WISS a Java Continuous Simulation Framework for Agro-Ecological Modelling

D. W. G. van Kraalingen(✉), M. J. Rob Knapen, A. de Wit, and H. L. Boogaard

Wageningen University and Research, Droevendaalsesteeg 3, 6708 PB, Wageningen, The Netherlands
{daniel.vankraalingen, rob.knapen, allard.dewit, hendrik.boogaard}@wur.nl

Abstract. A simulation framework is presented (WISS, Wageningen Integrated Systems Simulator) which targets the agro-ecological modelling domain. Especially simulation for a large number of locations, such as in detailed regional and global simulation studies. The framework strengths are in modularization, control, speed, robustness and computational protection (multiple system checks during simulation). The WOFOST model is currently implemented in WISS, through which it is used in a number of Wageningen University and Research projects. WISS is written in Java and the framework code is freely available.

Keywords: Simulation · Agro ecology · Modelling · WOFOST · Crop modelling · Framework

1 Introduction

In crop modelling there has been a tendency over the years for more modularization and looser coupling of software code describing the principal physiological and physical processes. The software architectures that were used had to evolve as model descriptions became more complex requiring increased modularization, understanding of reality improved and demands on speed, robustness and versatility increased (Holtzworth et al. 2015; Donatelli et al. 2010). But these trends also required solutions for communication of data among modules. In some implementations large arguments lists of variables were moved around, where in some cases people could hit the maximum of 255 arguments in some Fortran implementations. In other cases, large blocks of global storage were used (such as 'common blocks' in Fortran). Other solutions tended to introduce a software component responsible for the exchange of state and other variables by lookup in long lists. In all such solutions, ownership of state variables is with the software code where the states are calculated, but basically copies of these variables are communicated and are supplied on request by the communication component to other parts of the model, providing some greater flexibility and protection.

© IFIP International Federation for Information Processing 2020
Published by Springer Nature Switzerland AG 2020
I. N. Athanasiadis et al. (Eds.): ISESS 2020, IFIP AICT 554, pp. 242–248, 2020.
https://doi.org/10.1007/978-3-030-39815-6_23

The BioMa framework ([1]Donatelli et al. 2010) facilitates modularization and heterogenous model compositions but source code of the framework itself is not provided and its dependency on the .NET platform limits its use on non-Microsoft operating systems.

Another force behind the WISS development was the need for the WOFOST crop model (Wit et al. 2019) to be able to run for very large numbers of situations, where speed, robustness and versatility of the code are of utmost importance. An example where WOFOST as implemented using WISS is applied is the Agro Data Cube (https://agrodatacube.wur.nl/).

WISS is implemented in Java, which was chosen for its speed, language features, market penetration and machine independence (Microsoft Windows, Linux, mac OS), requiring no recompilation of the code. An installation of a JavaVirtual Machine (JVM) is required though (VMs are available for all relevant platforms).

2 WISS Approach to Agro-Ecological Modelling

In this short paper we present a different approach which has clear advantages over more traditional techniques of inter-module communication of data. In the WISS framework (Wageningen Integrated Systems Simulator), we have developed a technique whereby states are not dispersed and kept permanently in software code describing model processes (the classic approach) but are kept in a special object which manages all states. This approach we call the "shared state approach".

Having all states together in one object during the whole simulation implicates the huge advantage that this object can be queried not only during simulation, but also after simulation. Not only can final values be obtained through simple functions, but also more advanced functionality like the highest and lowest value of a state variable during simulation, the change in value between the first and the last date, the average value and more can be programmed in a very generic way even without knowledge of the names of the state variables!

A consequence of the centralized state principle is that a module has to obtain the value(s) of its own state variables from the central object before starting calculations for the new time step, next to obtaining the value for other state variables it may need, but does not own. The calculations provide rates of change to the central state object. At the start of the new time step, the central state object integrates all states for which it has received rates. Having this kind of architecture enables us to override state variables to specific values without any code change in the module that calculates its rates. In fact, there is no way of knowing for this module whether a state value it received was overridden or not.

The central state object has several advanced features that help to maintain simulation integrity and support modularity. First of all, it accepts states in a particular unit and deliver it in another unit to a module requesting it, of course only if the unit conversion is valid. This improves flexibility in coupling modules together into one

[1] Biophysical Model Applications, https://en.wikipedia.org/wiki/BioMA.

model, whereby each module can have its own set of units (which may differ from the units used by another module in the system), e.g. weight per area conversions from kg. ha-1 to g.m-2, or temperature conversions from degrees Celsius to Kelvin.

Second, the central state object can also safeguard calculations during simulation as the state variables can be registered with a valid range (e.g. zero to infinity, between zero and one etc.). Violation of these bounds will be detected by the central state object on acceptance of the rate of change of the state variable and simulation will be interrupted. Running the model with well-established bounds results in a higher quality model implementation and more confidence in the final results. Calculations can also be protected against accidental changes of state variables by other model components, through a strict ownership mechanism.

Finally, the central state object also enables us to suspend and resume the calculations of a model on a particular date during simulation, since the state of the system is completely defined by the states in the central object. This is an important feature particularly for more complicated models, for instance in the case of simulations that track near real time weather, as the simulation can start where it left off in the previous run.

Several other ambitions led to the development of the WISS framework. We wanted a framework in which the model can be called as a fast numerical function without any file based input and file based output. This is an essential requirement for models intended to run for a large number of geographically different units, e.g. in a distributed computing environment. We also wanted a framework in which model components can be run together with as little code changes to the model components as possible, and we wanted to have great flexibility in making those compositions (run a water balance for a bare soil, or one with a crop on top of it, rotations etc.). Flexibility was also required in starting and stopping subsystems during one simulation period, e.g. a crop on top of a soil water balance. Obviously, the crop module needs to be terminated at harvest (the crop is gone), but the water balance must continue to run.

3 Time Steps and Integration

WISS is targeted at the agro-ecological modelling domain. In this domain, daily time steps in numerical integration are most common (basically driven by the nature of daily weather data, hourly being much harder to obtain). Systems with flexible (=variable) time steps tend to introduce a level of complexity that would make programming a WISS model much less straightforward, and would slow down development of the initial framework. Simulation with flexible time steps may be introduced in future versions, should the need arise. Concurrent with daily time steps, rectangular integration (Euler) is the provided integration method. Remember it is the central state object doing the actual integration.

4 WISS Model Components

Besides the WISS framework components, a model implemented in WISS must have at least one so called SimObject, and one so called SimController. A SimObject is the place where the calculations for a model component are programmed, providing

separation of calculations. It normally consists of a section where the states it will produce are registered and a place where the states it needs are registered. There must also be a rate calculation section where the rate of change for each state variable must be calculated. Prior to the calculations, the latest values for the required states must be obtained from the central state object, which is called SimXChange. A SimObject, however, does nothing while the model is running unless it is started (=instantiated) by a so called SimController which's sole responsibility is to start and stop one or more SimObjects, providing separation of control. Not only can there be one or more SimObjects, there can also be one or more SimControllers (e.g. one for 'seeding' a crop, one for harvesting a crop).

A SimObject which is not started will do nothing, but if started it will need model parameters and initial states to properly initialize. This is where another important WISS model component kicks in, ParXChange. This component works like a key value list which is meant to hold parameters by name and is able to accept and provide the numerical value for that name. Similarly, for initial values for states, these are given to the ParXChange component, which provides it to SimObjects requesting it. Typically a ParXChange object is filled with data prior to starting the simulation.

The central state object is called SimXChange in WISS terminology. It is empty at the start of the simulation but gets filled with data as simulation proceeds. During simulation it constantly accepts rates of change and provides values to the running SimObjects. After termination of the model, it is loaded with all simulated data of every time step and ready for final processing or exporting the results of part of whole of the simulation period.

The simulation loop, in simplified form is given in Fig. 1.

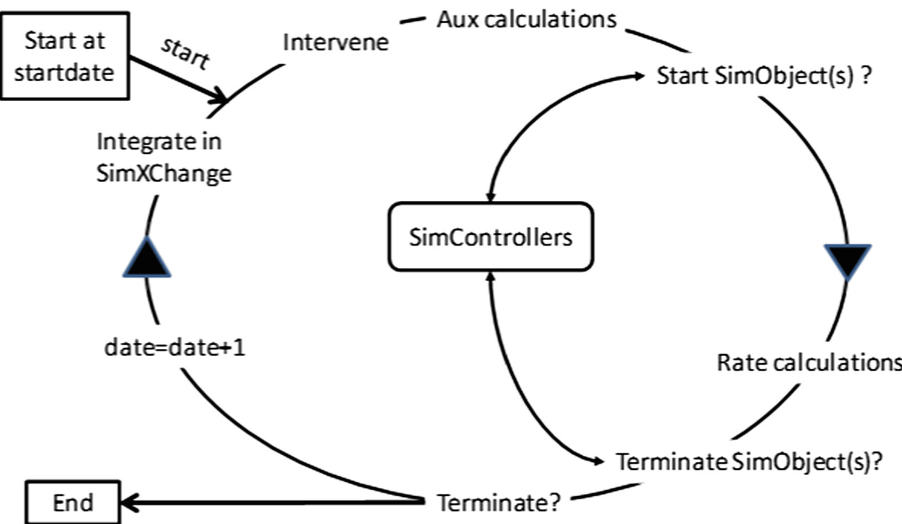

Fig. 1. The simulation loop in WISS

The schema in Fig. 1 shows the sequence of events that take place in WISS while the model is running. The implementation in code is in the TimeDriver class. Execution is in clockwise order, starting from "Start at start date". The following steps are made:

- Start one or more SimControllers (depending on the model, not shown here for clarity). These will oversee the simulation and start and stop the SimObject(s) if necessary (but starting is actually done later in the loop).
- Intervene:
  The Intervene step is an opportunity for every running SimObject in the system to override (=force) any state variable in the provided SimXChange object. Here adding or taking away part or whole of a state variable is allowed (if within the registered bounds). The provided SimXChange object will report these forcings in a special report with the date of overriding, the old and the new value. Examples are an external mowing event in case of grass simulation, a crop growth model in which the leaf area index needs to be forced by the values from a field experiment.
- Aux calculations:
  All running SimObjects are called to provide, if it is the SimObject's responsibility, time dependent driving data to provided SimXChange for SimObjects to use during the rate calculation step. Examples are air temperature, air carbon dioxide concentration etc. After this point, all existing states and time dependent driving data are up to date for the current simulation date!
- Start SimObject(s)?:
  All running SimControllers are asked whether additional SimObject(s) must be started. All started SimObject(s) also have their AuxCalculations method called (not shown here). At this point the system is up to date for the new states and auxiliary variables for the current simulation date.
- Rate calculations:
  All running SimObjects are called to carry out rate calculations and publish those to SimXChange.
- Terminate SimObjects(s):
  Call all controllers of the model and let them terminate the SimObject(s) which need to be terminated (through evaluation of states etc.), also ask remaining running SimObject(s) whether they can continue (method: SimObject.canContinue), if not, terminate them too.
- Terminate?:
  Terminate time loop if there were any previous running models but there are none anymore, or the simulation's end date has been reached. If either of that happens, any running SimObject(s) are terminated, the time loop ends, and post processing can take place.
- Date = Date + 1:
  Date increase if the loop is not terminated.
- Integrate in SimXChange:
  The internal states of SimXChange are integrated only if a rate has been provided. If the rate of change for a state has not been provided, the state will be missing from the new date until the end date. An error will occur if a SimObject tries to 'revive' a state variable by providing a rate. The state can only exist for one contiguous period.

5 Implementation Aspects

The above mentioned features are nice but we also strived for excellent execution performance. This is achieved by introducing a registration mechanism for state variables whereby the native unit and value bounds have to be provided. This registration returns a token which essentially contains the internal SimXChange's fixed array location of the state variable so that when communication takes place with SimXChange using this token the central state object immediately knows the name and other attributes of the state variable.

WISS is designed to safeguard valid simulation as much as possible. In general simulation will be terminated by a run-time exception whenever something goes wrong. Beit a non-existing unit conversion, a bounds check error, a required external variable not there etcetera. The principle being that if an error occurs, simulation results are unreliable anyway, so there is no need to continue. Best is to present the error with as much information as possible to the user, so the error can be located easily and repaired.

Extensive logging features are available in the WISS framework for the error, warning, info, debug and trace level.

6 Availability

The current version of the WISS framework is version 1 and it is available as open source software but the exact license still needs to be discussed (request the author of this paper for more information). WISS is available on https://github.com/DanielVanKraalingen/.

Currently 2 models have been implemented in WISS of which WOFOST is the most complicated one, the other one being a Lotka/Volterra prey predator model for demonstration purposes.

The current version of WOFOST (for which you'll need the WISS framework) is 7.2 but we will not automatically provide the Java source code. However, you can work with the jar file enabling you to run the WOFOST model with all valid inputs and program all possible outputs because the user has full control of the SimXChange object after simulation.

7 Real World Application

The WISS version of WOFOST is applied within the EU funded project AGINFRA+. AGINFRA+ is a D4Science Virtual Research Environment (VRE) and a use case was developed to perform parcel specific crop simulation taking data from the AgroDataCube. The AgroDataCube provides a large collection of open data at parcel level for use in agri-food applications in the Netherlands (https://agrodatacube.wur.nl/). Through the VRE, the user has access to a computer cluster, of scalable size. The DataMiner core component of the D4Science platform provides handles adding and running algorithms on the cluster, while also making them available through OGC WPS (Web

Processing Service) interfaces. On top of that, a dashboard has been created to select crop parcels, activate WISS-WOFOST and visualize and inspect results.

8 Future

We are currently finalizing the freely available manual of the WISS framework version 1.0. With that manual, and the software for WISS 1.0 (including the prey/predator example) you should be able to start your own WISS model. Contact the author for more details.

We intend to expand WISS as a modelling framework as well as expand the process descriptions of WISS-WOFOST with a true multi-layer water balance model.

References

Donatelli, M., et al.: A component-based framework for simulating agricultural production and externalities. In: Brouwer, F., Ittersum, M. (eds.) Environmental and Agricultural Modelling. Springer, Dordrecht (2010). https://doi.org/10.1007/978-90-481-3619-3_4

Holzworth, D., et al.: Agricultural production systems modelling and software: current status and future prospects. Environ. Model Softw. **72**, 276–286 (2015). https://doi.org/10.1016/j.envsoft.2014.12.013

de Wit, A., et al.: 25 years of the WOFOST cropping systems model. Agric. Syst. **168**, 154–167 (2019)