ELSEVIER

Agricultural Systems 70 (2001) 421–443

AGRICULTURAL SYSTEMS

www.elsevier.com/locate/agsy

Approaches to modular model development

J.W. Jones[a,*], B.A. Keating[b], C.H. Porter[a]

[a]*Agricultural and Biological Engineering Department, University of Florida, PO Box 110570, Gainesville, FL 32611, USA*

[b]*APSRU/CSIRO Tropical Agriculture, 306 Carmody Road, St. Lucia, 4067, Australia*

Accepted 30 March 2001

Abstract

One of the main goals of the International Consortium for Agricultural Systems Applications (ICASA) is to advance the development and application of compatible and complementary models, data and other systems analysis tools. To help reach that goal, it will adopt and recommend modular approaches that facilitate more systematic model development, documentation, maintenance, and sharing. In this paper, we present criteria and guidelines for modules that will enable them to be plugged into existing models to replace an existing component or to add a new one with minimal changes. This will make it possible to accept contributions from a wide group of modellers with specialities in different disciplines. Two approaches to modular model development have emerged from different research groups in ICASA. One approach was developed by extending the programming methods used in the Fortran Simulation Environment developed in The Netherlands. This method is being used in revisions of some of the Decision Support Systems for Agrotechnology Transfer crop models. A simple example of this approach is given in which a plant growth module is linked with a soil water balance module to create a crop model that simulates growth and yield for a uniform area. The second approach has been evolving within the Agricultural Production Systems Research Unit group in Australia. This approach, implemented in software called Agricultural Production Systems Simulator, consists of plug-in/pull-out modules and an infrastructure for inter-module communication. The two approaches have important similarities, but also differ in implementation details. In both cases, avoiding reliance on any particular programming language has been an important design criterion. By comparing features of both approaches, we have started to develop a set of recommendations for module design that will lead to a ‘toolkit’ of modules that can be shared throughout the ICASA network. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Crop models; Simulation; Modularity; Model structure; FSE/FST; DSSAT; APSIM

* Corresponding author. Tel.: +1-352-392-8694; fax: +1-352-392-4092.
E-mail address: jjones@water.agen.ufl.edu (J.W. Jones).

1. Introduction

Simulation models integrate knowledge from different disciplines and provide researchers with capabilities for conducting computer experiments to supplement actual experiments. They have evolved and are now used by researchers world-wide. However, they have yet to reach their potential that some predicted when they were first introduced. Development of models seems to have peaked in the last few years; most still incorporate only a few of the many factors that affect crop yield in farmers' fields. Whereas many of the most widely used crop models include climate, soil water availability, and soil nitrogen effects, they can not simulate actual yields when other factors, such as weeds, diseases, insects, tillage, phosphorus, etc. are limiting. Increasingly, issues related to sustainable use of natural resources and protection of the environment require attention, therefore extending the need for analyses beyond crop productivity. This creates a demand for comprehensive models that can be used to study tradeoffs between different, and often conflicting, goals of decision makers. Some model developers have attempted to add components to their crop models to describe these complex interactions. However, such efforts have generally led to models that are highly complex, unwieldy and virtually impossible to document and maintain. For a number of years, agricultural scientists have suggested that the crop modelling community adopt a modular structure to help solve these problems. However, this has not occurred in any co-ordinated way. Creating a modular model development framework acceptable to a wide group of modellers has not been easy.

One of International Consortium for Agricultural Systems Applications (ICASA's) main goals is to promote the most effective and efficient development and use of models of agricultural systems. Considerable progress has been made on the definition and documentation of data formats and files for use in crop simulation models. ICASA v1.0 Data Standards (Hunt et al., 2001) have been developed to encourage the compatibility of datasets for documenting experiments and for providing those data in formats that can be widely exchanged and used by different groups. These so-called data standards are not intended to replace all other approaches for storing data. Instead, they are intended to provide a convenient way for all who are interested in such data to access it and use it for their own purposes. We have found that the standards facilitate co-operation among experimentalists, model developers, and model users.

A second ICASA initiative is to investigate approaches for modular model development. A modular approach is needed for several reasons. It will help model developers add new components to include new factors, such as pests and diseases, with minimal changes to existing code. This ability is urgently needed to allow easy substitution of components for evaluating alternate model formulations. It would allow model developers to update documentation and to maintain code much more effectively. One goal of a modular approach is to allow scientists in different disciplines to develop modules using their knowledge, data, and expertise and not be burdened with development and maintenance of code for other components.

Different approaches for modular model development have emerged from three research groups in ICASA: (1) the 'School of de Wit' (Bouman et al., 1996); (2) the

Decision Support System for Agrotechnology Transfer (DSSAT) group (Tsuji et al., 1994; Jones et al., 1998), and (3) the Agricultural Production Systems Research Unit (APSRU; McCown et al., 1996). Each of these groups incorporated modular approaches in their respective models at different levels using different methods. Thus, attempts to compare models or other components among these and other groups have been very difficult. In this paper, a derived approach is described, which is based on the programming methods used in the Fortran Simulation Environment developed in The Netherlands by Van Kraalingen (1995). This method is being used in some new revisions of DSSAT crop models. An example of this approach is summarized in which simple modules of soil water and plant growth are linked to create a crop model that simulates growth and yield for a uniform area. Another significant approach has been evolving within the APSRU group in Australia. This approach, implemented in software called APSIM, consists of plug-in/pull-out modules and a protocol and infrastructure for inter-module communication. These two approaches have similarities, but they also have important differences in implementation details. In this paper, we compare features of approaches used by these groups, and conclude with key elements for module design that will facilitate a more effective and efficient evolution of application-oriented crop models.

2. Modularity criteria

Crop models should be modular such that new components can be added, modified, and maintained with minimal effort. This would greatly facilitate our ability to integrate knowledge from different disciplines and move models toward predicting actual as opposed to potential yields. Acock and Reynolds (1989) and Reynolds and Acock (1997) proposed criteria for a generic modular structure for crop models. Three of their criteria were: (1) the modules should separate easily along disciplinary lines (i.e. for crop, soil water, soil temperature, soil nitrogen, soil phosphorus, management, insects, diseases, etc.); (2) they should have a minimum number of input and output variables; and (3) modifying one module should not necessitate changing another module. By extending and generalizing these criteria, we suggest that an effective modular approach should:

1. Lead to components that can be plugged in or unplugged with little impact on the main program or other modules.
2. Facilitate comparisons of different models and components of models.
3. Facilitate the ability to integrate components from different authors as building blocks for expanding the scope and utility of models.
4. Facilitate documentation and sharing of code.
5. Enable models written in different programming languages to be linked together.
6. Allow greater flexibility in model updates and maintenance.
7. Extend the life and utility of simulation models.
8. Enhance the opportunities for collaboration amongst model development groups.

There are, however, many ways to meet these criteria, many languages that could be used and even different ways to define modules. Nevertheless, these are desirable criteria for developing models of crops or other systems. We found that approaches used for model development by the three groups met some or all of these criteria within the respective groups, however, major differences in approaches among the groups result in a lack of compatibility when one attempts to extend these criteria beyond individual groups. Furthermore, although one can claim that each group's approach meets these criteria within their own group, the amount of additional programming needed to satisfy them varied considerably as well. We wanted first to determine how the current approaches differ, then make suggestions for guidelines for meeting the above criteria, aimed at increasing modularity within and across groups. If each model development activity was to adhere to an agreed set of guidelines, modules could be integrated into more comprehensive models of cropping systems as needed for specific applications. This would allow modules to be developed, documented, and maintained at different locations.

3. Models from the 'School of De Wit'

3.1. Background

The origins of crop simulation models from the 'School of De Wit' (Bouman et al., 1996) were in the classical publication on modelling photosynthesis of leaf canopies (De Wit, 1965) and its use in the Elementary Crop Simulator, or ELCROS (De Wit et al., 1970). ELCROS later led to the more comprehensive BACROS (De Wit et al., 1978; Penning de Vries and Van Laar, 1982) crop model, which simulates potential growth and transpiration of a crop's vegetative phase to gain insight into these processes. These early models were forerunners of several versions of the Simple and Universal Crop growth Simulator, or SUCROS (Van Keulen et al., 1982), which were aimed toward practical applications, such as studies of climate effects on production and water management. Several versions of SUCROS were published for specific purposes, each building on the original model (Spitters et al., 1989; Van Laar et al., 1992; Goudriaan and Van Laar, 1994). Models for a number of crops (e.g. wheat, potato, soybean, maize, and sugar beet) were included in different versions of SUCROS by altering crop-specific parameters. SUCROS also led to the development of INTERCOM (Interplant Competition, Kropff and Van Laar, 1993), which simulates interactions of weeds and field crops. The MACROS (Modules of an Annual Crop Simulator, Penning de Vries et al., 1989) model was developed as part of the Simulation and Systems Analysis for Rice Production (SARP) project to facilitate the transfer of simulation and systems analysis methodologies to researchers in southeast Asia (Ten Berge, 1993). MACROS provided users with a development tool for developing and applying models for different applications, including the management of water, nutrients, and pests. The ORYZA rice production models (Kropff et al., 1994) evolved from MACROS and SUCROS in this project to serve specific applications. These models have had a large impact on model development in many other groups and are now widely used world-wide for different applications. Bouman et al. (1996) provide a more detailed history and explanation of these and other crop models developed by this group over the years.

3.2. *Modularity in 'School of De Wit' models*

A high degree of modularity was designed into many of the models developed in the 'School of De Wit'. Models or components of models evolved through time around a sound set of programming principles and small set of programming tools and languages. Many of the early crop models from this group were programmed in the dynamic simulation language named CSMP (Continuous Simulation Modeling Program; IBM, 1975). This language was similar to Fortran, and actually produced Fortran code, which was subsequently compiled and run. However, this language was discontinued, and the Fortran simulation translator (FST) was developed at the Wageningen Agricultural University by Rappoldt and Van Kraalingen (1996) to replace it. FST translates CSMP statements into Fortran code with a highly modular structure. This structure has been developed and documented by Van Kraalingen (1991, 1995) as the Fortran simulation environment (FSE).

FSE is a set of Fortran utilities and programming protocols for simulating deterministic, continuous dynamic systems. It provides users with all of the power of programming in Fortran and a structure that greatly facilitates development of modules that can be used by other models programmed in the same environment. There are several principles that guided the development of FSE. First, the scientific part of a model is separated from the non-scientific overhead. This way, model developers can concentrate on the scientific components and develop modules that are linked with the so-called FSE driver program. The driver controls simulated time, provides access to weather data from files, and correctly integrates various modules that may be present (Van Kraalingen, 1995). A second principle is that the complex functionality has been hidden in utility routines. Thus, an FSE model consists of an FSE driver, utility routines, and the model programmed in Fortran with a prescribed structure. Standard Fortran is used to ensure portability of the code and to minimize the chances of obsolescence. SUCROS, ORYZA and other models that are programmed in FSE have taken advantage of the modularity by re-using code and creating libraries of modules to help teach new modellers how to develop and apply models.

What makes FSE so powerful is the structure required for each model and the control of the sequence of calculations by the driver program. Each model has four sections: initialization, rate calculations, integration, and termination. The driver program initializes each module, then starts the time loop. Inside the time loop, driving variables are set and rates of change of all state variables in each module are computed, variables are written, time is updated, and all state variables in each module are then updated using Euler integration (Van Kraalingen, 1995). Fig. 1 shows the sequencing of different tasks in FSE and how time is updated in the sequence. This structure facilitates the development and publication of compatible modules (e.g. Van Kraalingen and Stol, 1997). The limitations to this FSE are the

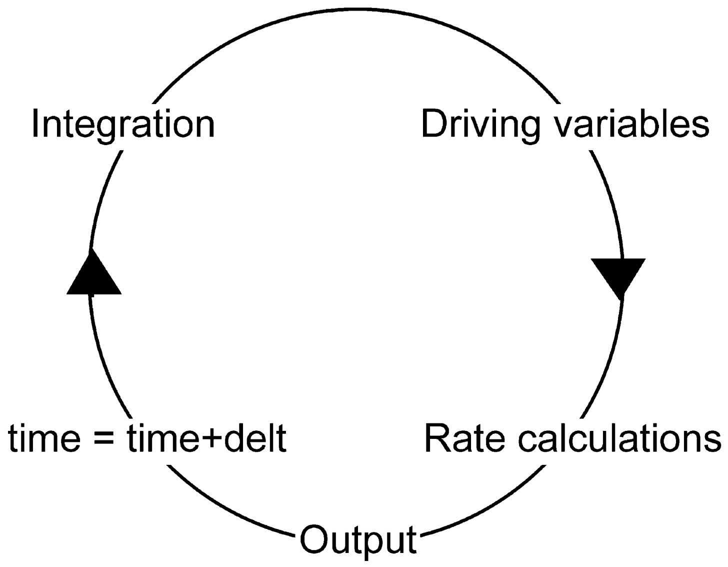

Fig. 1. Schematic of the simulation program sequence of calculations in Fortran simulation environment to facilitate modular implementation of crop models (Van Kraalingen, 1995).

requirement of Fortran as a programming language and the requirement that model developers learn the structure and available utilities.

4. The DSSAT family of crop models

4.1. Background

The Decision Support System for Agrotechnology Transfer (DSSAT) is the major product of the IBSNAT (International Benchmark Site Network for Agrotechnology Transfer) project, initiated in 1982 (Uehara and Tsuji, 1998). Although this project ended in 1993, its developers have expanded since then, and they continue to update and maintain this software under the auspices of ICASA. DSSAT was designed to enable its users to apply systems analysis and simulation to evaluate outcomes and risks of alternative management choices. The central components of the DSSAT software are crop simulation models and programs to facilitate their application in different regions of the world. Components allow users to: (1) input, organize, and store data on crops, soils, and weather; (2) retrieve, analyse and display data; (3) validate and calibrate crop growth models; (4) simulate different crop management practices; and (5) evaluate economic risks associated with different options (Jones et al., 1998).

One of the main characteristics of the IBSNAT project that differed from experiences in the APSRU and 'School of De Wit' groups was that developers of the crop models, application programs, and the integrated DSSAT software package were widely dispersed. They belonged to different institutions in different countries. This situation turned out to have both advantages and disadvantages. An advantage was that by combining efforts, considerably more was accomplished than was possible by any one group. In addition, the project realized from the start that a minimum data set was needed to allow model transportability across regions, and that formats and protocols for data storage and use would need to be standardized. Participants had already developed crop models but these were incompatible when the project started. This realization led to the development and documentation of a minimum data set and data standards (Hunt and Boote, 1998). The fact that all of the models and other components in DSSAT use the same inputs and outputs allowed participants to design components for performing a wide range of functions in DSSAT (Jones et al., 1998). Disadvantages include the difficulties in co-ordinating such an undertaking and the resulting multiplicity of program languages and screen designs implemented by different contributors. Most of the component developers were agricultural researchers from different disciplines. This was crucial in ensuring credibility of the models and the accuracy of application programs. However, these efforts took far more time than originally envisioned and diverted attention from scientific issues to those related to software, standards, and compatibility.

Four versions of DSSAT have been released since 1989, the most recent one in 1998 (V3.5), 5 years after the end of the IBSNAT project. This version has models of 15 crops, programs to analyse biophysical and economic risks of different practices, and programs to analyse the spatial variability of crop production at field and regional scales (Jones et al., 1998). DSSAT has been distributed to more than 1000 researchers in 90 countries, it has been used in training workshops world-wide, and it has been applied to study food production problems in many countries.

4.2. Modularity in DSSAT v3.5

There are two levels of modularity in DSSAT. The first level allows crop models to be plugged into the software, if they adhere to its data standards and protocols for inputs and outputs (Jones et al., 1998). Because the analysis programs also adhere to these standards and protocols, users can analyse results from any compatible model added to the system. The second level of modularity is inside the crop models themselves. Each of the models incorporated in the released software has common components for soil water, soil nitrogen, weather, and sensitivity analysis, and they are operated by a common 'driver' program. However, accomplishing this level of compatibility has required considerable programming because of the different sets of crop models in DSSAT. All of them belong to or were derived from the CERES (Ritchie et al., 1998) or CROPGRO (Boote et al., 1998) families of crop models, with a total of seven separate executable programs. Each set of code has to be changed any time that changes are made in any of these components, making it very difficult to maintain them and DSSAT. The DSSAT group has recognized the need for better modularity for reasons explained in this paper. As a result of early ICASA studies of modular modelling approaches, a major effort is now underway to revise the CROPGRO model into a modular structure similar to that described by

Van Kraalingen (1995). Table 1 shows the major modules and sub-modules developed in this effort, each of which has the recommended features described later in this paper.

5. APSIM models

5.1. Background

The Agricultural Production Systems Research Unit (APSRU) in Australia formed in 1991 from the respective modelling efforts of CSIRO (AUSIM, McCown and Williams, 1989) and the State of Queensland (DPI and DNR; PERFECT, Littleboy et al., 1989). These efforts had in turn been influenced by various modelling efforts in the 1980s, including CERES (Jones and Kiniry, 1986), SORKAM (Rosenthal et al., 1989) and EPIC (Williams, 1983). Both groups that came together to form APSRU had been progressively moving from the application of stand-alone crop models to cropping systems models that addressed longer term issues of soil and crop productivity and sustainability. Both groups had paid scant attention to software design, and had progressively added capability and complexity to their models to address new R&D challenges faced in the 1980s. By 1990, these software development efforts were clearly unsustainable. The software was increasingly unstable and inflexible. The code was poorly designed, containing all the worst features of 'spaghetti' Fortran. The addition of a new capability was a difficult exercise as it frequently led to failure of other aspects of model performance (the 'ripple' effect). Error tracking, model testing, version control and documentation were

Table 1
Modules and sub-modules created for the new modular version of CROPGRO. The sub-modules are components of the modules, but each also has the characteristics of a module as outlined in this paper

Module type	Modules	Sub-modules
Biological	Plant growth and partitioning Phenology Photosynthesis Pest damage	Growth demand, nitrogen uptake, nitrogen fixation, grain growth, fruit abortion, vegetative growth and partitioning, senescence, root growth
Environmental	Weather Soil water balance Soil nitrogen balance Soil temperature	Weather modification, weather generation Surface water runoff, infiltration, saturated flow, root water uptake, potential evapotranspiration, soil evaporation
Management	Planting Harvesting Irrigation Fertilizer Residue	

poorly addressed. The code was highly personalized and only a few individuals could effectively work with it. When APSRU formed in 1991, it was recognized that a new approach to model development was needed. This approach needed to reconsider the design of 'systems' simulation, instead of simply continuing to add additional complexity to individual crop or soil models. We also recognized that more attention was needed to good software development principles, although it took over 5 years to fully understand what this meant. The Agricultural Production Systems Simulator (APSIM, McCown et al., 1996) became the modelling platform for APSRU's efforts on modeling agricultural systems. Modularity was an important means by which APSRU's software addressed these two needs.

5.2. *Modularity in APSIM*

There are two ways in which modularity in APSIM can be viewed, namely a systems view and a software view.

5.2.1. *The systems view*

Any system can be broken up into a series of sub-systems or components. These sub-systems are characterized by a set of state variables, incorporate a set of processes, respond to an external environment, and interact with other sub-systems. APSIM has been developed as a series of separate modules that link together via a communications framework to represent the overall system being simulated (Fig. 2). The design is generic to any simulation domain, but the focus to date has been on agricultural systems. The communications framework (sometimes called the engine) is a key element of APSIM's modularity. It consists of module interfaces that supply

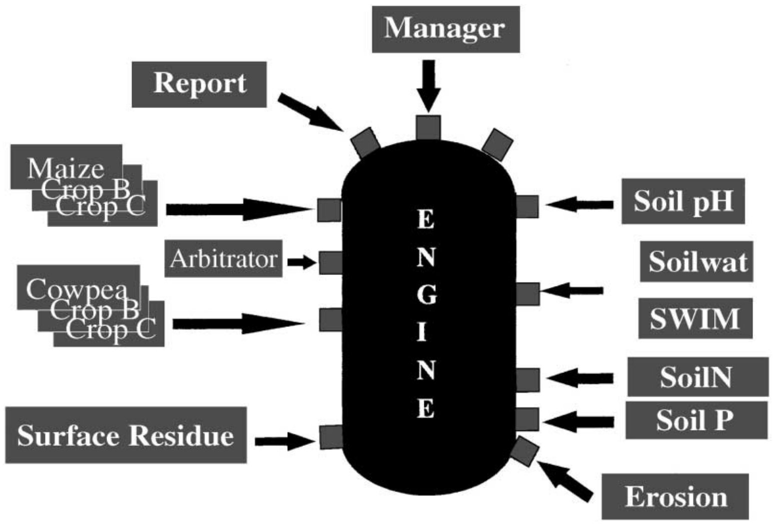

Fig. 2. Schematic of Agricultural Production Systems Simulator, showing modules as they are plugged in or unplugged from the 'Engine' or main driver (from McCown et al., 1996).

variables and events from within modules to the wider simulation and a messaging facility that controls all inter-module communications.

Originally developed in FORTRAN 77, APSIM has recently been converted to FORTRAN 90. Modules are now present as dynamic linked libraries, which need no longer be written in Fortran (although this remains the most common language for modules). Increasingly the APSIM high level infrastructure is being written in c++. While the software design is very flexible, science considerations often impose restrictions. The abstraction of data and processes into discrete modules involves both science and software engineering perspectives. APSIM modules have largely been structured around major biological, environmental or management elements (Table 2).

From a scientific point of view it is convenient to design modules around individual biological components (e.g. a crop or pasture species) and major environmental constraints (e.g. soil water, soil N, soil P, etc). System features such as surface residues are simulated via a common module, which receives inputs from all plant modules and transfers mass (C and nutrients) to soil modules. Major soil processes (such as soil water, soil nitrogen, erosion and acidification) are dealt with via separate modules that communicate extensively with the wider system. Major management issues are also addressed via separate modules. These include both 'software management' matters such as simulation time stepping (CLOCK module) and data input and output (MET, INPUT and REPORT modules), as well as system management issues (e.g. FERTILISE, IRRIGATE, OPERATIONS and MANAGER). Although the APSIM interface is a unique design, it is a relatively simple matter to add the required interface code to the code of some other model, provided the overall abstraction in this other model is compatible with that deployed in other necessary APSIM modules. The overall abstraction and interface for modules has been developed by the APSRU design team.

5.2.2. The software view

In designing modules from a software point of view, the elimination of code duplication is a major consideration. For example, if multiple modules need soil

Table 2
Major APSIM modules

Module type	Module names
Biological	Maize, Wheat, Barley, Sorghum, Millet, Sugarcane, Sunflower, Canola, Chickpea, Mungbean, Cowpea, Soybean, Peanut, Navybean, Fababean, Stylo pasture, Lucerne, Cotton (OzCot)[a], Native pasture (GRASP), Hemp, Pigeonpea[b], FOREST[c]
Environmental	SoilN, SoilP, Soilwat, Solutes, Soil pH[c], Residue, Manure[b], Erosion, SWIM[c]
Management	Manager, Fertilize, Irrigate, Accumulate, Operations, Canopy, Micromet, Clock, Report, Input, Met

[a] In association with CSIRO Plant Industry.
[b] In association with ICRISAT.
[c] In association with CSIRO Land and Water.

temperature, a soil temperature module would be created to communicate with all other modules. These modules incorporate the standard interfaces that enable their connection to the simulation engine, in exactly the same manner as do other biological, environmental or management modules.

This software engineering perspective of modularity addresses issues of code construction within what is defined as a module in the APSIM system. In this case, we are not referring to modularity that is achieved via APSIM's common module interface and communication system, but to a lower level modularity achieved via various structured programming devices. Two examples of this lower level modularity are extensively deployed in APSIM. One involves the extensive use of custom-built function calls within the code to deal with calculations that are commonly repeated in modules. Some examples of such functions with self explanatory roles include functions such as remove_from_real_array, stage_is_between, on_day_of, find_stage_delta, and find_layer_no.

The other example of lower level modularity in APSIM involves the development of a library of sub-routines that are commonly used in biological modules. For example, a crop library contains subroutines for a diverse range of routines used in many crop models, such as thermal time and radiation interception calculations. This library is compiled as a dynamic link library and the routines are available for deployment in all crop, pasture or tree modules. At present, approximately 50% of the executable code in APSIM's crop modules is made up of generic routines contained within the crop library. These routines are usually parameterized with different data that are supplied via crop module initialization files. This feature in APSIM modules is similar to the concept of utility routines in FSE.

5.2.3. The APSIM engine and associated infrastructure

The developers of APSIM have placed considerable investment into what has been called APSIM's Engine (Fig. 2). This refers to all the programming infrastructure used to link the modules and drive the simulation forward. There is no fixed time step implicit in APSIM's engine. Historically, the major time-stepping mechanism was the time interval provided by the inputs to the MET module, which has generally been daily. Increasingly, intermodule communications are treated as 'Events', as this has been found to deliver greater flexibility. Different modules can operate on different time steps (e.g. the SWIM module of water and solute movement operates on variable and sub-daily time step within the daily time step of other modules). A 'CLOCK' module is available for all modules to request the time to enable synchronization when required. APSIM's infrastructure has the notion of within-time step 'phases', such as initialization, pre-process, process, post-process. These phases deliver similar functionality to the initialization, calculation, integration, and termination sections in an FSE program (Fig. 1).

5.2.4. Applications of APSIM to complex farming systems

APSIM's modularity has given it great flexibility. This is best demonstrated via the diverse range of uses to which it has been put over the last 7 years. For example, studies have been conducted on drought and water management (Asseng et al., 1998; Muchow and Keating, 1998; Keating and Meinke, 1998; Snow et al., 2000), climate forecast applications (Meinke et al., 1996), long-term wheat cropping systems (Probert et al., 1995), nutrient leaching to groundwater (Verburg et al., 1996; Keating et al., 1997), and cropping potential in new regions (Carberry et al., 1996). Because multiple biological modules can be plugged into it, APSIM has been able to be deployed in studies of intercropping (Carberry et al., 1996) and weed competition (Keating et al., 1999). APSIM's MANAGER module is very flexible and has been used (by non-model developers) to simulate a situation in which water is captured from a variety of sources into a farm dam and this dam is then available for irrigation of a sugarcane crops (S. Lisson et al., unpublished). The modularity built into APSIM has also facilitated comparisons of alternative modelling approaches. Examples include a comparative study of 'tipping bucket' or Richard's Equation approaches to modelling water and solute movement (Verburg, 1996).

6. Comparisons

Although each modelling group has attained some level of modularity in their procedures for developing and maintaining crop models, we found that there were many differences in how modularity was interpreted and implemented. These differences prevent modules from one group being used by other groups without sometimes considerable amounts of additional programming. The most common features of the approaches are that models from each group are written in the Fortran programming language, they all use daily time steps and, in general, they consist of components developed along scientific discipline lines. Examples of this latter feature can be seen by comparing environmental and management modules in APSIM with those in DSSAT models (Tables 1 and 2). Indeed, there has been considerable sharing of code among these groups in the past. For example, the soil water component developed by Ritchie (1998) for the CERES models was the most important influence on the early development of APSIM's SOILWAT module, and the soil nitrogen component in the DSSAT models (Godwin and Singh, 1998) originally was derived from the PAPRAN model published by Seligman and Van Keulen (1981) from the 'School of De Wit'. Another similarity found in the three groups is that each has models of the major food crops, and the crop components are usually separate from the environmental and other components.

These levels of compatibility of components among groups, however, are not sufficient if more widespread co-operation in model development and use is to occur in the future. There are still too many differences to provide easy ways to compare models or their components under different situations and conditions. Differences in methods for reading parameters, initial conditions, and environmental data, for exchanging information among model components, and for computing rates and updating state variables along with differences in state variables that describe a component and their units are barriers to more widespread adoption of any one of these approaches for modular model development. In addition, order of computations and signalling of events may differ among models from different groups.

Models from each group also still use different data formats and files, even though progress has been made on 'standard' files and formats for data exchange or for direct use by models (Hunt et al., 2001). It is highly unlikely that a single solution will be found that is acceptable to all who are developing crop and other agricultural models. However, modularity approaches for modelling already being used in individual groups clearly demonstrate advantages, and below we suggest some key elements to modular model development.

7. Key elements of modular model structure

The following modular structure guidelines are proposed based on the approach of Van Kraalingen (1995) and the experience of the APSIM development team (McCown et al., 1996) Each module should:

1. encapsulate all data and functionality for a well defined simulation component, with abstraction usually along scientific discipline lines;
2. enforce 'data hiding' for all state variables within the simulation component (e.g. modules own and protect their state variables);
3. have its own well defined interface for input and output;
4. initialize itself;
5. compute rates of change for its state variables;
6. integrate its state variables; and
7. be capable of being run independently or in various combinations with other modules (e.g. the 'plug-in, pull-out' capability).

We believe following these guidelines would result in better structured simulation models that would deliver some of the key benefits of modularity, namely greater flexibility, enhanced collaborative opportunities, greater capability to simulate more comprehensive systems and reduced maintenance costs. We recognize that these guidelines can be met in different ways with different programming approaches and different levels of investment in software engineering. We believe that whatever the software implementation approach chosen, the ability for biophysical scientists to specify, understand, interact with and test the code should be retained. This requirement leads to one final guideline, namely:

8. Modular approaches should maintain the involvement of biophysical researchers in the scientific aspects of module design and construction, and spare these researchers the need to get involved in the non-science aspects of software construction.

8. An example

The criteria for generic modular structure developed by Reynolds and Acock (1997) and the specific guidelines for construction of a module were used in the development of the modules for the simple crop model which is used as our example. For larger, more complex models, it will be difficult to decide on how many modules to create. A few relatively large modules, using the criteria and principles above, would allow separation of these major components so that each could easily be replaced by other modules similarly structured, with exact matching of input and output variables. However, some of the modules may be components of larger system models or they may be made up of smaller modules, each of which could have the same criteria applied. Thus, the determination of modules is, to some extent, arbitrary.

Fig. 3 illustrates the modular format used in a model, in which each module has two or more of the following five components:

1. initialization
2. rate calculations
3. integration
4. output
5. final calculations.

The main program (MAIN.FOR) contains up to five calls to each module to accomplish the various components of processing. The dynamic flow of processing within the program is regulated with the DYNAMIC variable. Each module is called once at the beginning of simulation with DYNAMIC set equal to 'INITIAL', resulting in execution of the initialization portion of the module. During the daily time loop, each module is called three times: once each for rate calculation (DYNAMIC = 'RATE'), integration calculations (DYNAMIC = 'INTEG'), and for output of daily computed data (DYNAMIC = 'OUTPUT'). A final call to each module is made to write summary output files and to close input and output files (DYNAMIC = 'CLOSE') after the simulation is complete.

The FORTRAN code used for directing calls to a module from the main program is presented in Fig. 4. Fig. 5 lists typical code used to control processing within a module. The following sections describe the five divisions of modules:

1. *Initialization*: at the beginning of each simulation, the 'initialization' section is used to read in data from input data files and to initialize all variables in each module.
2. *Rate calculations*: rate calculations are performed at the beginning of the daily time step loop. This ensures that rates of change of state variables for a given day of simulation are all based on values of these state variables for a common point in time, i.e. at the end of the previous day.
3. *Integration calculations*: the integration portion of the model updates state variables throughout the model for each day of simulation using the values calculated in the rate calculations portions of the module.
4. *Output*: in the 'Output' section of the modules, daily computed data is written to output files.
5. *Final*: The 'Final' section of processing is used to write end of simulations summary output files and to close output files.

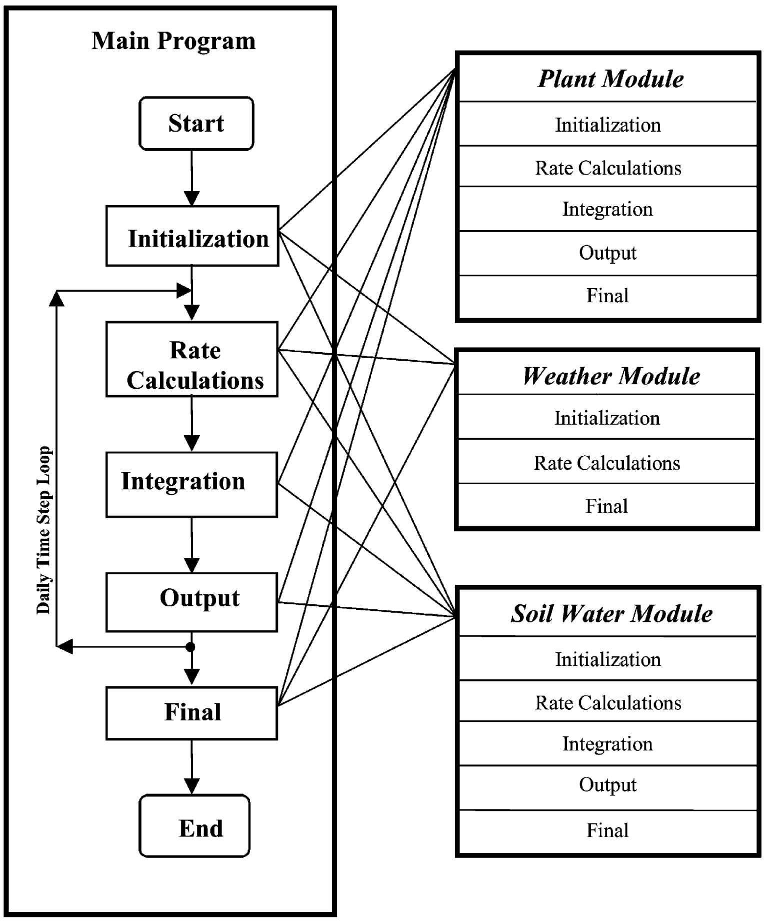

Fig. 3. Modular structure derived from the approach used in Fortran simulation environment (Van Kraalingen, 1995).

This structure leads to modules that completely represent a component, including all of its state variables and parameters. Any communication between the module and the rest of the model is through inputs and outputs explicitly defined. Thus, the module must provide the correct list of outputs and be able to function with the inputs passed from the rest of the model. This feature is a critical one. If one wants to plug in an alternate version of a module, such as a soil water module, it may have a different set of parameters than the one it is being compared with. By isolating these concept-dependent parameters in the module, the other parts of the model do not have to be changed when a new module is coupled. This structure has been shown to work well for many modules. However, it does not provide guidelines for all situations that crop modellers are likely to face. For example, if an iterative solution to rate calculations is necessary and it involves more than one module, there is no provision in the above structure to ensure compatibility across models.

```
!====================================================================
       PROGRAM DRIVER
!====================================================================

!====================================================================
!      Initialization Section
!====================================================================
       .
       CALL WEATHER(arg1, arg2, . . . . , 'INITIAL')
       .
       CALL PLANT  (arg1, arg2, . . . . , 'INITIAL')
       .
       CALL SOILWAT(arg1, arg2, . . . . , 'INITIAL')
       .
!====================================================================
!      Begin Daily Loop
!====================================================================
!      Rate Calculation Section
!====================================================================
       .
       CALL WEATHER(arg1, arg2, . . . . , 'RATE')
       .
       CALL PLANT  (arg1, arg2, . . . . , 'RATE')
       .
       CALL SOILWAT(arg1, arg2, . . . . , 'RATE')
       .
!====================================================================
!      Integration Section
!====================================================================
       .
       CALL PLANT  (arg1, arg2, . . . . , 'INTEG')
       .
       CALL SOILWAT(arg1, arg2, . . . . , 'INTEG')
       .
!====================================================================
!      Output Section
!====================================================================
       .
       CALL PLANT  (arg1, arg2, . . . . , 'OUTPUT')
       .
       CALL SOILWAT(arg1, arg2, . . . . , 'OUTPUT')
       .
!====================================================================
!      End Daily Loop
!====================================================================
!      FINAL Section
!====================================================================
.
       CALL WEATHER(arg1, arg2, . . . . , 'FINAL')
       .
       CALL PLANT  (arg1, arg2, . . . . , 'FINAL')
       .
       CALL SOILWAT(arg1, arg2, . . . . , 'FINAL')
       .
!====================================================================
!      End of Program
!====================================================================
       END DRIVER
```

Fig. 4. Module processing within main program, showing how each module is called.

```
!=====================================================================
       SUBROUTINE PLANT(arg1, arg2, arg3, . . ., DYNAMIC)
!=====================================================================
       CHARACTER*10 DYNAMIC

!=====================================================================
!      Initialization Section
!=====================================================================
       IF (INDEX(DYNAMIC,'INITIAL') .NE. 0) THEN
       <Open files>
       <Read input data>
       <Initialize variables>
       <Perform once-only calculations>
       .
       .
!=====================================================================
!      Rate Calculation Section
!=====================================================================
       ELSEIF (INDEX(DYNAMIC,'RATE') .NE. 0) THEN
       <Calculate daily rates>
       .
       .
!=====================================================================
!      Integration Section
!=====================================================================
       ELSEIF (INDEX(DYNAMIC,'INTEG') .NE. 0) THEN
       <Update state variables>
       .
       .
!=====================================================================
!      Output Section
!=====================================================================
       ELSEIF (INDEX(DYNAMIC,'OUTPUT') .NE. 0) THEN
       <Write daily output>
       .
       .
!=====================================================================
!      Final Section
!=====================================================================
       ELSEIF (INDEX(DYNAMIC,'FINAL') .NE. 0) THEN
       <Write summary output>
       <Close files>
       .
       .
!=====================================================================
!      End of Module
!=====================================================================
       ENDIF
!=====================================================================
       RETURN
       END SUBROUTINE PLANT
!=====================================================================
```

Fig. 5. Sample Fortran code for plant module structure (similar structure for soil water and weather modules).

Porter et al. (1999, http://icasanet.org) provide complete documentation, source code, example data, and an executable file for the simple plant–soil water model summarized in this paper. This example is programmed in Fortran 90, however, programming language is no longer a critical issue for dynamic models. This approach has been successfully used by students in an advanced simulation class using Fortran, Basic, Pascal, and C programming languages. As long as the guidelines are followed and the main program can control access to modules as outlined earlier, individual modules can even be in different programming languages. This has been demonstrated also in APSIM which achieves this via use of dynamic linked libraries (dlls) that are supported by many current programming languages. This is another major feature of the recommended modular approach.

One question that remains, however, is how compatible is this recommended approach with the other modular approaches used by the three major model development groups currently in ICASA. First of all, it is highly compatible with the approach used for most of the 'School of De Wit' crop models because they use FSE, which is the basic structure being proposed. The major extension recommended here is that neither the programming language nor programming environment is a requirement. Original DSSAT models were not programmed using the guidelines listed earlier. Thus, even though they had similar subroutine structures, the programming style intermingled parameters from different modules and did not follow a strict sequence of calculations. This is the main reason that considerable effort has already gone into revising the DSSAT models to adhere to these guidelines. The APSIM models largely adhere to these guidelines, but APSIM has a much more elaborate communication system than that outlined in the simple 'driver' program in Fig. 4. APSIM has much 'looser' coupling between modules based on a customized messaging system that manages data-flow, event signalling and action communications amongst modules. This communications infrastructure has required a significant software engineering investment (for development and evolutionary maintenance) but, on the positive side, it has delivered considerable flexibility in the applications to which APSIM could be deployed. For example, APSIM modules can be instantiated, meaning that multiple 'instances' of a module can be linked into the simulation. This feature opens up a new frontier of simulation possibilities, such as multiple paddocks and complex species mixtures/arrangements. While there has been a large software engineering investment in APSIM, the scientific leadership has been maintained in the hands of biophysical scientists, with the software engineers taking leadership on software infrastructure matters.

We have not yet attempted to integrate modules from each of these two approaches into models using the other drivers and variable communication systems. We believe this would be possible now, but would require significant programming effort to interface respective modules to current simulation drivers or communications systems.

9. Prospects for the future

A number of researchers have used object-oriented programming languages to develop crop models with supposedly more modular structures (e.g. Van Evert and Campbell, 1994; Acock and Reddy, 1997; Caldwell and Fernandez, 1998; Pan et al., 2000). An object is a programming unit that groups together a data structure and the operations that can use or affect that data. Objects are the principal building blocks of object-oriented programs. However, these new programming techniques have not been widely adopted by the crop modelling community. Until recently, there was no leading object oriented language or approach for modelling. Although the different object-oriented languages shared a set of commonly accepted concepts, the differences helped to fragment the object-oriented industry and discouraged new users from learning these concepts (Jacobson et al., 1998). Recent developments in software engineering tools provide new opportunities for developing models of agricultural systems. These recent developments provide a unified approach for software development that is now widely adopted and used throughout the software industry. One of the most promising developments is the Unified Modelling Language (UML), a language for specifying, constructing, and documenting software systems (Jacobson et al., 1998; Quatrani and Booch, 1999; http://www.platinum.com/corp/uml/summ_11.pdf). The UML was developed by 18 companies to lower the cost of software development, maintenance, implementation and training associated with large software projects. Some of the major goals of developing UML were to (http://www.platinum.com/corp/uml/summ_11.pdf): (1) provide users a ready-to-use, expressive visual modelling language; (2) be independent of particular programming languages; (3) provide a formal basis for understanding the modelling language; (4) encourage the growth of object-oriented tools market; and (5) support high level development concepts such as collaborations and frameworks. In essence, the purpose of UML is to visualize, specify, construct and document object-oriented models.

In UML, the development of a model starts by the creation of a conceptual model. This conceptual model is a graphical representation of the system showing its components, their linkages, and the information passed among them. Once the conceptual model of the system is complete, the specific procedures or functions for each object or class can be programmed in a variety of programming languages such as JAVA, C++ or VISUAL BASIC. UML allows for both forward engineering (conversion of UML conceptual model into code) and reverse engineering (conversion of code into UML conceptual model). Apart from the executables, UML produces documentation that includes requirements, architecture, design, source code, project plans, tests, prototypes and releases.

UML is currently being explored as a mechanism for cooperation among different ICASA groups for future model development. Papajorgji et al. (2000) used the Rational Rose UML software (http://www.rational.com/products/rose/index.jtmpl) to implement the modular model example described above (Porter et al., 1999). This approach was used to serve as an example for those who have until now developed models in procedural languages like FORTRAN. We also wanted to learn how we might promote a transition toward more modular agricultural systems models using these new tools. The Java language (Jbuilder; http://www.borland.com/jbuilder) was used for implementation. In parallel with the modular FORTRAN version (Porter et al., 1999), the resulting Java model had three main modules or objects (plant, soil, and weather), which were controlled by a Simulate method. In the resulting object-oriented model, we also used the same sequence of computations (initialisation, rate calculations, and integration) as in the example presented by Porter et al. (1999).

Papajorgji et al. (2000) concluded that UML is an excellent tool for understanding agricultural systems and creating object oriented models. There appear to be many advantages of using these new industry standard tools for modelling agricultural systems relative to the modularity criteria listed earlier. However, these tools will not solve all of the problems by themselves. For example, in order to plug a module into an existing model with little impact to the other program components (criteria No.1 listed earlier), the module itself must provide the same functionality as the one it is to replace. Even though the module itself may be more or less detailed and even have different parameters and state variables, it must be conceptually consistent with the module it replaces and it must communicate the information (messages) required by other modules in the model. This means that the agricultural system modelling community must define these essential components and the messages that they pass for different types of system models. There will be a need for more than one conceptual model for a given system to account for simple vs. complex models and for different time steps used in simulating the systems.

We anticipate a continuing evolution in modelling tools. The UML could be adopted by various modelling groups for enhancing communication and increasing compatibility of models and modules. However, large investments have been made in existing models, which are still widely used. Thus, those who anticipate an immediate conversion of existing models into this paradigm will require some patience. It is clear that investments in models are necessary for their continued use. These investments will need to be both to improve the science embodied in the models as well as the software used to implement them. A continuing dialogue is needed to help the agricultural modelling community achieve a higher level of compatibility and more efficient procedures for making our knowledge available to users through the use of agricultural systems models. While much more effort is needed before the exchange of modules becomes a seamless process, there is no doubt that attention to the criteria and guidelines for module construction outlined in this paper is an essential first step to improving the utility of future collaborative modelling efforts.

References

Acock, B., Reddy, V.R., 1997. Designing an object-oriented structure for crop models. Ecol. Modelling 94, 33–44.

Acock, B., Reynolds, J.F., 1989. The rationale for adopting a modular generic structure for crop simulators. Acta Horticulturae 248, 391–396.

Asseng, S., Fillery, I.R.P., Anderson, G.C., Dolling, P.J., Dunin, F.X., Keating, B.A., 1998. Use of the APSIM wheat model to predict yield, drainage and NO_3 leaching in a deep sand. Australian Journal Agricultural Research 49, 363–377.

Bouman, B.A.M., Van Keulen, H., Van Laar, H.H., Rabbinge, R., 1996. The 'School of de Wit' crop growth simulation models: a pedigree and historical overview. Agric. Systems 52, 171–198.

Boote, K.J., Jones, J.W., Hoogenboom, G., Pickering, N.B., 1998. The CROPGRO model for grain legumes. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 99–128.

Caldwell, R.M., Fernandez, A.A.J., 1998. A generic model for hierarchy for systems analysis and simulation. Agric. Systems 57, 197–225.

Carberry, P.S., Adiku, S.G.K., McCown, R.L., Keating, B.A., 1996. Application of the APSIM cropping systems model to intercropping systems. In: Ito, O., Johansen, C., Adu-Gyamfi, J.J., Katayama, K., Kumar Rao, J.V.D.K., Rego, T.J. (Eds.), Roots and Nitrogen in Cropping Systems of the Semi-arid Tropics. Japan Agricultural Research Centre for Agricultural Sciences, Ibaraki, Japan, pp. 637–648.

De Wit, C.T., 1965. Photosynthesis of Leaf Canopies (Agricultural Research Report 663). Pudoc, Wageningen, The Netherlands.

De Wit, C.T., Brouwer, R., Penning de Vries, F.W.T., 1970. The simulation of photosynthetic systems. In: Setlik, I. (Ed.), Prediction and Measurement of Photosynthetic Productivity. Proceedings IBP/PP Technical Meeting Trebon 1969. Pudoc, Wageningen, The Netherlands, pp. 47–50.

De Wit, C.T., Goudriaan, J., Van Laar, H.H., Penning de Vries, F.W.T., Rabbinge, R., Van Keulen, H., Louwerse, W., Sibma, L., De Jonge, C., 1978. Simulation of Assimilation, Respiration and Translocation of Crops. Simulation Monographs, Pudoc, Wageningen, The Netherlands.

Godwin, D.C., Singh, U., 1998. Nitrogen balance and crop response to nitrogen in upland and lowland cropping systems. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 55–77.

Goudriaan, J., Van Laar, H.H., 1994. Modeling Potential Crop Production Processes. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Hunt, L.A., Boote, K.J., 1998. Data for model operation, calibration, and evaluation. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 9–39.

Hunt, L.A., White, J.W., Hoogenboom, G., 2001. Agronomic data: advances in documentation and protocols for exchange and use. Agric. Systems 70 (2–3), 477–492.

IBM, 1975. Continuous System Modeling Program III (CSMP III), Program Reference Manual. IBM SH19-7001-3. Techn. Publ. Dept., White Plains, USA.

Jacobson, I., Booch, G., Rumbaugh, J., 1998. The Unified Software Development Process. Addison-Wesley, Reading, MA, USA.

Jones, C.A., Kiniry, J.R. (Eds.), 1986. CERES-Maize: A Simulation Model of Maize Growth and Development. Texas A&M University Press, College Station, USA.

Jones, J.W., Tsuji, G.Y., Hoogenboom, G., Hunt, L.A., Thornton, P.K., Wilkens, P.W., Imamura, D.T., Bowen, W.T., Singh, U., 1998. Decision support system for agrotechnology transfer. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 157–177.

Keating, B.A., Meinke, H., 1998. Assessing exceptional drought with a cropping systems simulator: a case study for grain production in north-east Australia. Agric. Systems 57, 315–332.

Keating, B.A., Verburg, K., Huth, N.I., Robertson, M.J., 1997. Nitrogen management in intensive agriculture: sugarcane in Australia. In: Keating, B.A., Wilson, J.R. (Eds.), Intensive Sugarcane Production: Meeting the Challenges Beyond 2000. CAB International, Wallingford, UK, pp. 221–242.

Keating, B.A., Carberry, P.S., Robertson, M.J., 1999. Simulating N fertiliser response in low-input farming systems 2. Effects of weed competition. In: Donatelli, M., Stockle, C., Villalobos, F., Villar Mir, J.M. (Eds.), Proceedings of the International Symposium on Modelling Cropping Systems. European Society for Agronomy, Lleida, Spain, pp. 205–206.

Kropff, M.J., Van Laar, H.H. (Eds.), 1993. Modelling Crop–Weed Interactions. CAB International, Wallingford, UK.

Kropff, M.J., Van Laar, H.H., Matthews, R.B., 1994. ORYZA1: An Ecophysiological Model for Irrigated Rice Production. SARP Research Proceedings. IRRI, AB-DLO, Wageningen, The Netherlands, Philippines.

Littleboy, M., Silburn, D.M., Freebairn, D.M., Woodruff, D.R., Hammer, G.L., 1989. PERFECT: A Computer Simulation Model of Productivity, Erosion, Runoff Functions to Evaluate Conservation Techniques. Queensland Department of Primary Industries, Bulletin QB89005, Australia.

McCown, R.L., Williams, J., 1989. AUSIM: a cropping systems model for operational research. In: Proceedings of Eighth Biennial Conference on Modelling and Simulation, Australian National University, 25–27 September, 1989, Canberra. Simulation Society of Australia Inc. affiliated with International Association for Mathematics and Computers in Simulation, pp. 54–59.

McCown, R.L., Hammer, G.L., Hargreaves, J.N.G., Holzworth, D.P., Freebairn, D.M., 1996. APSIM: a novel software system for model development, model testing, and simulation in agricultural systems research. Agric. Systems 50, 255–271.

Meinke, H., Stone, R.C., Hammer, G.L., 1996. Using SOI phases to forecast climatic risk to peanut production: a case study for northern Australia. Int. J. Climatol 16, 783–789.

Muchow, R.C., Keating, B.A., 1998. Assessing irrigation requirements in the Ord Sugar Industry using a simulation modelling approach. Australian Journal Experimental Agriculture 38, 345–354.

Papajorgji, P., Braga, R., Jones, J.W., Porter, C.H., 2000. Modular Crop Model Using Unified Modeling Language (Research Report No. 2000-1101). Agricultural and Biological Engineering Department, University of Florida, Gainesville, FL. http://www.icasanet.org (in preparation).

Pan, X., Hesketh, J.D., Huck, M.G., 2000. OWSimu: an object-oriented and web-based simulator for plant growth. Agric. Systems 63, 33–47.

Penning de Vries, F.W.T., Van Laar, H.H., 1982. Simulation of growth processes and the model BACROS. In: Penning de Vries, F.W.T., Van Laar, H.H. (Eds.), Simulation of Plant Growth and Crop Production. Simulation Monographs. Pudoc, Wageningen, The Netherlands, pp. 114–136.

Penning de Vries, F.W.T., Jansen, D.M., Ten Berge, H.F.M., Bakema, A., 1989. Simulation of Ecophysiological Processes of Growth in Several Annual Crops. Simulation Monographs. Pudoc, Wageningen, The Netherlands.

Porter, C.H., Braga, R., Jones, J.W., 1999. Research Report No. 99-0701. Agricultural and Biological Engineering Department, University of Florida, Gainesville, FL. http://www.icasanet.org/.

Probert, M.E., Keating, B.A., Thompson, J.P., Parton, W.J., 1995. Modelling water, nitrogen and crop yield for a long-term fallow management experiment. Australian Journal Experimental Agriculture 35, 941–950.

Quatrani, T., Booch, G., 1999. Visual Modeling with Rational Rose 2000 and UML. Addison-Wesley, Reading, MA, USA.

Rappoldt, C., Van Kraalingen, D.W.G., 1996. The Fortran Simulation Translator: FST Version 2.0. Introduction and Reference Manual. Quantitative Approaches in Systems Analysis No. 5. C.T. de Wit Graduate School for Production Ecology, Wageningen University, The Netherlands.

Reynolds, J.F., Acock, B., 1997. Modularity and genericness in plant and ecosystem models. Ecological Modelling 94, 7–16.

Ritchie, J.T., 1998. Soil water balance and plant water stress. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 41–54.

Ritchie, J.T., Singh, U., Godwin, D.C., Bowen, W.T., 1998. Cereal growth, development and yield. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding options for agricultural production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 79–98.

Rosenthal, W.D., Vanderlip, R.L., Jackson, B.S., Arkin, G.F., 1989. SORKAM: A Grain Sorghum Crop Growth Model. Texas Agricultural Experiment Station Computer Software Documentation Series. Texas A&M University, College Station, TX, USA.

Seligman, N.G., Van Keulen, H., 1981. PAPRAN: A simulation model of annual pasture production limited by rainfall and nitrogen. In: Frissel, M.J., Van Veen, J.A. (Eds.), Simulation of Nitrogen Behaviour of Soil-plant Systems. Pudoc, Wageningen, The Netherlands, pp. 192–220.

Snow, V.O., Smith, C.J., Polglase, P.J., Probert, M.E., 2001. Modelling nitrogen dynamics in an eucalypt plantation irrigated with sewage effluent or bore water. Aust. J. Soil Science (in press).

Spitters, C.J.T., Van Keulen, H., Van Kraalingen, D.W.G., 1989. A simple and universal crop growth simulator: SUCROS87. In: Rabbinge, R., Ward, S.A., Van Laar, H.H. (Eds.), Simulation and Systems Management in Crop Protection. Simulation Monographs. Pudoc, Wageningen, The Netherlands, pp. 147–181.

Ten Berge, H.F.M., 1993. Building capacity for systems research at national agricultural research centers: SARP's experience. In: Penning de Vries, F.W.T., Teng, P.S., Metselaar, K. (Eds.), Systems Approaches for Agricultural Development. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 515–538.

Tsuji, G.Y., Uehara, G., Balas, S., 1994. DSSAT v3: Vols. 1, 2, and 3. IBSNAT Project. Department of Agronomy and Soil Science, University of Hawaii, Honolulu, Hawaii.

Uehara, G., Tsuji, G.Y., 1998. Overview of IBSNAT. In: Tsuji, G.Y., Hoogenboom, G., Thornton, P.K. (Eds.), Understanding Options for Agricultural Production. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 1–7.

Van Evert, F.K., Campbell, G.S., 1994. CropSyst: a collection of object-oriented simulation models of agricultural systems. Agron. J. 86, 325–331.

Van Keulen, H., Penning de Vries, F.W.T., Drees, E.M., 1982. A summary model for crop growth. In: Penning de Vries, F.W.T., Van Laar, H.H. (Eds.), Simulation of Plant Growth and Crop Production. Pudoc, Wageningen, The Netherlands, pp. 87–97.

Van Kraalingen, D.W.G., 1991. The FSE System for Crop Simulation (Simulation Report CABO-TT No. 23). Centre for Agrobiological Research and Dept of Theoretical Production Ecology, Wageningen, The Netherlands.

Van Kraalingen, D.W.G., 1995. The FSE System for Crop Simulation: Version 2.1 (Quantitative Approaches in Systems Analysis Report No. 1). C.T. de Wit Graduate School for Production Ecology, Wageningen University, Wageningen, The Netherlands.

Van Kraalingen, D.W.G., Stol, W., 1997. Evapotranspiration Modules for Crop Growth Simulation. Implementation of the Algorithms from Penman, Makkink and Priestley-Taylor (Quantitative Approaches in Systems Analysis Report No. 11). C.T. de Wit Graduate School for Production Ecology, Wageningen University, Wageningen, The Netherlands.

Van Laar, H.H., Goudriaan, J., Van Keulen, H. (Eds.), 1992. Simulation of Crop Growth for Potential and Water-limited Production Situations (as applied to spring wheat; Simulation Report CABO-TT, 27). CABO-DLO, Wageningen, The Netherlands.

Verburg, K., 1996. Methodology in Soil Water and Solute Balance Modelling: An evaluation of the APSIM-SoilWat and SWIMv2 models (Divisional Report No. 131). CSIRO Division of Soils, CSIRO Australia (ISBN 0 643 05868 0).

Verburg, K., Keating, B.A., Bristow, K.L., Huth, N.I., Ross, P.J., Catchpoole, V.R., 1996. Evaluation of nitrogen fertiliser management strategies in sugarcane using APSIM-SWIM. In: Wilson, J.R., Hogarth, D.M., Campbell, J.A., Garside, A.L. (Eds.), Sugarcane: Research Towards Efficient and Sustainable Production. CSIRO Division of Tropical Crops and Pastures, Brisbane, Australia, pp. 200–202.

Williams, J.R., 1983. EPIC, The Erosion-Productivity Impact Calculator, Vol. 1. Model Documentation. Agricultural Research Service, United States Department of Agriculture, Beltsville, MD.