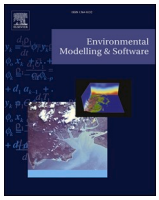
Environmental Modelling and Software 168 (2023) 105790

Contents lists available at ScienceDirect

Environmental Modelling and Software

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



Crop modeling frameworks interoperability through bidirectional source code transformation

Cyrille Ahmed Midingoyi [a,1], Christophe Pradal [b,c,**], Andreas Enders [d], Davide Fumagalli [e], Patrice Lecharpentier [f], Hélène Raynal [g], Marcello Donatelli [h], Davide Fanchini [h], Ioannis N. Athanasiadis [i], Cheryl Porter [j], Gerrit Hoogenboom [j,k], F.A.A. Oliveira [j], Dean Holzworth [l], Pierre Martre [a,*]

[a] LEPSE, Univ Montpellier, INRAE, Institut Agro Montpellier, Montpellier, France
[b] CIRAD, UMR AGAP Institut, 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] AgroClim, INRAE, Avignon, France
[g] AGIR, INRAE, Université de Toulouse, Castanet-Tolosan, France
[h] Centro Ricerca Ingegneria e Trasformazioni Agro-alimentari, CREA, Milano, Italy
[i] Wageningen University, Wageningen, the Netherlands
[j] Department of Agricultural & Biological Engineering, University of Florida, Gainesville, USA
[k] Global Food Systems Institute, University of Florida, Gainesville, USA
[l] CSIRO Agriculture and Food, Toowoomba, Australia

ARTICLE INFO

Handling editor: Daniel P Ames

Keywords:
Crop model
Crop2ML
Model exchange and reuse
One-to-many transformation

ABSTRACT

Recently, we proposed Crop2ML, an open-source modeling framework for exchanging and reusing crop model components between modeling platforms. Here, we present an approach based on reverse engineering to automatically extract and transform meta-information and algorithms of existing crop biophysical models into a platform-independent model component. A search algorithm using Crop2ML concepts, and a many-to-one transformation system were used for producing high-level models. The system consists of parsing the code-base of model components written in different languages using the ANother Tool for Language Recognition (ANTLR) parser generator and processing the generated syntax trees to produce various model implementations. The system was evaluated for three crop model components provided by the BioMA, SIMPLACE, and DSSAT platforms. We demonstrated the extensibility of our approach with the STICS, OpenAlea, and *SiriusQuality* modeling platforms. CyMLTx is a significant contribution towards the interoperability of crop modeling platforms and the reuse of model components beyond programming languages.

1. Software availability

Software name: PyCrop2ML
Contact: Dr. Pierre Martre (pierre.martre@inrae.fr) or Dr. Christophe Pradal (christophe.pradal@cirad.fr)
Year first available: 2022
Hardware required: IBM compatible PC or Apple;
Operation System required: Windows 10 or above, Mac OS X or above, or Linux
Program languages: CyML, Python

* Corresponding author.
** Corresponding author. CIRAD, UMR AGAP Institut, Montpellier, France.
E-mail addresses: cyrille_ahmed.midingoyi@cirad.fr (C.A. Midingoyi), christophe.pradal@cirad.fr (C. Pradal), aenders@uni-bonn.de (A. Enders), davide.fumagalli@ext.ec.europa.eu (D. Fumagalli), patrice.lecharpentier@inrae.fr (P. Lecharpentier), helene.raynal@inrae.fr (H. Raynal), marcello.donatelli@crea.gov.it (M. Donatelli), davide.fanchini@crea.gov.it (D. Fanchini), ioannis.athanasiadis@wur.nl (I.N. Athanasiadis), cporter@ufl.edu (C. Porter), gerrit@ufl.edu (G. Hoogenboom), fabioaug.antunes@ufl.edu (F.A.A. Oliveira), dean.holzworth@csiro.au (D. Holzworth), pierre.martre@inrae.fr (P. Martre).
[1] Current address: AIDA, Univ Montpellier, CIRAD, Montpellier, France.

https://doi.org/10.1016/j.envsoft.2023.105790
Received 7 July 2023; Accepted 1 August 2023
Available online 16 August 2023

Availability:

2. Introduction

Crop models provide a scientific understanding of biophysical processes involved in crop growth and development. They have been increasingly developed and continuously expanded to meet a wide range of needs and applications (Jones et al., 2017). Crop growth models are commonly decomposed into software components implementing the biophysical and ecophysiological functions (e.g., phenology, morphogenesis, resource acquisition, pests, and disease impact) that occur within the plant-soil-atmosphere interactions. Most crop model development teams have adopted crop modeling and simulation platforms to avoid building components from scratch and to rely on good practices in software engineering. These platforms are aimed at a specific community of cropping and farming systems modelers (Argent et al., 2006). However, there is an increasing need for the exchange and reuse of model components from different communities to simulate and model new assumptions and processes in cropping systems (Holzworth et al., 2014). However, the difference between modeling platforms is a technical barrier for reusing a crop model component in other platforms (Muller and Martre, 2019). In addition, building models from legacy model components written in different programming languages and provided by different crop modeling platforms remains challenging (Midingoyi et al., 2020).

Different modeling communities and networks adopt and support different standards to enable the reuse and the interoperability of multilingual models. For instance, in the plant science community, the Crops in Silico (Cis) community (Marshall-Colon et al., 2017) provides a multilanguage and integrated modeling framework through interfaces in the supported languages (Lang, 2019). In the environmental modeling community, web-service approaches are also used to couple components (Gao et al., 2019). Others use service-oriented wrapper systems for adapting components with their software design constraints (Hutton et al., 2020; Jiang et al., 2017; Peckham et al., 2013).

Other communities, such as the system biology community, have developed domain specific languages (DSLs) as standard languages for model representation and exchange (Cuellar et al., 2003; Gleeson et al., 2010; Hucka et al., 2003). Likewise, in the crop modeling community there is an increasing need for a seamless model component exchange and reuse mechanism (Athanasiadis et al., 2011; Holzworth et al., 2014; Martre et al., 2018). Recently, the Agricultural Modeling Exchange Initiative (AMEI) proposed Crop2ML, an open-source modeling framework for exchanging and reusing crop model components between modeling platforms (Midingoyi et al., 2021). Crop2ML is shared and can be retrieved on an open-source format through an accessible online model repository (http://crop2ml.org). Crop2ML follows the Minimum Information Required in the Annotation of Models (MIRIAM (Le Novère et al., 2005) and the Open Modeling Foundation (OMF,Barton et al., 2022a,b) sets of guidelines that define how a model should be encoded.

The Crop2ML framework provides a unified description of model components specification at a high-level of abstraction based on shared concepts, lifting constraints of modeling platforms, and a minimal domain language, called CyML, for the description of associated algorithms (Midingoyi et al., 2020). CyML is a subset of the Cython language representing the common language constructs used for crop model implementation in crop modeling platforms. A transformation system (CyMLT) has been implemented to transform Crop2ML models into model components in different languages and is targeted either at a specific platform, or to components with no dependency to a specific platform. However, existing components should first be re-written in Crop2ML before they can be automatically transformed using CyMLT. This rewriting process can be cumbersome, error-prone, and may require significant efforts in the case of large model components. Therefore, the goal of this project was to extend the Crop2ML framework and the CyMLT transformation system by providing an automatic system (CyMLTx) which generates automatically Crop2ML model components from existing platform-specific crop model components written in different languages.

Crop modeling platforms offer different capabilities for implementing and specifying model components based on the patterns they share with the model developers who use them. These two aspects of a component (implementation and specification) are useful to address transformation into another platform (Holzworth et al., 2010). The description and accessibility of component meta-information (variables and parameters description, domain of validity, measurement units, …) depend on the language capabilities, as well as the modeling platform specifications, which use either the procedural or the object-oriented programming paradigms. The former is historically well suited to represent biophysical processes in simple functions. For instance, in the Fortran language, used in DSSAT (Hoogenboom et al., 2019) and STICS (Brisson et al., 2009), the meta-information for model components is provided as comments in the code, with no specific format and conventions that model developers must implement. On the other hand, platforms that use the object-oriented paradigm benefit from the encapsulation property which allow separation of specification from implementation. Components are implemented with standard class methods that could facilitate code understanding and the discovery of meta-information.

It is a great challenge to automatically generate platform-independent model specifications. A reverse engineering approach has been used to extract information from model documentation and source code. For instance, some crop modeling platforms (e.g., BioMA, SIMPLACE) facilitate the discovery of types and properties via reflection operating at runtime (Villa et al., 2006). Most of the extraction methods allowing to generate model specifications are based on concepts derived from a specific domain knowledge (Gyori et al., 2017; Nigam et al., 2015). Bidirectional source-to-source transformation systems between multiple languages require using subset languages and to convert them to the same intermediate representation (Plaisted, 2013). Inspired by these projects, we hypothesize that the CyML language, which represents the intersection of several high-level language constructs, can be leveraged to enable bidirectional transformation between several programming languages. CyMLTx combines extraction methods and source-to-source transformation that lower barriers in both platforms and languages barriers. This new system complements CyMLT by adding transformation from multi-platforms and languages to Crop2ML language transformation. It thus leads to platform interoperability using Crop2ML as a bridge. Consequently, CyMLTx and the Crop2ML framework, can be used to automatically import and export model components from several crop modeling platforms. CyMLTx was designed to transform model components developed in various platforms into high-level abstractions, thus promoting model reuse, augmentation, and collaboration between different modeler groups. It allows abstracting a component into a comprehensible representation through which any modeler is driven to work on without knowing platform specificities.

In this paper, we first present the approach and the implementation of CyMLTx. Then, we demonstrate the interoperability between three different crop modeling platforms (BioMA, (Donatelli and Rizzoli, 2008); SIMPLACE, (Enders et al., 2010, 2023), and DSSAT (Hoogenboom et al., 2019),) with the use of CyMLTx on three components: an energy balance model component provided by the BioMA platform and two soil temperature model components provided by SIMPLACE and DSSAT. Finally, we illustrate the extensibility of CyMLTx to other crop modeling platforms with two other crop modeling platforms (STICS, (Brisson et al., 2009); and *SiriusQuality* (Martre et al., 2006), and the plant modeling platform OpenAlea (Pradal et al., 2008, 2015). Finally, we discuss our results and present some perspectives.

3. Methods

3.1. CyMLTx transformation workflow

CyMLTx extends the capability of CyMLT. It takes the source code of a model component (M1) from a modeling platform as input and transform it into a Crop2ML model component (M2) at the same level of granularity. M2 is an abstract model that can either be a Crop2ML ModelUnit with a fine granularity or a Crop2ML ModelComposite represented as a graph of ModelUnits connected by their inputs and outputs to manage model complexity (Midingoyi et al., 2021). Each ModelUnit consists of a model specification and an associated model algorithm. Model specifications contain formal descriptions of the model, its inputs and outputs, a link to an initialization function of its state variables, and a set of parameters and unit tests. Model algorithms, auxiliary functions, and state variable initialization functions are expressed in the CyML language (Midingoyi et al., 2020). Hence, the CyMLTx approach is designed to capture these concepts from the M1 source code and to generate the corresponding Crop2ML model M2.

The main processes of the CyMLTx transformation workflow are (1) the extraction of meta-information from the source code of M1 to build the specifications of M2; (2) the pre-processing of the Abstract Syntax Tree of M1 and the generation of the Abstract Semantic Graph of the different parts of M1 that represent the algorithms, initialization, and auxiliary functions; (3) and the transformations of these parts into the CyML language and their merging with model specification to generate M2 (Fig. 1).

The workflow starts with parsing of the source code of M1 that consists of finding the syntactic structure of the codebase to generate the parse trees called Concrete Syntax Trees (CST). The parse tree is then transformed into an Abstract Syntax Tree (AST) as an annotated graph to improve the average access time of information. It is a refinement of the parse tree, with some non-terminals, keywords, and punctuations removed while maintaining the meaning of the program. A data structure model is designed for each language to represent AST in such a way that the AST of any source provided by different platforms using the same language will be represented by the same model. Comments are also extracted and are added directly to the correct nodes in the AST. Meta-information is extracted from the code to recover the values of the concepts of Crop2ML model specifications related both to the description of the component (authors, names, version, etc.) and to variables and parameters information (names, description, types, units, …). These values can be retrieved either from text pattern matching applied on the code comments or directly from the AST analysis (step 1).

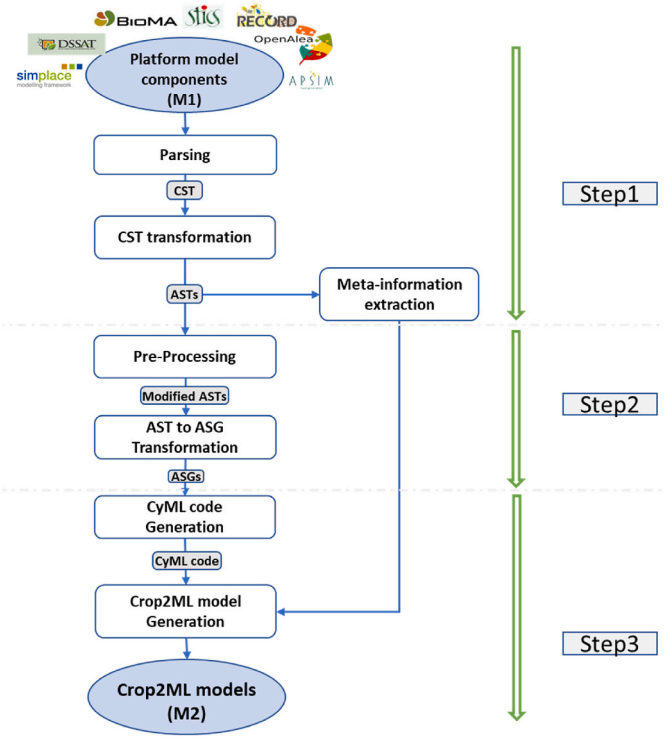


Fig. 1. Schema of the main steps of the CyMLTx transformation workflow for Crop2ML model generation from platform model components. Green arrows show the three main steps of the transformation process: (1) generation of the Concrete Syntax tree (CST) and extraction of meta-information from the Abstract Syntax Trees (ASTs) of the source code; (2) pre-processing of the ASTs and generation of the Abstract Semantic Graphs (ASGs) of the source of the algorithms, initialization and auxiliary functions; and (3) transformation of the ASGs into the CyML language and merging with model specifications to generate a Crop2ML model component. Intermediate results are highlighted in grey.

Step 2 is to first extract the model algorithms and the initialization of the state variables based on the code annotations or design provided by the modeling platform. If some algorithms depend on auxiliary functions, their ASTs are extracted. Then, the ASTs of the algorithms, and initialization and auxiliary functions are modified to remove platform-specific features. Finally, each modified AST is transformed to an Abstract Semantic Graph (ASG; Midingoyi et al., 2020), which is a self-contained representation of the source code. The ASGs are independent of the source language and their nodes are only based on CyML constructs.

In Step 3, the CyML code is produced based on ASGs traversal and predefined rules mapping each node type to a corresponding code. Since the ASG design is independent of any language and platform, this step is unique whatever the language and the platform. Then, the model specifications are inferred to generate the Crop2ML model component and reference the auxiliary and initialization functions, if they exist.

3.2. Requirements for a crop model component

In our context of model reuse and exchange, the code of the source model should meet some criteria:

- A model component should be self-contained, adaptable, and reusable for different purposes by third parties. All functions level dependencies should be explicit, i.e., the model component codebase should have a direct access to native code of all dependencies written in the language of the platform.
- Components should explicitly define their inputs and outputs and without side effects. They should be decoupled from the data storage. The main challenge for the compositional approach is that components must be divided into independent fine-grained models that can be coupled through their inputs-outputs and run sequentially. Thus, model components should provide a well-defined interface containing all required information to ensure their extraction. Similar to Javadoc (Kramer, 1999) and Doxygen (Laramee, 2011), some annotations relating to Crop2ML concepts are provided for model description in the case where the modeling platform does not provide an interface for the model description such as @author, @timestep, @shortDescription, etc. Besides, this meta-information can be provided as code comments or docstring that allow extraction of the required information using a search algorithm. This minimizes the need to embed markup and maintains the overall readability of the comments in the code. Most object-oriented platforms provide a design of components that make it possible to know exactly where algorithms and initialization methods are implemented, and to infer them. However, for non-object-oriented platforms that do not provide a fixed design, components should include additional information based on specific annotations within comments to indicate algorithms and state initialization parts.

- The current version of CyML does not support exception handling, events, generators, and unconditional branching; therefore, the code of the source model should be modified to remove these constructs.

3.3. Design and architecture of CyMLTx

The design and some implementation details of the core of CyMLTx are presented below. The transformation rules that have been defined in CyMLT associate a CyML construct to one construct of each target language. We place the restriction on this forward transformation by limiting the CyML constructs with a set of constructs shared by the target languages, which consequently limits the target language constructs. The backward transformation rules can be either generated from the forward transformation or explicitly defined from a new set of language constructs that have their equivalence in CyML.

3.3.1. Code transformation principles

Let us consider CyMLT as the forward transformation $\varphi : C \rightarrow P_i$, where C and P_i are the set of constructs of the CyML language and a language used by a platform, respectively. C is the common set of constructs in the implementation of components in crop modeling platforms. In this context, a total mapping means that the function is defined for all possible constructs in the CyML language, so whatever the construct inputted into the function, a corresponding construct in the target platform's language will always be produced. Therefore, the function φ is considered a total mapping because it maps every possible input to a corresponding output. Furthermore, P_i is a restriction of the corresponding language so that the sets C and P_i are semantically equivalent. Thus, CyMLT provides a base of constructs for each restricted platform language, and for each p in P_i, there is one c in C such that $\varphi(c) = p$ and φ is a bijection mapping between C and P_i. It is also possible to find the inverse transformation φ^{-1} so that $\varphi o \varphi^{-1} = Identity$. However, this consideration using the same restricted set P_i for the inverse transformation makes the mapping system too restrictive. There are other constructs in the platform not included in P_i that can have an equivalence in C. These constructs were not included in P_i because φ is an application since in CyMLT there is only one construct in P_i associated to a construct in C. To consider these constructs in the inverse transformation, we need to define a backward transformation $\mu : Q_i \rightarrow C$ distinct from the forward transformation so that Q_i is an extension of P_i. The set of constructs of CyML does not change. The main new constructs included in Q_i are composite variables and other constructs equivalent to conditional branching statements, such as switch, select case, etc. Depending on the language of the platform.

These new constructs need to be integrated in P_i in order to increase the capacity of the transformation system so that it is able to transform many components. Indeed, the use of composite variables is a common practice in crop model development. For example, in the DSSAT platform, a single variable related to weather can include different types of information such as day length, precipitation, maximum and minimum air temperature, and wind speed. A composite data structure is used to instantiate composite variables. Composite variables are automatically decomposed into several individual variables according to the CyML data structures. Thereby, for q *in* $Q, \varphi(\mu(q)) = q^{'} q^{''} \in P_i \subset Q_i$. $q^{'} \neq q$ but $q^{'}$ and $q^{''}$ are semantically equivalent. Consequently, the implementation of CyMLTx provides a set of transformation definition rules that are distinct from those of CyMLT to take into account constructs provided by the modeling platforms that are defined in CyML.

3.3.2. CyMLTx implementation

CyMLTx has a modular architecture based on the technical design that underly the different steps that are defined in the transformation workflow (Fig. 1). Similar to CyMLT, we implemented it in the Python language to provide an interactive mode in which users have access to the intermediates results of the workflow.

It is cumbersome to implement a parser for each language for a scalable system in the context of multiple languages. To reduce implementation efforts, particularly the programming time for the implementation of parsers and considering the extensibility and efficiency requirements, we used the ANTLR (ANother Tool for Language Recognition) parser generator (Parr, 2013) that produces lexical and syntactic parsers for different grammars of programming languages. The ANTLR provides a collection of grammars for many popular programming languages, including C, C++, Java, C#, Fortran 77, and Python. It augments the grammar with tree operators and rewrites rules and actions. In the case where the grammar database does not contain an input language, its grammar can be expressed based on the ANTLR syntax, as we did in this work for Fortan 90 based on Fortran 77. This new grammar has been reviewed and approved by the ANTLR community and is available at https://github.com/antlr/grammars-v4/tree/master/fortran/fortran90.

The relationships of the main classes of CyMLTx are illustrated in Fig. 2. The generated parsers *(LanguageParser)* were used to generate CST from the model components. A set of classes (*LanguageTransformer*) whose names are suffixed by "Transformer" were implemented for the transformation of the CST to the AST for each language of the supported platforms, for instance *CsharpTransformer* class for the C# language that is supported by BioMA. Each class implements visitor methods based on the grammar constructs. Each visitor method name is composed of "visit_" followed by the type of the constructs and emits a new node.

A *MetaExtraction* class is an abstract class that implements methods that allow for extracting information from the AST or the code comments. It provides four main methods:

- *Get Node from its Type (getTypeNode)* that takes as input a tree and a type of node T and returns a subtree or list of subtrees of Nodes of type T.
- *Get Node from its Attributes (getAttNode)* that takes as input a tree and the values of node attributes and returns the nodes that have these attribute values.
- *Get Method (getMethod)* that takes as input a tree and a function or method name and returns a list of subtrees of the function or method that has this name. It is a particular method of getAttNode.
- *Get From Comments (getFromComments)* that allows extracting meta-information from code comments.

An *Extraction* class embedding the specificities of the modeling platforms (such as model design) is implemented for each platform (*PlatformExtraction*). It specializes the MetaExtraction class and is used to capture some required information, in particular model meta-information, by generating a Model (ModelUnit or ModelComposition) object. The Model object is then translated into Model Specification in an XML format validated with the Crop2ML Document Type Definition (DTD). The methods of the "Meta*Extraction*" class are also used to analyze the AST and to extract the model dynamics, initialization functions, and external functions as subtrees. These subtrees are then procedurally manipulated with incremental and iterative processes according to the types of the nodes (Preprocessing). No formal grammar is used to describe the part of the platform specification on the language. Thereby, the regular constructs (patterns) of platforms are embedded in the implementation process. The number of processes depends on the invasiveness of the modeling platform.

A class corresponding to a node type is implemented and inherits the *Transformer* abstract class that implements a method to capture all the nodes of the desired type on which the processes are applied. A set of classes named *Language_CyML* (for example *java_cyml*) are used to operate the transformation of the processed AST to the ASG whose nodes are only based on CyML constructs that are provided from the intersection of the framework languages.

Lastly, the *CyMLGenerator* class implements a method (*write()*) to generate CyML code from the ASG. It is based on the Visitor design

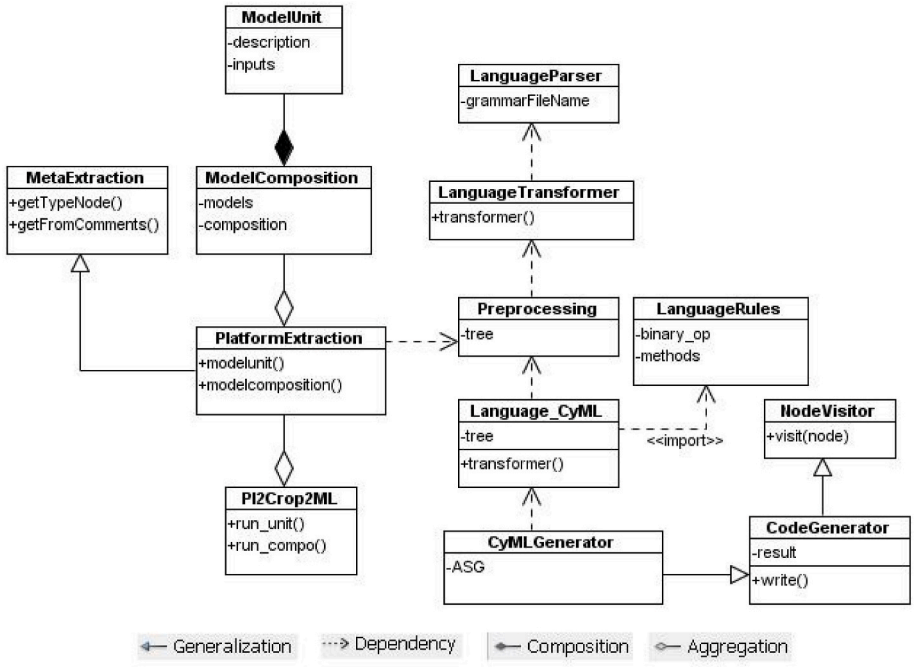


Fig. 2. Class diagram illustrating the implementation of the platform to Crop2ML transformation system (CyMLTx). The classes contain attributes and methods.

pattern (Gamma et al., 1995) to avoid a procedural implementation approach. *NodeVisitor* contains a dispatch method that enables recursive tree traversal through the nodes of the ASG. The appropriate visitor method corresponding to the type of the current node is called and the associated code fragment is emitted. Since all the required built-in methods provided from the languages and their mapping with CyML methods are listed in the system database, any other methods called in the algorithm part are considered external or auxiliary functions. The names of these functions help to extract their AST and to perform all the above transformation processes to generate their ASG and then the corresponding CyML code.

CyMLTx generates a Crop2ML ModelComposite as a graph of components from a composite expressed in a declarative form or using a procedural approach. In the latter case, the composite must be implemented as sequential calls of the unit components. Although an algorithm can be expressed in the Crop2ML ModelComposite, the current version of CyMLTx does not support an automatic transformation of a composite expressed with control structures.

To ensure modularity and extensibility, the generation of Crop2ML ModelComposite consists of two stages: the processing of the platform component to generate an instance of the *ModelComposition* class based on the concepts of Crop2ML ModelComposite, and the transformation of this object in XML format that can be visualized as a graph of unit components. The last process is implemented through the Pl2Crop2ML (Platform to Crop2ML) class that contains two methods *run_unit() and run_compo()* methods for ModelUnit specification and ModelComposite specification file generation.

3.4. Use cases

We demonstrate our approach with three widely used crop modeling platforms: BioMA (Donatelli and Rizzoli, 2008), SIMPLACE (Enders et al., 2010), and DSSAT (Hoogenboom et al., 2019a,b; Jones et al., 2003). These platforms raise different challenges: i) they use different programming languages (DSSAT uses Fortran 90, BioMA uses C#, and SIMPLACE uses Java); ii) they use object-oriented (SIMPLACE and BioMA) or procedural (DSSAT) programming paradigms; and iii) their components are specified through explicit interfaces (BioMA and SimPlace) or code comments (DSSAT).

We provide one use case model component for each of the three platforms to highlight the challenges of the automatic transformation from programming language and modeling platform to Crop2ML. Specificities of each transformation are summarized in Table 1.

Use case 1: The first use case is the energy balance model component of the Sirius wheat model (Jamieson et al., 1995) implemented in the BioMA framework (Manceau et al., 2023). It includes eleven simple strategies (ModelUnits) executed sequentially to estimate canopy temperature, soil evaporation, and crop transpiration. BioMA adopts the strategy design pattern (Gamma et al., 1995) to make available a set of models that represent biophysical processes in a component through the same interface (IStrategy). Such models are called “simple strategies”. They encapsulate model specifications (inputs, outputs variables, algorithms), parameters, and pre- and post-conditions tests, and are translated to Crop2ML *ModelUnits* (Midingoyi et al., 2021). Simple strategies are embedded in a composite strategy that is mapped to Crop2ML *ModelComposite*.

Use case 2: The second use case is the soil temperature component implemented in SIMPLACE. This component simulates the daily average soil temperature at the center of each soil layer. Soil temperature fluctuations in each soil layer depend on the soil surface temperature (including the effect of snow cover or crop cover), the distance of the layer from the soil surface, and the damping depth where the soil temperature is equal to the annual average air temperature (Williams and Izaurralde, 2005). SIMPLACE is designed to encapsulate the solution of a modeling problem through discrete, replaceable, and interchangeable software units called SimComponents (Enders et al., 2010). Different approaches for the simulation of biophysical processes can be used via alternate mathematical formulations. SimComponents are the smallest building blocks.

The specification of SimComponents has been mapped to Crop2ML *ModelUnit* specification (Midingoyi et al., 2021). Group Components combine SimComponents into logical structures of components that belong together and map to Crop2ML *ModelComposite*.

Use case 3: The third use case is the DSSAT soil temperature component, originally based on the EPIC soil temperature model (Williams et al., 1989) and further improved by the DSSAT community. DSSAT has a modular structure in which the different components are structured to allow easy replacement or addition of new modules. The

Table 1
Transformation specificities of the crop modeling platforms that have been analyzed in this project.

	BioMA	SIMPLACE	DSSAT	STICS	OpenAlea
Metadata description	BioMA provides a user interface (DCC, Domain Class Coder) to generate model variable information in XML file and in C# code with BioMA classes	A SimComponent class implements a method "createVariables" that specifies variable information as arguments. Other information is provided through annotation such as Authors' names and reference	Model metadata is described as code comments		Metadata is associated explicitly to each package and component (e.g. authors, description, version).It can also be provided inside the node function as code comments
Element used to create Crop2ML ModelUnit specifications	Strategy and Variable information classes (*VarInfo* class)	Arguments order of the method *createVariables()*	Regular expression in code comments that matches specified patterns		Regular expressions and code analysis
Missing Crop2ML ModelUnit specifications	Reference, strategy version, extended description, time step, parameter category	Extended description, author institution, version, timestep, parameter category			Units, Categories, if not provided in code comments
Model composition	Procedural approach using a Composite strategy class I that implements a sequential calls of unit components and can contain control structure	Declarative approach using XML format (GroupComponent DTD)	Procedural approach using a subroutine that sequentially calls other subroutines considered as a unit component		A composite node encapsulates others nodes defining a hierarchy of components
Transformation of a model composite	From code to components graph (control structures are not handled)	Graphs transformation (SimComponent [inputs and outputs] in the concepts of the GroupComponent DTD are not considered)	From code to component graph: the composite implements a sequential calls of unit components		Graph transformation
Test-driven development	Pre and post condition tests. Unit tests defined at composite level	Unit test defined inside the code in declarative format	No unit test		
Transformation of unit-test	Not handled (requires completing the generated Crop2ML specifications)				
Language in which the component behavior is implemented in the platform	C#	Java	Fortran		Python
Transformation of the model algorithm	Yes, but language constructs are limited to the (extensible) CyML grammar Utility functions are not handled				
Transformation of the platform language into CyML	In the limit of the platform requirements	Annotation could be added to remove some parts of code	Platform specificities (e.g. control variable) are handled through annotations	Pure Fortran code without platform specificities	Annotations used to identify the initialization function and other parts of code

core of the DSSAT platform is the Cropping System Model (CSM) designed with a modular architecture in which components are separated along scientific domains (crop, soil, soil and land management, weather) and use interfaces to replace or add modules (Jones et al., 2003). The CSM is divided into five primary modules including weather, management, soil, plant, and energy balance. Each primary module is further subdivided into secondary modules and finer delineations. Each module is called to perform initialization of state variables at the beginning of a simulation, and at each iteration to calculate daily rates and perform daily integration of state variables, while output is generated at different times during the simulation.

3.5. Extension of CyMLTx to new platforms

The modular architecture of CyMLTx has been designed to allow its extensibility to a platform that is currently not supported. It presents three levels of processes: the first relates to the language that produces the AST, the second relates to the platform component design allowing the ASG generation and meta-information extraction, and the third generates the Crop2ML component. The last step of ASG transformations into the CyML language can be reuse in any new platform since, as stated previously, is implemented regardless of the platform and language specificities that transform the ASG and Model object to the Crop2ML model.

Therefore, the addition of a new platform involves:

- Providing the parser of the programming language of the platform. This step could be skipped if it has already been included in the CyMLTx language parsers repository, which is currently the case for C++, C#, Java, Fortran, Python, and R. Many platforms can share the same programming language, and in this case the same language parser will be used. Language grammars could also be provided from the ANTLR repository (https://github.com/antlr/grammars-v4). Otherwise, a new language parser in Python should be integrated in the CyMLTx parser repository.
- Defining the transformation rules between the new language and CyML. These mapping rules are only be based on the language constructs used to express the compute function of components outputs, initialization, and auxiliary functions. At this level, it requires considering the shared constructs identified to express model algorithms. The set of model components or platform that can be translatable in Crop2ML is limited by the constructs defined in CyML but it can be extended.
- Describing the processes of meta-information extraction and platform specificities handling. It requires a series of defined actions that extract, transform, and remove specificities of the new platform to generate intermediate tree or object with no platform dependency. The complexity of this step depends on the distance between the platform specificities and Crop2ML/CyML concepts.

To evaluate the extensibility of the system, OpenAlea (Pradal et al., 2015), STICS (Brisson et al., 2009), and SiriusQuality (Martre et al., 2006) were added to the system. OpenAlea is an open-source modular and component-based framework that addresses the modeling needs of the plant biology research community. OpenAlea model components are commonly written in the Python language and are represented as Nodes

that expose their inputs and outputs ports. The composition of a model component is represented as a graph of Nodes that allows for defining the hierarchy of components. In STICS, a model is organized into modules commonly implemented in the Fortran language. Each module can be subdivided into sub-modules that can be called sequentially to define a composition. In SiriusQuality, most of the processes were implemented as BioMA Components. The main principles of the extension of CyMLTx with each platform are provided in the Results section.

4. Results

The evaluation of CyMLTx aims to show that the system can convert model components of diverse crop modeling platforms into Crop2ML models. The generated models can be reused, modified, and composed with other model components by modelers seeking alternative modeling assumptions or formalisms, or models of new processes to tackle real-world system modeling issues. Here, we present some implementation elements for the three crop modeling platforms and the three test cases described above.

4.1. Generation of Crop2ML ModelUnit specifications

One class is implemented for each platform to achieve the meta-information extraction (e.g. biomaExtraction or simplaceExtraction), which can then be used to build the Crop2ML ModelUnit specifications. Each class implements an extraction method based on the pattern provided by each platform for meta-information description.

The *biomaExtraction* class implements a method that generates Crop2ML model specifications from the processing of both the *VarInfo* and *Strategy* classes (Fig. 3). The pattern used to identify the meta-information is defined as follows: (1) the meta-information of the parameters and the name and category (states, rates, …) of the variables are retrieved from the attribute *PropertyName* of one instance of the *PropertyDescription* class defined in the constructor of the *Strategy* class; (2) the retrieved variable names are then used to get their attribute values in the *DescribeVariables()* method of the corresponding *VarInfo* class; and (3) other meta-information such as description (in the *Description* property) and authors' names (in the *SetPublisherData()* method) are retrieved in each strategy class (see Fig. 4).

The *simplaceExtraction* class produces Crop2ML ModelUnit specifications from the SIMPLACE SimComponent (Erreur ! Source du renvoi introuvable.). It extracts the variables and parameters meta-information from the *createVariables()* method implemented by any SimComponent. Using the CyMLTx core method *getAttNode(AST, createVariables()),* the subtree corresponding to the *createVariables.)* method of the component is filtered. Each *Call* node (node of type "Call" corresponding to the call of a method) of the *CreateSimVariable()* method inside this subtree is accessed. So, the *SimplaceExtraction* method consists in retrieving the arguments of each *CreateSimVariable()* call node to infer the Crop2ML model specifications based on the order of the arguments.

The *dssatExtraction* class uses the meta-information of subroutines provided as code comments and code statements (e.g. variable type and datatype) to generate Crop2ML ModelUnit specification (Fig. 5). CyMLTx detects composite variables or parameters with derived types used in the subroutine and retrieves the unit variable or parameter with built-in types required by the component. In the third use case, the Soil Temperature Subroutine contains SOILPROP and WEATHER composite arguments that encapsulate 59 soil properties and 43 weather variables, respectively.

Only six soil properties and two variable parameters are used in this subroutine. They are automatically detected, extracted, and explicitly used to generate the model specification. Other arguments that are specific to the DSSAT execution environment, such as control variables that are not useful to be represented as component inputs in Crop2ML ModelUnit, are automatically removed.

4.2. Automatic transformation of the source code of the platform's algorithm into CyML

CyMLTx generates Crop2ML ModelUnit algorithms from each component (i.e. Simple Strategy, sub-routine, or SimComponent of the BioMA, DSSAT, and SIMPLACE platforms.

In BioMA the algorithm of each Simple Strategy is extracted from the algorithm method defined by the component's *IStrategy* implementation. In this particular case, this method is "*CalculateModel()*" called by an "*Estimate()*" method. The query on the AST of each simple strategy class using the name of the method allows recovering the sub-AST corresponding to the strategy algorithm (computation process), which is transformed in CyML code. The processing of the major specificity of

a)

```
public class EBAuxVarInfo : IVarInfoClass
{
    static VarInfo _minTair = new VarInfo();
    public static VarInfo minTair
    {
        get { return _minTair;}
    }
    static void DescribeVariables()
    {
        _minTair.Name = "minTair";
        _minTair.Description = "min air temperature";
        _minTair.MaxValue = 45;
        _minTair.MinValue = -30;
        _minTair.DefaultValue = 0.7;
        _minTair.Units = "degC";
            //…
    }
}
```

b)

```
public class CanopyTemperature : IStrategyEnergyBalance
{
   public CanopyTemperature()
   {
        //Parameters
        VarInfo v1 = new VarInfo();
        v1.DefaultValue = 2.454;
        v1.Description = "latent heat of vaporization of water";
        v1.Id = 0;
        v1.MaxValue = 10;
        v1.MinValue = 0;
        v1.Name = "lambdaV";
        v1.Size = 1;
        v1.Units = "MJ/kg";

        //Inputs
        PropertyDescription pd1 = new PropertyDescription();
        pd1.PropertyName = "minTair";
        pd1.PropertyVarInfo =(EnergyBalance.DomainClass.EBAuxVarInfo.minTair);

        //Outputs
   }
}
```

c)

```
<Inputs>
    <Input name="minTair" description="min air temperature" inputtype="variable" variablecategory="auxiliary" datatype="DOUBLE"
           min="-30" default="0.7" unit="degC"/>
    <Input name="lambdaV" description="latent heat of vaporization of water" inputtype="parameter" parametercategory="constant"
           datatype="DOUBLE" max="10" min="0" default="2.454" unit="MJ/kg"/>

</Inputs>
```

Fig. 3. Transformation of the BioMA strategy and VarInfo classes to Crop2ML model specifications: variable description in a VarInfo class (a), parameter description and variable names accessible in a simple strategy (b)and part of the generated Crop2ML model specifications (b).

a)

```
public HashMap<String, FWSimVariable<?>> createVariables()
{
    addVariable(FWSimVariable.createSimVariable("iSoilWaterContent", "Content of water in Soil", DATA_TYPE.DOUBLE, CONTENT_TYPE.input,
                                                "mm", 1.5, 20d, 5d, this));
    addVariable(FWSimVariable.createSimVariable("SoilTempArray", "Array of temperature ", DATA_TYPE.DOUBLEARRAY, CONTENT_TYPE.out,
                                                "degC", -20, 40, null, this));
    //...
     return iFieldMap;
 }
```

b)

```
<Inputs>
    <Input name="iSoilWaterContent" description="Content of water in Soil" inputtype="variable" variablecategory="exogenous"
           datatype="DOUBLE" max="20" min="1.5" default="5" unit="mm"/>
    <Input name="iSoilSurfaceTemperature" description="Temperature at soil surface" inputtype="variable" variablecategory="exogenous"
           datatype="DOUBLE" max="20" min="1.5" default="" unit="deg"/>

</Inputs>
<Outputs>
    <Output name="SoilTempArray" description="Array of temperature " datatype="DOUBLEARRAY" variablecategory="auxiliary" len="" max="40"
            min="-20" unit="degC"/>
</Outputs>
```

Fig. 4. From SIMPLACE SimComponent to Crop2ML model specifications: variable and parameter description in the createVariables method of SimComponent class (a) and part of the generated Crop2ML model specifications (b).

a)

```
      SUBROUTINE STEMP_EPIC(CONTROL, ISWITCH,
     &    SOILPROP,CUMDPT, ...)         !InOut

      USE ModuleDefs

      REAL ABD, B, CUMDPT
      REAL, DIMENSION(NL) :: BD, DLAYR, DS

      TYPE (SoilType)    SOILPROP

      DS   = SOILPROP % DS
      !!!!!!!....
      RETURN
      END SUBROUTINE STEMP_EPIC

      !%%CyML Description Begin%%

      ! CUMDPT   Cumulative depth of soil profile (mm) () state variable
      ! DS(NL)    Cumulative depth in soil layer NL (cm) () soil parameter
      ! SOILPROP Composite variable containing soil properties
            !
      ! %%CyML Description End%%
```

b)

```
<Inputs>
   <Input name="DS" description="Cumulative depth in soil layer NL" inputtype="parameter" parametercategory="soil" datatype="DOUBLEARRAY"
          len="NL" max="" min="" default="" unit="cm"/>
   <Input name="CUMDPT" description="Cumulative depth of soil profile" inputtype="variable" variablecategory="state" datatype="DOUBLE"
          max="" min="" default="" unit="mm"/>

</Inputs>
```

Fig. 5. From DSSAT subroutine to Crop2ML ModelUnit specifications:part of Soil Temperature subroutine which uses a derived type (SoilType) to declare a composite variable SOILPROP. DS variable is a member of this variable that can be retrieved by the *dssatExtraction* class through the assignment. The meta-information are enclosed in two tags with a well-defined format (%%CyML Description Begin/End%%) (a) and part of Soil Temperature Crop2ML ModelUnit specification (b). Although the input DS is not defined as the subroutine argument, it is a parameter of the component extracted from the derived type.

BioMA component, i.e., the use of the instances of the domain classes to implement the computation process, made it possible to move from the object-oriented approach to a procedural approach. Fig. 6a shows the computation process of the Priestley-Taylor strategy of the Energy Balance component implemented in BioMA. The access to variables through domain classes instances is removed and only variables and parameters are used to express the generated algorithm in CyML (Fig. 6). If two instances of the state domain class encapsulating state variables manage the current and previous time steps, the generated CyML code has two variables to emulate the two states of the same state variable. Moreover,

a)

```
private void CalculateModel(EBState s, EBState s1, EBRate r, EBAuxiliary a, EBExogenous ex)
{
   r.evapoTransPTaylor  = Math.Max(Alpha * a.hslope * a.netRadEquiEvapo / (a.hslope + psychrometricConst), 0.0d);
}
```

b)

```
evapoTransPTaylor = max(Alpha * hslope * netRadEquiEvap / (hslope + psychrometricConst), 0.0)
```

Fig. 6. Computation of Priestley-Taylor evapotranspiration in the Energy Balance component: BioMA code (a). EBState, EBRate, EBAuxiliary and EBExogenous encapsulate the state, rate, auxiliary and exogenous variables, respectively; Crop2ML ModelUnit algorithm generated by CyMLTx (b).

the *CalculateModel()* method of a simple strategy class may depend on a method whose arguments can be the instance of domain classes and without explicit return values. Thus, CyMLTx implements in the pre-processing module the function that analyzes the calling method to extract the actual inputs and outputs. Inputs are fields of the domain class on which the method actually depends. They are used to calculate other variables and outputs corresponding to the modified variables of domain classes in the body of the called method.

For SIMPLACE, CyMLTx identifies **Erreur ! Source du renvoi introuvable.** the computation and initialization code parts of the SimComponent. Any model component in SIMPLACE implements a *Compute ()* method to express the model algorithms and an *init()* method for the initialization of the **Erreur ! Source du renvoi introuvable.** state variables**Erreur ! Source du renvoi introuvable**. This information makes it possible to extract the sub-ASTs of the model algorithm and initialization part. These subtrees are transformed to generate the model algorithm and initialization function in CyML. One specificity of SIMPLACE is its custom data type *FWSimVariable* that encapsulates variables and parameters and implements the getter and setter methods to access and update their values, respectively (Fig. 7). The accessor method is removed and only the member corresponding to the model variables or parameters are used. The setter method is emulated to an assignment statement (Fig. 7Erreur ! Source du renvoi introuvable.). The Soil Temperature component provides an initialization function that estimates the number of soil layers, the depth of each layer, and initializes the temperature of each layer (Fig. 8a). The member variables (e.g, SoilTempArray) that are defined within the class of the component outside of any method are either inputs or outputs of the SimComponent. Consequently, they are also not declared in the generated CyML code (**Erreur ! Source du renvoi introuvable. b**) since they are defined in the generated model specifications. Only the local variables are declared and the difference of variable scope between the source and target language are handled.

CyMLT annotations were inserted into the DSSAT sub-routines to allow the transformation system to identify the initialization and algorithm parts. The algorithm may be composed of rate calculations and integration processes. Some statements were ignored through annotations, including the input and output operations and the calls of functions that do not impact output computations such as formatting. CyMLTx implements a mechanism to parse imported modules through the USE statement and extract the information required by the computation of the outputs. For example, the *ModuleDefs* module imported in the *SoilTemp* sub-routine defines the complex variable *Weather*, the variable *Soil* and the parameters *Soil*. Most of the DSSAT sub-routines use this module, which contains definitions of variables that are not used in this sub-routine. One of the main patterns of DSSAT is the use of a control variable that separates the state variables initialization, rates calculation, and integration. With this design, the system generates an algorithm for rate calculation and integration processes. A model specification is associated with each algorithm. This design implies that all rates are calculated before the integration process. The soil temperature sub-routine calls a subroutine (SOILT_EPIC) that calculates, among others, the surface temperature (SRFTEMP) and soil temperature (ST) in each soil layer. In CyMLTx, the called sub-routine is transformed into a CyML function, which requires the inputs and outputs to be made explicit by the Fortran INTENT argument. The Call expression is then transformed in an assignment expression based on the position of the inputs and outputs arguments of the called sub-routine (Fig. 9). However, an analysis needs to be performed in the caller sub-routine to detect parameters provided through the import statements. These parameters are used as arguments of the generated CyML function.

4.3. Generation of Crop2ML ModelComposite

Crop2ML ModelComposite has been generated from declaratively written model composite in the case of SIMPLACE or from model composite implemented with a procedural approach in the case of BioMA and DSSAT. The declarative approach states the list of the model units and their links through their inputs and outputs while the procedural approach allows to express the model composite as a sequence of model units calls that are implemented in an imperative language.

The declarative approach simplifies the generation of ModelComposite since it provides the same level of abstraction than Crop2ML. The SIMPLACE composite components called *GroupComponent* are expressed in an XML format following a DTD. Here, the transformation consists in mapping the DTD concepts of Crop2ML and

a)

```
for (int i = 0; i < SoilTempArray.getValue().length; i++)
{
    double ZD = 0.5*(Z1 + tZp[i])/DD;
    //Factor of the depth in soil: Middle of depth of layer divided by damping depth
    double RATE = ZD / (ZD + exp(-.8669 - 2.0775 * ZD)) * (cAVT.getValue() - iSoilSurfaceTemperature.getValue());
    //RATE = Rate of change of STMP(ISL) (degC)
    SoilTempArray.setArrayValue(i, XLAG * SoilTempArray.getValue()[i] + XLG1 * (RATE +
                                iSoilSurfaceTemperature.getValue()), this);
    Z1 = tZp[i];
}
```

b)

```
for i in range(0 , len(SoilTempArray) , 1):
    ZD=0.5 * (Z1 + pSoilLayerDepth[i]) / DD
    # Factor of the depth in soil: Middle of depth of layer divided by damping depth'
    RATE=ZD / (ZD + exp(-.8669 - (2.0775 * ZD))) * (cAVT - iSoilSurfaceTemperature)
    # RATE = Rate of change of STMP(ISL) (degC)'
    SoilTempArray[i]=XLAG * SoilTempArray[i] + (XLG1 * (RATE + iSoilSurfaceTemperature))
    Z1=pSoilLayerDepth[i]
```

Fig. 7. Transformation of the SIMPLACE algorithm to Cro2ML ModelUnit algorithm: Snippet of SIMPLACE Soil temperature code showing the update of SoilTempArrayusing the setArrayValue method (a) and code of the snippet of SIMPLACE Soil temperature process method generated by CyMLTx (b).

a)

```
protected void init()
{
    Double[] Z = cSoilLayerDepth.getValue();
    double tProfileDepth = Z[Z.length-1];
    Double firstDayMeanTemp = cFirstDayMeanTemp.getValue();
    Double additionalDepth = cDampingDepth.getValue() - tProfileDepth;
    Double firstAdditionalLayerHight = additionalDepth - floor(additionalDepth);
    Integer layers = (int)(abs(ceil(additionalDepth)))+Z.length;
    Double[] tStmp = new Double[layers];
    Double[] tz = new Double[layers];
    for (int i = 0; i < tStmp.length; i++)
    {
        double depth;
        if (i < Z.length)
        {
            depth = Z[i];
        }
        else //first additional layer might be smaller than 1 m
        {
            depth = tProfileDepth + firstAdditionalLayerHight + i - Z.length;
        }
        tz[i] = depth;
        //set linear aproximation to the soil temperature as initial value
        tStmp[i] = (firstDayMeanTemp * (cDampingDepth.getValue() - depth) + cAVT.getValue() * depth) /
                    cDampingDepth.getValue();
    }
    SoilTempArray.setValue( tStmp, this);
    pSoilLayerDepth.setValue(tz, this);
}
```

b)

```
cdef float tProfileDepth
cdef float additionalDepth
cdef float firstAdditionalLayerHight
cdef int layers
cdef float tStmp[]
cdef float tz[]
cdef int i
cdef float depth
tProfileDepth=cSoilLayerDepth[len(cSoilLayerDepth) - 1]
additionalDepth=cDampingDepth - tProfileDepth
firstAdditionalLayerHight=additionalDepth - float(floor(additionalDepth))
layers=int(abs(float(ceil(additionalDepth)))) + len(cSoilLayerDepth)
tStmp.allocate(layers)
tz.allocate(layers)
for i in range(0 , len(tStmp) , 1):
    if i < len(cSoilLayerDepth):
        depth=cSoilLayerDepth[i]
    else:
        # first additional layer might be smaller than 1 m
        depth=tProfileDepth + firstAdditionalLayerHight + i - len(cSoilLayerDepth)
    tz[i]=depth
    # set linear aproximation to the soil temperature as initial value
    tStmp[i]=(cFirstDayMeanTemp * (cDampingDepth - depth) + (cAVT * depth)) / cDampingDepth
SoilTempArray=tStmp
pSoilLayerDepth=tz
```

Fig. 8. Transformation of the initialization and process methods of the Soil temperature SimComponent of SIMPLACE to Crop2ML model initialization and algorithms generated by CyMLTx, respectively: Initialization in the CyML language (a) and process in the CyML language (b).

a)

```
      CV = (BIOMAS + MULCHMASS) / 1000.
      BCV1 = CV / (CV + EXP(5.3396 - 2.3951 * CV))
      BCV2 = SNOW / (SNOW + EXP(2.303 - 0.2197 * SNOW))
      BCV = MAX(BCV1, BCV2)
      CALL SOILT_EPIC (B, BCV, CUMDPT, DP, DSMID, NLAYR, PESW, TAV, TAVG, TMAX, TMIN, WetDay(NDays),
     &     WFT, WW, TMA,X2_PREV, ST, SRFTEMP, X2_AVG)
```

b)

```
      CV = (BIOMAS + MULCHMASS) / 1000.
      BCV1 = CV / (CV + exp(5.3396 - (2.3951 * CV)))
      BCV2 = SNOW / (SNOW + exp(2.303 - (0.2197 * SNOW)))
      BCV = max(BCV1, BCV2)
      (TMA, X2_PREV, ST, SRFTEMP, X2_AVG) = SOILT_EPIC(NL, B, BCV, CUMDPT, DP, DSMID, NLAYR, PESW, TAV, TAVG,TMAX,
                                                       TMIN, WetDay[NDays - 1],WFT, WW, TMA, X2_PREV, ST)
```

Fig. 9. Transformation in CyML of a DSSAT call subroutine to assignment statement: DSSAT Fortran code (a) and CyML code (b).

SIMPLACE allowing to extract the XML elements, attributes and values of a GroupComponent and to set the fields of the ModelComposition object (Fig. 10).

An instance of the *biomaExtraction* class allows to automatically generate the corresponding Crop2ML ModelComposite specification based on the source code of the composite. The extraction method

a)

```
<configuration class="SoilTemperature">
    <simgroup>
        <simcomponent id="SnowCoverCalculator" class="SnowCoverCalculator">
            <input id="cCarbonContent" source="SoilTemperature.cCarbonContent" />
            <input id="iTempMax" source="SoilTemperature.iAirTemperatureMax" />
            <input id="iSoilTempArray" source="STMPsimCalculator.SoilTempArray" />
            <output id="SoilSurfaceTemperature" destination="SoilTemperature.SoilSurfaceTemperature" />

        </simcomponent>
        <simcomponent id="STMPsimCalculator" class="STMPsimCalculator">
            <input id="cSoilLayerDepth" source="SoilTemperature.cSoilLayerDepth" />
            <input id="iSoilSurfaceTemperature" source="SnowCoverCalculator.SoilSurfaceTemperature" />

            <output id="SoilTempArray" destination="SoilTemperature.SoilTempArray" />
        </simcomponent>
    </simgroup>
</configuration>
```

b)

```
<Composition>
    <Model name="SnowCoverCalculator" id="SoilTemperature.SnowCoverCalculator"
           filename="unit.SnowCoverCalculator.xml"/>
    <Model name="STMPsimCalculator" id="SoilTemperature.STMPsimCalculator" filename="unit.STMPsimCalculator.xml"/>

    <Links>
        <InputLink target="SnowCoverCalculator.cCarbonContent" source="cCarbonContent"/>
        <InputLink target="SnowCoverCalculator.iTempMax" source="iAirTemperatureMax"/>
        <InputLink target="SnowCoverCalculator.iSoilTempArray" source="SoilTempArray"/>

        <InputLink target="STMPsimCalculator.cSoilLayerDepth" source="cSoilLayerDepth"/>
        <InputLink target="SnowCoverCalculator.AgeOfSnow" source="AgeOfSnow"/>
        <InputLink target="STMPsimCalculator.pSoilLayerDepth" source="pSoilLayerDepth"/>

        <InternalLink target="STMPsimCalculator.iSoilSurfaceTemperature"
                      source="SnowCoverCalculator.SoilSurfaceTemperature"/>

        <OutputLink target="SoilSurfaceTemperature" source="SnowCoverCalculator.SoilSurfaceTemperature"/>
        <OutputLink target="SoilTempArray" source="STMPsimCalculator.SoilTempArray"/>
        <OutputLink target="AgeOfSnow" source="SnowCoverCalculator.AgeOfSnow"/>

    </Links>
</Composition>
```

Fig. 10. From SIMPLACE GroupComponent to Crop2ML ModelComposite: Part of Soil Temperature SIMPLACE component with two SimComponents (a). They are listed in order of calls, and each contain a list of input and output XML elements. Their source and destination XML attributes indicate the links between SimComponents; and 0art of the generated Soil Temperature Crop2ML ModelComposite (b). The internal links provide information on the order of ModelUnits. Unlike Simplace, state variables (e.g. AgeofSnow) are exposed as inputs of the ModelComposite, as the private SimComponent variables (pSoilLayerDepth).

implemented in the *biomaExtraction* class consists in retrieving the subtrees of the called methods *Estimate()* of each Simple Strategy in the body of the *EstmatedOfAssociatedClasses()* method implemented in any BioMA composite strategy (Fig. 11a). The ordered list of the subtrees gives the order of the instances of the Simple Strategy classes which allows for establishing the links between them based on their inputs and outputs. The internal links of the generated Crop2ML ModelComposite (Fig. 11b) are used to automatically produce a model graph by using Graphviz library (Ellson et al., 2002) that represents the behavior of component execution (Midingoyi et al., 2021). The ModelComposite inputs derived from the difference between the set of inputs and outputs of all simple strategies, and the Crop2ML ModelComposite outputs derived from all the simple strategies outputs that are not recalculated internally For DSSAT model composites represented as a sequence of calls of subroutines, the same process is applied to identify Crop2ML Model Links and to generate *Crop2ML ModelComposite*.

4.4. Interoperability between modeling platforms

CyMLTx addresses the challenge of modeling platform interoperability by automatically transforming with CyMLT the generated Crop2ML models into other modeling platforms. Based on these two systems, all the components used in the use cases have been made available for the platforms supported by Crop2ML (Table 2). Components have also been made available for non-specific platform in the language supported by CyMLT. In Table 2, we illustrate the interoperability of BioMA, SIMPLACE and DSSAT with 6 platforms and 6 programming languages. The code of the initial component and of the resulting transformation is available on GitHub.

4.5. Extension of CyMLTx to new modeling platforms

We extended CyMLTx to three very different modeling platforms, OpenAlea, SiriusQuality, and STICS. The transformation of an OpenAlea CompositeNode to Crop2ML ModelComposite is straightforward since the two platforms express a composite as a workflow defined as a directed graph of Nodes. However, the source components are limited to workflows without algebraic operators. The transformation approach is then similar to Simplace-Crop2ML. Each Node and the specification of its inputs and outputs are mapped with a Crop2ML ModelUnit but some concepts are missed such as the units of inputs/outputs, their category (state, rate, …), and their type (variable or parameter). This limitation is removed by describing meta-information as code comments in the function associated to each node. The CyML and Python languages are very close and only few actions need to be expressed to obtain the ASG. However, it requires to annotate Python functions with type hints before the transformation. The Python grammar written in ANTLR, available in the ANTLR Github repository, and the Python parser generated are integrated in the CyMLTx grammars repository. CyMLTx is limited to OpenAlea Node associated to pure Python function which allows to

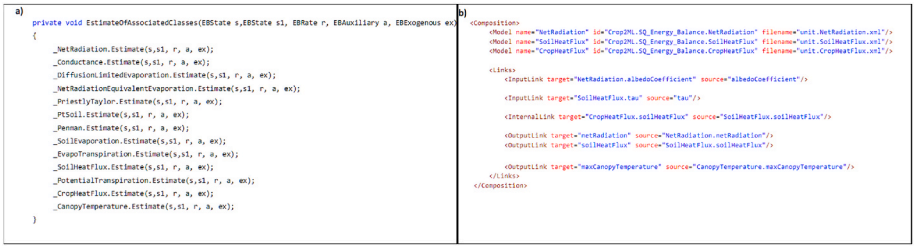


Fig. 11. From the BioMA composite strategy to Crop2ML ModelComposite: part of BioMA Energy balance composite strategy class showing the sequence of calls (a) and part of automatically generated Crop2ML ModelComposite (b).

Table 2
List of model components, language in which they are available, and links.

Model component	Source Platform	Target Language	Target platform	Link
Energy Balance	BioMA	C#, R, C++, Java, Python, Fortran	Record, DSSAT, SIMPLACE, OpenAlea, STICS, SiriusQuality	https://github.com/Crop2ML-Catalog/SQ_Energy_Balance
Soil Temperature	SIMPLACE	C#, R, C++, Java, Python, Fortran	DSSAT, OpenAlea, BioMA, STICS, SiriusQuality	https://github.com/Crop2ML-Catalog/Simplace_Soil_Temperature
Soil Temperature	DSSAT	C#, R, C++, Java, Python, Fortran	SIMPLACE, OpenAlea, BioMA, STICS, SiriusQuality	https://github.com/Crop2ML-Catalog/DSSAT_EPICST_standalone

generate the Crop2ML ModelUnit algorithm.

For STICS transformation, the Fortran 90 grammar expressed in ANTLR and its parser used for DSSAT transformation are reused since STICS components are also implemented in Fortran 90. Likewise, the fact to handle DSSAT artifacts (Control variables, IO operations) through annotations make the ASG generation reusable in STICS case. The difference of the two transformations comes from the identification and extraction of the initialization and algorithm parts. Unlike DSSAT, where initialization and algorithm are defined in the body of the subroutine through the use of DSSAT control variables, initialization and algorithm can be expressed in different subroutines in STICS. Thus, annotations are provided to make difference of the two subroutines during extraction. The composite is also implemented as the sequential calls of subroutines and the procedure of transformation is similar to DSSAT and BioMA.

Most of the SiriusQuality components were implemented in BioMA. They can also be coded as framework-independent components in the C# language. Meta-information is provided as code comments from annotations. This allows to reuse the same meta-information extraction method (getFromComments ()), as was the case with STICS and DSSAT, to produce Crop2ML ModelUnits specifications, instead of implementing another method based on a specific pattern like BioMA and Simplace. The C# parser used for BioMA models is reused to parse the source code. Likewise, the approach to generate Crop2ML ModelComposite from BioMA is reused since the composite is also expressed as sequential calls of model units.

5. Discussion

In this project we extended the CyMLT transformation system (Midingoyi et al., 2020) to automatically transform model components implemented in crop modeling platforms into Crop2ML. The proposed system is based on the analysis and translation of fragments of the source code of model components through the recognition of shared concepts. The main contributions of this project are: (1) the proposal of an architecture for model specification extraction using code comments and codebase of components provided by different crop modeling platforms; (2) the combination of source code analysis and search algorithm to interpret source code and extract information; (3) the implementation of a many-to-one transformation system (CyMLTx); and (4) the demonstration of its applicability to three platforms that use different concepts and languages (DSSAT, BioMA, and SIMPLACE), and its extension to three other platforms (STICS, OpenAlea, and *SiriusQuality*).

5.1. Advantages of the CyMLTx approach

This work has been motivated by the increasing need of crop model components exchange and reuse (Holzworth et al., 2014; Martre et al., 2018). CyMLTx allows porting a model component outside of its crop modeling platform where it has been implemented, and transforming it in the Crop2ML exchangeable format. With the CyMLTx system, components can be reused in various languages and modeling platforms. Thus, making alternative components available for different crop modeling groups, CyMLTx can contribute to the strengthening of the Agricultural Model Intercomparison and Improvement Project (AgMIP; Rosenzweig et al., 2013) efforts of crop model intercomparison and especially crop model improvement.

Instead of rewriting existing components in a new language or to develop wrappers to adapt them to the specificities of the target platforms, CyMLTx reduces the cost of reusing legacy components through automatic transformation to Crop2ML. Crop model components are codebase and are highly dependent on the modeling platforms in which they are implemented. The CyMLTx approach is based on reverse engineering to support identifying and extract component meta-information. It relies on the Crop2ML concepts to generate Crop2ML models specifications through source code analysis. CyMLTx captures the dynamics of model components represented as a pseudo-code that describes a sequential order of statements defining outputs computation at a given time step. Whatever the artifacts of the modeling platforms, the dynamic of the component can be expressed uniquely, close to its mathematical expressions. The philosophy of each modeling language and platform is well integrated into the transformation system.

The CyMLTx approach is flexible and can be extended to support other languages and crop modeling platforms. Core modules facilitate the extension of CyMLTx to new platforms. The level of difficulty to extend CyMLTx to a new platform depends on both the level of invasiveness of the platform and the ability to retrieve meta-information. For example, extracting meta-information requires less processing in SIMPLACE than BioMA. As stated above, BioMA includes meta-information in both the VarInfo files and in the strategy class while SIMPLACE provides them as arguments of a specific method that SimComponents implement. Crop2ML provides a set of guidelines for each platform to be in line with the formalisms and concepts used and allow automatic bidirectional code transformation (from platform-to-platform).

The annotation of codes to extract information or parts of code provides a robust transformation system. For instance, the use of control variables in the DSSAT components can help to identify the different concepts such as the initialization algorithm that are translated. However, they may vary from one component to another or evolve with changes in the DSSAT component design. This justifies the use of annotations that remain unchanged. In the cases of BioMA and SIMPLACE, modifying the component description interfaces or model design will cause a failure of the transformation system. CyMLTx recognizes platforms patterns to extract information and translate those patterns into their equivalent in Crop2ML. A recent line of research has focused on the use of machine learning (Lachaux et al., 2020) and natural language processing (Galanis et al., 2020) on source code. It could also potentially be beneficial to explore this domain to make CyMLTx more flexible.

Based on Crop2ML concepts and languages intersection, CyMLTx provides two transformation definitions: from Crop2ML to modeling platforms (one-to-many) and from modeling platforms to Crop2ML (many-to-one), all based on a shared representation of abstract semantic graph (ASG) and transformation rules. These two definitions lead to an interoperable system between crop modeling platforms. This approach of transformation based on Crop2ML reduces the complexity of the transformation of algorithms. Let us consider n platforms. A direct side-by-side transformation system gives $A_n^2 = n(n-1)$ transformation definitions, while our transformation system provides $2n$ transformation definitions. As the number of platforms increases, the complexity of the direct transformation increases exponentially unlike in our approach.

CyMLTx requires that platform developers incorporate model specifications into their model implementation. Automatic reuse is not possible without model meta-information. Model specification and source code should be more closely linked to infer the corresponding Crop2ML model. CyMLTx enables users to focus on the scientific aspect of their model rather than on the platform specificities. A model component can be reused, improved, integrated, and simulated on various platforms. Therefore, our system fosters the diffusion of models, sharing them as software and scientific artifacts, thus, enhancing the transparency and reproducibility of crop modeling activities.

5.2. Limitations of the CyMLTx approach

Although CyMLTx allows establishing transformation rules with different languages and platforms, several limitations exist. They are related to the CyML language limitations and Crop2ML concepts. One of the main limitations is the restriction of the transformation to stateless components. Implementing stateless or declarative components in some platforms can be challenging.

A restricted set of constructs supported and shared by the different platforms has been identified to describe the component algorithm and are defined in the CyML grammar (Midingoyi et al., 2021). The use of constructs not defined in the CyML grammar will cause the transformation system to fail. This may be a limitation for implementing complex algorithms or using composite data types. This limitation requires adapting existing model components to use only the shared constructs. The use of complex and composite variables (i.e. variables made up of two or more variables or measures highly related to one another conceptually or statistically (Ley, 1972)) is a common practice in crop model development. Complex data types could be added in future versions of Crop2ML, with some limitations related to language interoperability (e.g. matrices are not defined in C++ while they are in Fortran) and the platforms themselves (e.g. SIMPLACE does not support natively dictionaries). as for composite variable, we addressed this limitation of CyML by decomposing them into several individual variables according to Crop2ML data types. The decomposition of composite variables leads to handle a high number of input and output variables in Crop2ML, but it allows defining more explicitly the actual variables of a component. This decomposition of complex variables requires also some work to recompose the variables when integrating the component into a platform that requires such a data structure. However, most often platforms do not share the same composite variables. Adopting composite variables in Crop2ML for reuse purpose can be a real challenge.

CyMLTx handles auxiliary functions that are implemented in model components. However, for the need of modularity, several model components can share libraries of functions used to express the model algorithms and that are implemented outside them. These libraries could also be provided as compiled format. The current version of CyMLTx does not address the use of external libraries such as solver or compiled libraries to implement components. There is no strategy to manage compiled libraries since the goal of Crop2ML is to provide a white-box, self-contained component. However, the transformation system could be extended to support utility functions through the representation and management of customized import system (other than built-in module).

Our system does not support computation based on event-driven programming. The logic flow of this programming paradigm is driven by events such as messages and actions and it may not always be possible to explicitly identify the sequence of the event calls. Therefore, to achieve interoperability between heterogeneous platforms, it is necessary to keep consensus in the representation of the model components through AMEI.

5.3. CyMLTx supports model improvement, reuse and exchange

The reuse of model components requires that they are defined at a level of abstraction that facilitates their refinement or integration with other components. This challenge has been addressed in this project through the CyMLTx, which facilitates the automatic transformation of model components into Crop2ML. The generated Crop2ML models are the input source of CyMLT that produces platform specific model components. Although CyMLT and CyMLTx ensure the syntactic composability, the semantic composability and the selection of components are essential to address the challenge of model reuse (Holzworth et al., 2014). It is ultimately the modeler's task to decide which functionalities of its modeling solution or components can be exposed for sharing. Model reuse constraints should be an integral part of the modeling process, which implies that model reuse should be considered from the beginning of the modeling process and model should be modular with fine granularity to facilitate component transformation, extension, and test.

6. Conclusions

Here we presented an approach for generating Crop2ML model components from source model components implemented in different languages and crop modeling platforms. Our approach provides Crop2ML model components at a high level of abstraction that could be transformed into platform-compliant model components. It extends the Crop2ML framework and leads to an interoperable system using Crop2ML as a bridge for the reuse and exchange of model components between different crop modeling platforms. Crop2ML framework development accommodates the software engineering skillsets of framework users and handles constraints of the programming languages and software architectures of various crop modeling platforms. It gives modelers the freedom of choice of a modeling platform and the capacity to minimize the efforts required in software development for the reuse or improvement of a model component provided by another platform. The Crop2ML framework is intended to support crop model improvements and the reuse and exchange between crops models and modeling platforms of model components notably in the frame of AgMIP and other crop model intercomparison and improvement projects. Future work will develop of a semantic representations of model component composition and will also extend CyMLTx with other languages (R, C++, etc.) and modeling platforms to integrate more crop modeling groups.

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.

Data availability

No data was used for the research described in the article.

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 MathNum. P.M. acknowledges the support of INRAE Division AgroEcoSystem through the *Modélisation du fonctionnemnet des Peuplements Cultivés* (MFPC) network. C.P. received support from CIRAD's MaCS4Plants network, initiated by the AGAP Institute and AMAP joint research units, and from the EU's Horizon 2020 research and innovation program (IPM Decisions project No. 817617).

References

Argent, R.M., Voinov, A., Maxwell, T., Cuddy, S.M., Rahman, J.M., Seaton, S., Vertessy, R.A., Braddock, R.D., 2006. Comparing modelling frameworks - a

workshop approach. Environ. Model. Software 21 (7), 895–910. https://doi.org/10.1016/j.envsoft.2005.05.004.
Athanasiadis, I.N., Rizzoli, A.E., Donatelli, M., Carlini, L., 2011. Enriching environmental software model interfaces through ontology-based tools. Int. J. of Appl. Syst. Stud. 4, 94–105. https://doi.org/10.1504/IJASS.2011.042205.
Barton, C.M., Ames, D., Chen, M., Frank, K., Jagers, H.R.A., Lee, A., Reis, S., Swantek, L., 2022a. Making modeling and software FAIR. Environ. Model. Software 156, 105496. https://doi.org/10.1016/J.ENVSOFT.2022.105496.
Barton, C.M., Lee, A., Janssen, M.A., van der Leeuw, S., Tucker, G.E., Porter, C., Greenberg, J., Swantek, L., Frank, K., Chen, M., Albert Jagers, H.R., 2022b. How to make models more useful. Proc. Natl. Acad. Sci. U.S.A. 119 (35) https://doi.org/10.1073/PNAS.2202112119.
Brisson, N., Launay, M., Mary, B., Beaudoin, N., 2009. Conceptual basis, formalisations and parameterization of the stics crop model. Editons Quae.
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., Rizzoli, A.E., 2008. A design for framework-independent model components of biophysical systems, 2008. In: 4th Biennial Meeting of International Congress on Environmental Modelling and Software: Integrating Sciences and Information Technology for Environmental Assessment and Decision Making, vol. 2. IEMSs, pp. 727–734. July.
Ellson, J., Gansner, E., Koutsofios, L., North, S.C., Woodhull, G., 2002. Graphviz— open source graph drawing tools. In: Mutzel, P., Jünger, M., Leipert, S. (Eds.), Graph Drawing. Springer Berlin Heidelberg, pp. 483–484.
Enders, A., Diekkrüger, B., Laudien, R., Gaiser, T., Bareth, G., 2010. The IMPETUS spatial decision support systems. In: Impacts of Global Change on the Hydrological Cycle in West and Northwest Africa. Springer Berlin Heidelberg, pp. 360–393. https://doi.org/10.1007/978-3-642-12957-5_11.
Enders, A., Vianna, M., Gaiser, T., Krauss, G., Webber, H., Srivastava, A.K., Seidel, S.J., Tewes, A., Rezaei, E.E., Ewert, F., 2023. Simplace - a versatile modelling and simulation framework for sustainable crops and agroecosystems. Silico Plants. https://doi.org/10.1093/insilicoplants/diad006.
Galanis, M., Dietrich, V., Kast, B., Fiegert, M., 2020. RTFM: towards understanding source code using natural language processing. In: ICINCO 2020 - Proceedings of the 17th International Conference on Informatics in Control. Automation and Robotics, Icinco, pp. 430–437. https://doi.org/10.5220/0009826604300437.
Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.
Gao, F., Yue, P., Zhang, C., Wang, M., 2019. Coupling components and services for integrated environmental modelling. Environ. Model. Software 118 (April), 14–22. https://doi.org/10.1016/j.envsoft.2019.04.003.
Gleeson, P., Crook, S., Cannon, R.C., Hines, M.L., Billings, G.O., Farinella, M., Morse, T. M., Davison, A.P., Ray, S., Bhalla, U.S., Barnes, S.R., Dimitrova, Y.D., Silver, R.A., 2010. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Comput. Biol. 6 (6), 1–19. https://doi.org/10.1371/journal.pcbi.1000815.
Gyori, B.M., Bachman, J.A., Subramanian, K., Muhlich, J.L., Galescu, L., Sorger, P.K., 2017. From word models to executable models of signaling networks using automated assembly. Mol. Syst. Biol. 13 (11), 954. https://doi.org/10.15252/msb.20177651.
Holzworth, D.P., Huth, N.I., de Voil, P.G., 2010. Simplifying environmental model reuse. Environ. Model. Software 25 (2), 269–275. https://doi.org/10.1016/j.envsoft.2008.10.018.
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., 2019a. 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.
Hoogenboom, G., Porter, C.H., Shelia, V., Boote, K.J., Singh, U., White, J.W., Hunt, L.A., Ogoshi, R., Lizaso, J.I., Koo, J., Asseng, S., Singels, A., Moreno, L.P., Jones, J.W., 2019b. Decision Support System for Agrotechnology Transfer (DSSAT) Version 4.7.5. DSSAT.Net. DSSAT Foundation, Gainesville.
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., et al., 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.
Hutton, E., Piper, M., Tucker, G., 2020. The Basic Model Interface 2.0: a standard interface for coupling numerical models in the geosciences. J. Open Source Softw. 5 (51), 2317. https://doi.org/10.21105/joss.02317.
Jamieson, P.D., Brooking, I.R., Porter, J.R., Wilson, D.R., 1995. Prediction of leaf appearance in wheat: a question of temperature. Field Crops Res. 41 (1), 35–44. https://doi.org/10.1016/0378-4290(94)00102-I.
Jiang, P., Elag, M., Kumar, P., Peckham, S.D., Marini, L., Rui, L., 2017. A service-oriented architecture for coupling web service models using the Basic Model Interface (BMI). Environ. Model. Software 92, 107–118. https://doi.org/10.1016/j.envsoft.2017.01.021.
Jones, J.W., Antle, J., Basso, B., Boote, K., Conant, R., 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. Toward a new generation of agricultural system data, models, and knowledge products: state of agricultural systems science. Agric. Syst. 155, 269–288. https://doi.org/10.1016/j.agsy.2016.09.021.
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 (3–4) https://doi.org/10.1016/S1161-0301(02)00107-7.
Kramer, D., 1999. API documentation from source code comments: a case study of Javadoc. In: Proceedings of the 17th Annual International Conference on Documentation, vol. 1999. SIGDOC, pp. 147–153. https://doi.org/10.1145/318372.318577.
Lachaux, M.-A., Roziere, B., Chanussot, L., Lample, G., 2020. Unsupervised Translation of Programming Languages.
Lang, M., 2019. yggdrasil: a Python package for integrating computational models across languages and scales. Silico Plants 1 (1). https://doi.org/10.1093/insilicoplants/diz001.
Laramee, R.S., 2011. Bob's Concise Introduction to Doxygen.
Le Novère, N., Finney, A., Hucka, M., Bhalla, U.S., Campagne, F., Collado-Vides, J., Crampin, E.J., Halstead, M., Klipp, E., Mendes, P., Nielsen, P., Sauro, H., Shapiro, B., Snoep, J.L., Spence, H.D., Wanner, B.L., 2005. Minimum information requested in the annotation of biochemical models (MIRIAM). Nat. Biotechnol. 23 (12), 1509–1515. https://doi.org/10.1038/nbt1156.
Ley, P., 1972. Quantitative Aspects of Psychological Assessment: Introduction.
Manceau, L., Martre, P., Midingoyi, C., 2023. Energy Balance Component of Sirius Quality Model (V.0).
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., et al., 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.
Martre, P., Marcello, D., Pradal, C., Enders, A., Midingoyi, C.A., Athanasiadis, I., Fumagalli, D., Holzworth, D.P., Hoogenboom, G., Porter, C., Raynal, H., Rizzoli, A. E., Thorburn, P., 2018. The agricultural model exchange initiative. In: IICA (Ed.), 7th AgMIP Global Workshop.
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.
Midingoyi, C.A., Pradal, C., Enders, A., Fumagalli, D., Raynal, H., Donatelli, M., Athanasiadis, I.N., Porter, C., Hoogenboom, G., Holzworth, D., Garcia, F., Thorburn, P., Martre, P., 2021. Crop2ML: an open-source multi-language modeling framework for the exchange and reuse of crop model components. Environ. Model. Software 142, 105055. https://doi.org/10.1016/j.envsoft.2021.105055.
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.
Nigam, V., Donaldson, R., Knapp, M., McCarthy, T., Talcott, C., 2015. Inferring executable models from formalized experimental evidence. Lect. Notes Comput. Sci. 9308 (1), 90–103. https://doi.org/10.1007/978-3-319-23401-4_9.
Parr, T., 2013. The definite ANTLR 4 reference. In: The Pragmatic Programmers. https://doi.org/10.1016/j.anbehav.2003.06.004.
Peckham, S.D., Hutton, E.W.H., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12. https://doi.org/10.1016/j.cageo.2012.04.002.
Plaisted, D.A., 2013. Source-to-Source translation and software engineering. J. Software Eng. Appl. 6 (4), 30–40. https://doi.org/10.4236/jsea.2013.64A005.
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. SSDBM: Scientific and Statistical Database Management 1–6. https://doi.org/10.1145/2791347.2791365.
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.
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. In: 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. July 2006. CD ROM. Internet: Ht.
Williams, J.R., Izaurralde, C.A., 2005. The APEX Model, vol. 2. Blackland Research Center Reports.
Williams, J.R., Jones, C.A., Kiniry, J.R., Spanel, D.A., 1989. EPIC crop growth model. Trans. ASAE (Am. Soc. Agric. Eng.) 32 (2), 497–511.