

White, J.W., Hunt, L.A., Boote, K.J., Jones, J.W., Koo, J., Kim, S., Porter, C.H., Wilkens, P.W., Hoogenboom, G., 2013. Integrated description of agricultural field experiments and production: the ICASA Version 2.0 data standards. Comput. Electron. Agric. 96, 1–12. https://doi.org/10.1016/j.compag.2013.04.003.

Wyatt, D.L., 1990. A framework for reusability using graph-based models. 1990 Winter Simulat. Conf. Proc. 472–476. https://doi.org/10.1109/WSC.1990.129562.