

A Design for a Modular, Generic Soil Simulator to Interface with Plant Models

Dennis Timlin,* Yakov A. Pachepsky, and Basil Acock

ABSTRACT

The various linkages in agricultural systems are so complex that research questions are often studied via simulation modeling. The development and upgrading of detailed models requires a large investment of time and resources; therefore, it is necessary to simplify the process of building the model and incorporating modifications as research finds new information to include. The goal of this study was to develop the framework for a generic soil simulator that could easily be modified and incorporated into crop models. Soil and root processes are represented by modules that interact on the spatial-temporal grid covering the soil profile and the simulated time interval. Data were divided into public and private components, to minimize information passing between modules. This modular structure and information hiding simplifies replacement or addition of modules and promotes code reuse. The classes of modules include (i) control modules that oversee interactions between processes, (ii) water, solute, heat, and gas transport modules, (iii) inter-phase chemical transformation modules, (iv) biochemical transformation modules, and (v) root growth and uptake modules. A representative simulator, 2DSOIL, was assembled according to the proposed design. Examples include the incorporation into a simple crop model and the expansion of 2DSOIL with a management module to simulate chemical application.

COMPUTER SIMULATION MODELING has become an important tool for researchers to study agricultural systems, where the linkages of components can be quite complex. Computer simulation models used in agricultural research have typically been designed for a particular purpose: e.g., to predict chemical transport, LEACHM (Hutson and Wagenet, 1992), and crop development and yield, GLYCIM (Acock and Trent, 1991), or to simulate management effects on water quality, RZWQM (Ahuja et al., 1991). Models that contain comprehensive soil components have generally concentrated on environmental issues such as water quality, while models with comprehensive plant components have been concerned with crop yield and development. In these and other similar models, the level of detail is typically high in either the plant or soil component, but not in both. There are also models with much detail in the atmospheric component but less in the soil and plant components. Currently, agricultural managers and researchers are increasingly concerned with both crop yields and the environmental impacts of agriculture. Indeed, the two are closely linked and strongly interact.

There are a number of water and solute transport models that use comprehensive descriptions of mass and energy transport in soils. They use finite element (Benjamin et al., 1990; Šimůnek et al., 1992) and finite difference (Ahuja et al., 1991; Hutson and Wagenet, 1992) approximations of the governing equations. A salient property of two-dimensional finite element models is flexibility in incor-

poration of management practices. Uneven soil surface, local changes of soil properties and composition (due to tillage, fertilizer or plant residue placement), and internal boundaries (drainage) and other management-induced features can be readily handled. These models, however, are limited to soil and water transport processes and include only simple plant processes. Overall, these powerful soil simulators have been developed independently of the plant simulators. As a result, plant and soil modelers have not been able to take advantage of each other's work.

It is not practical to develop ad hoc models for each system, nor is it possible to build a single, all-purpose model for all applications. Reynolds et al. (1992) suggested that the goal should be to build a suite of models based on general principles derived from structures and behaviors that are fundamentally similar across plants and ecosystems. To accomplish this, Reynolds et al. (1992) proposed the development of generic models that have a similar structure and contain similar modules, some of which may be used in several different models with appropriate parameter changes. In a generic modular structure, related processes are grouped into submodel components, and the function of each module and the variables to be calculated for output are explicitly defined. A modular structure would also simplify the process of updating or modifying model components as new knowledge or data become available.

Modular design has received much attention from software engineers (Einbu, 1989; Witt et al., 1994). The importance and usefulness of modularity come only partially from the ability to decompose a system into manageable parts. The most important advantage is that modules can be constructed to hide information from programmers of other modules (Einbu, 1989). Information is hidden by grouping both the data and operations that use the data into a single module. The importance of a modular structure and information hiding has been recognized in the agricultural modeling community. Hodges et al. (1992) described a modular structure for the IBSNAT models that enhances the ability to modify the model for new applications or improvements. A modular structure can be developed following accepted guidelines for modular software design or object-oriented programming (OOP) paradigms (Thomas, 1989; Wirfs-Brock et al., 1990). It is not dependent on programming language, though some restrictions are applied.

A potentially useful type of modular generic model would be a plant-soil simulator that could incorporate computer code from the best plant and soil models available today. The advantages include the ability to reuse code developed by other modelers and to allow a modeler to focus on her or his own expertise.

For this study, we developed the design of a modular,

D. Timlin and B. Acock, USDA-ARS, Systems Res. Lab., Bldg. 007, Rm. 008, BARC-West, Beltsville, MD 20705; Y.A. Pachepsky, Dep. of Botany, Duke Univ., Durham, NC 27708. Received 15 Aug. 1994. *Corresponding author (Email: dtimlin@asrr.arsusda.gov).

generic soil simulator that (i) allows the easy addition of soil, plant, or atmospheric processes of varying complexity, (ii) provides a means of expanding and replacing sub-models of processes, and (iii) simplifies the addition of management modules. The model is generic, because the computer code to simulate soil processes is not a permanent part of the model; any other code can be substituted if the code follows the precepts outlined in this paper. We show how we can take advantage of the recent developments in software engineering to achieve this goal. Finally, we demonstrate an application of this methodology in the form of a model, 2DSOIL.

CONCEPTUAL DEVELOPMENT

Modular Design

The purpose of decomposing a system into modules is to be able to hide code and data from other modules in the system (Einbu, 1989). The goal is to separate the module user from the developer (Thomas, 1989). Modules are designed to enhance reuse, maintainability, and reliability of the code. This goes beyond previous philosophy where modularization was influenced by the "everything is a hierarchy" view of programming (Kirk, 1990). Modules should have loose interunit coupling and high internal cohesion (Witt et al., 1994). Loose coupling is characterized by independence, a simple interface between modules, a limited number of interfaces, and a minimum of information passing. High internal cohesion is characterized by interdependent elements packaged together and by information hiding (Blum, 1992; Witt et al., 1994).

The design requirements for modules as outlined above do not require a special programming language (Blum, 1992), although they are more easily implemented using an OOP language. In OOP, an object (module) contains specific information (data) and is coded to perform certain operations (functions). Our design, coded in FORTRAN, follows OOP precepts, but within the constraints of FORTRAN. Loose coupling is implemented by having each module input and manage its own data. By doing this, the functions and data are kept together in the object (module). High cohesion is achieved by developing distinct, independent modules, each for a specific soil or root process.

Soil and root processes belong to one of the following groups:

1. Transport processes (e.g., water movement, heat movement, nitrate movement, oxygen movement).
2. Root processes (e.g., water uptake, NH_4 uptake, root respiration).
3. Interphase exchange processes (e.g., reversible Ca-Mg-Na exchange).
4. Biotransformation processes (e.g., denitrification, CO_2 production).
5. Management processes (e.g., tillage, ammonium phosphate application).

The classes of soil and root process modules based on this grouping are shown in Fig. 1. Note that every class is on the same level; there is no hierarchy among the classes. Each class may include several process modules. For example, transport processes can be water movement, solute movement, heat movement, and gas movement; root processes can be water or nutrient uptake, root growth, root respiration, and the like. The design assumes that only the process modules needed in a particular application have to be included in the simulator. Each process module may contain one or more submodules that, for example, calculate coefficients such as hydraulic conductivity for the equations.

The general structure that allows the modules to be loosely

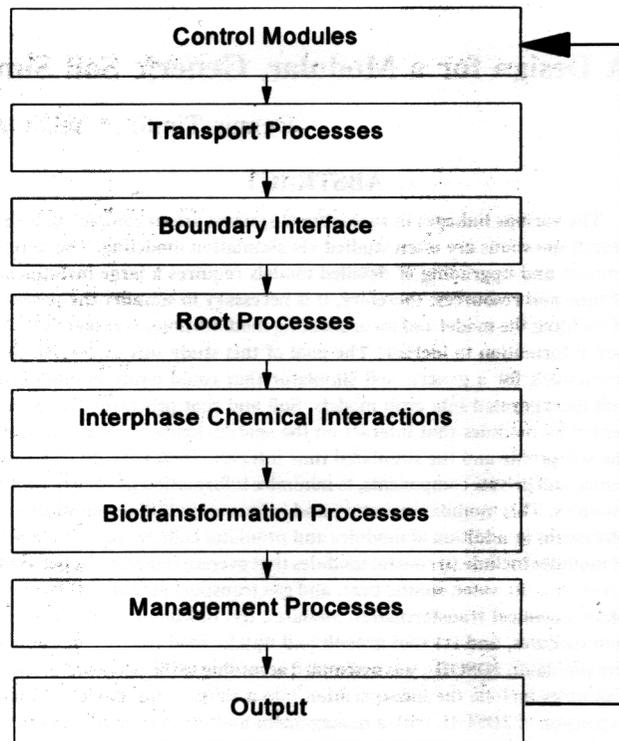


Fig. 1. The classes of modules used in the generic simulator and the sequence in which they are called.

coupled constitutes the framework of the generic simulator. This framework is supported by control modules. If a process module (such as a plant model) is structured in the manner described in this paper, the process module can be added to the simulator. Control modules will oversee the new module's functioning and its interaction with other modules. This structure is described next.

Module Structure and Interaction

Modules interact by passing or sharing data. To share data among process modules, the data must be represented consistently in all modules. To provide a loose coupling of modules, the amount of data accessible to all modules must be minimized.

The minimum data set available to all modules, termed *global*, has to be independent of the model or algorithms used to represent the processes. These data must also be sufficient to describe the state of the system at any particular time. Soil and root processes in the soil-plant-atmosphere system are characterized by the volumetric contents of substances (e.g., water content, bulk density, oxygen concentration, root length density, etc.). Potentials of physical fields and related physical values are also used (e.g., matric potential, temperature, etc.). These values are state variables of the soil-root system. The state variables are subject to changes caused by fluxes of energy and matter into or out of the system. Internal point fluxes are known as sources or sinks, depending on their direction. Distributed fluxes through boundaries of the soil profile are referred to as boundary fluxes. Examination of soil process models shows that calculations of changes in one soil state variable may require the values of several other soil state variables. For instance, temperature and soil water content are needed to calculate changes in ammonium concentration caused by nitrification. Similarly, several processes may contribute to changes in concentration of a particular substance. For example, both root respiration and decomposition of organic materials contribute to CO_2 pro-

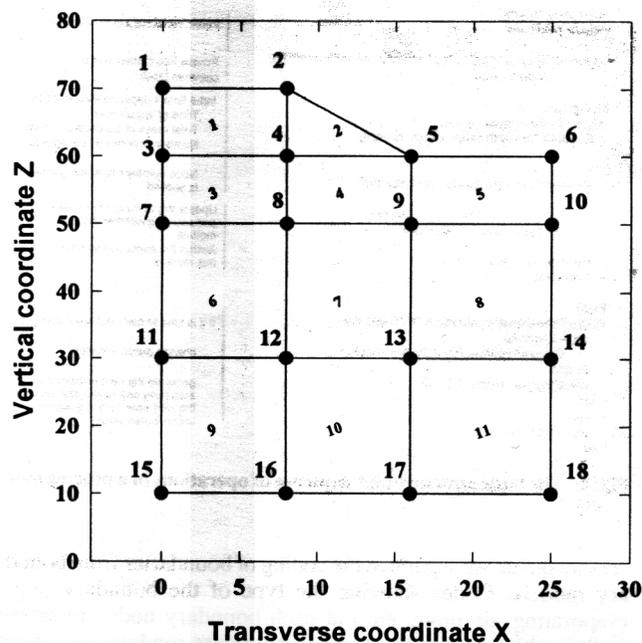


Fig. 2. An example of the spatial grid for a soil profile with numbered nodes (large numbers) and elements (small numbers at an angle).

duction. Therefore, soil state variables and fluxes have to be available to all modules.

Values of the soil and root state variables can be recorded and calculated for specific locations within a soil profile, and these locations have to be the same for all modules. The spatial reference system in the generic simulator is introduced by a spatial grid (Fig. 2). The grid is a polygonal geometric structure representing either a vertical profile of the soil for the one-dimensional mapping of soil variables or a vertical plane cross section for the two-dimensional case. The grid shown in Fig. 2 is given for the purpose of illustration, the grid density may be too coarse for practical use. The grid in Fig. 2 is for the two-dimensional case; a ridge and a furrow form the soil surface. The spatial locations at the intersections of the grid lines where values of state variables and fluxes are known are called *nodes* (Fig. 2). The nodes provide the necessary spatial reference for interactions among soil and root processes, and for the boundary interface with plant and atmosphere models. The nodal coordinates, therefore, have to be available to all modules. The grid is set by a control module, 'Grid_Boundary_Setting'.

Modules have to be synchronized to calculate state variables and fluxes for simulated times that are the same for all modules. To do this, we used a sequential iteration approach (Yeh and Tripathi, 1991). Process modules execute sequentially as shown in Fig. 1. It is assumed that the values of fluxes are available in the beginning of the time step, before the transport codes are called.

Each process module may have its own requirements for a time increment at any point in the simulation. For example, the time interval between fertilizer applications can be 60 d, the atmospheric boundary can be modified hourly, and calculations of infiltration may require time increments on the order of seconds. Combining modules that work with different time steps into one simulator is referred to as an asynchronous coupling (ten Berge et al., 1992).

The control module 'Synchronizer' calculates the actual time step that the whole simulator will use. It is invoked at the beginning of every time step. It uses the requirements of all modules for the next time step and the numbers of iterations that have occurred during the previous time step as input. The 'Synchronizer' calculates the optimum time increment that (i) allows

for convergence of iterations within modules, (ii) enables the simulator to read or to deliver data at specified times, (iii) does not increase drastically as the simulated time progresses. If there are no modules with time step requirements, the time step will increase gradually until it reaches a prescribed maximum value. Current time and time step, simulated time, number of iterations, and time step limitations are also available to all process modules to use and modify.

Data Structure

We used the concept of encapsulation of information to facilitate independence of modules. Encapsulation, a characteristic of OOP, is the grouping into a single module of both the data and the operations that modify or use the data (Wirfs-Brock et al., 1990). The data and operations, encapsulated in a single module, can be hidden from the developer of other program units.

In order to implement data encapsulation in FORTRAN, we divided the variables into public and private fields. Public variables are available to two or more modules. Private variables are available only to the module in which they are declared; they are not available to other modules. Public variables are further divided into two fields, global public and local public. Global public variables are available to all process modules. Local public variables are shared between a process module and its submodules. The use of public and private fields is illustrated in Fig. 3. The global public field includes soil and root state variables, fluxes, nodal coordinates, and temporal variables described in the previous section. The private data field may include control variables to read or print data, parameters to calculate coefficients of the numeric algorithms coded in the module, and state variables and fluxes from the previous time step. Local public variables can include matrices passed to a submodule to solve for coefficients of a simultaneous set of equations or the coefficients in the model equations, such as thermal conductivity. If any module requires data from an external file to fill its private data field, it opens and reads its own file. One drawback of this scheme, however, is that the number of data files will grow as modules are added. Because the size of each file is small, it should not be difficult to manage the files. Each module can also produce output according to its own schedule.

Submodules that calculate coefficients for equations to represent a model are also encapsulated within process modules. Neither data nor equations of submodules of a particular process module are needed for the functioning of other process modules. The recommended data structure includes a subdivision into information that is locally public (i.e., available to one or more submodules) or private (i.e., information hidden within a submodule) (Fig. 3). The hidden information is read independently, and therefore the submodule can be easily replaced or modified. A need to replace a submodule is as common as a need to replace a process module. Consider, for example, the water movement module that requires hydraulic conductivity values and water content as a function of the water potential. The replacement of equations to calculate these values may be desirable (Alessi et al., 1992). The replacement will not affect the process module which needs only the value of the hydraulic conductivity, not the equations used to obtain this value. Therefore, the submodule can read its own parameters from its own data file and have them hidden from the process module and the whole simulator.

Global public variables are passed in FORTRAN COMMON blocks to process modules; no arguments are used in CALL statements at the process level. The same COMMON blocks are placed in all process modules. Errors are minimized by the use of INCLUDE statements, to insert a file containing a list of named COMMON blocks into each process module. If it is desirable to transfer some private variable into a public field, only the insert file

```

Subroutine Process_module_1()
COMMON /global variables/Global_var1,Global_var2,...

COMMON /Module_1_private/ Priv_var1,Priv_var2...
Code

Call sub1(var1,var2,var3)

End

Subroutine sub1(var1,var2,var3)
COMMON /Sub1_Private/ Priv_var1,Priv_var2...

Return

```

Global_var1, etc. are available to all process modules. This same statement is present in all process modules.

The COMMON statement, Module_1_private, is present only in this module

var1,var2 and var3 are locally public between sub1 and process_module_1

The COMMON statement, Sub1_private, is present only in this subroutine

Fig. 3. Implementation of global public, local public, and private variables in 2DSOIL.

referenced in the INCLUDE statement has to be changed. Local public variables (i.e., variables shared between process modules and their submodules) are passed in CALL statements (Fig. 3).

FORTTRAN COMMON blocks are also used to store private variables within a module or submodule. The primary reason for using COMMON blocks for private variables was to save the values of private variables between invocations of the subroutine. Any particular COMMON block containing private information is present in only one module. Because there is no reference to this block in other program units, the information remains hidden.

Structure of a Process Module

The general sequence of operation includes reading the time-independent data, calculating the auxiliary variables to speed up routine calculations, checking whether it is time to execute and (if it is time to execute) reading the time-dependent data, changing the public variables, changing the private variables, calculating the requirements for time increments, and writing output data. Some steps may be absent.

An example of a typical module is shown in Fig. 4. This module simulates a chemical application. At the start of the module, designated by the public variable *LInput*, it reads time-independent private data that are listed in the private COMMON block and will be stored. The number *NumMod* of the module in the invocation sequence is used to keep the place for the module time step requirements. This number is stored as private information. The public variable *tNext* stores the time to execute code for a particular activity, therefore, one of the future time steps has to be ended exactly at this time. When the execution time arrives, concentrations of the chemical (which are public) in prescribed nodes are changed by dividing mass of chemical applied (which is private) by soil water content (which is public). The public variable *tNext* is finally assigned an unreachable value (1^{32} , written $1.0E+32$) which means that there are no further restrictions on the time step from this module.

Boundaries in the Context of Modularity

The boundary interface mentioned in Fig. 1 has to provide exchange of data between the soil simulator and both plant and atmosphere components of a crop model. This interface is extraneous for the generic soil-root simulator, but any particular simulators derived from the generic structure will require such an interface and both the corresponding plant and atmosphere components to run.

To simplify the development of interfaces for the soil-atmosphere boundaries and enhance the reusability of transport pro-

```

Subroutine Mngm()
Include 'public.ins'

Common /Mngm/ tAppl, cAppl, NumAp, nAppl(NumBPD),
Module_Num

If(LInput.eq.1) then
Open(40,file='Param_M.dat')
Read(40,*,Err=20) tAppl, cAppl, NumAp

Read(40,*,Err=20) (nAppl(i),i=1,NumAp)

Num_of_Modules=Num_of_Modules+1
Module_Num=Num_of_Modules

tNext(Module_Num)=tAppl
Close(40)

Endif
If(Abs(Time-tNext(NumMod)).lt.0.1*Step) then
Do = 1, NumAp
Conc(nAppl(i))=cAppl/ThNew(nAppl(i))
Enddo
tNext(Module_Num)=1.E+32
Endif
Return
20 Stop 'Mngm data error'
End

```

Public variables are in "Public.ins"

Private information is in a common block

Input time independent information
Time of application
Total mass of chemical per node
Number of nodes where applied

Node numbers to which chemical is applied

Update the number of modules and assign a number to this module
Assign the execution time for this module

If it is time to carry out any calculations:
change public variables
generate the next proposed time step (only one application here so the next time step is assigned an unreachable value)

Fig. 4. The basic structure and sequence of operations in a process module.

cess modules, we separated the coding of boundaries from boundary models. Codes showing the type of the boundary (e.g., evaporating, draining, etc.) at each boundary node are stored in the public data field. Transport process modules use these codes to process information supplied by the interface. Transport modules, however, do not read data on boundary fluxes, concentrations, and the like. These values have to be supplied by the interface as public variables that will be used by the transport module.

We considered and then rejected including the potential boundary flux calculations in transport modules. We specifically avoided having the water transport module input values of evaporation or precipitation, because this would mix the water transport code with the code to determine surface boundary fluxes. Later modifications to the method of determining evapotranspiration or precipitation would then require changes in the water transport code as well. In our design, the water transport code is provided values of potential fluxes and is then free to modify these fluxes on the basis of the current soil conditions only.

Another component of the boundary interface supplies data on the shoot status to the root activity module and returns data on the availability of water and nutrients to the plant. The data are specific to the plant model. It was important for our purposes to realize that the number of variables relating shoot and root is very small in the majority of plant models. A list of these variables typically may include potential transpiration and/or leaf water potential, C available for root growth, available soil water supply and N flux, and indexes of soil stresses. We assumed that the interface calculates the potential root water uptake and potential C supply to roots, and the root module returns the actual supply of water and nutrients together with the actual C use. A separate COMMON block is designated to contain shoot-root interaction variables. It has to be filled as required for a particular combination of shoot and root models.

Input and Output of Data

The process modules and some submodules read their own data files, so there is no input module. The control modules (Grid_Boundary_Setting, Synchronizer) read their own information in a predefined format. We did not include any modules to generate grids. It is assumed that either finite element or finite difference methods may be used for the numerical estimation of the intrasoil transport, so data for both nodes and grid cells (elements) are required. If the user chooses modules that use the finite difference or simplified mass balance techniques, then the element data will not be used.

Table 1. Process modules used in 2DSOIL and their sources.

Module name	Process	Source
<u>Transport</u>		
WaterMover	Water redistribution according to two-dimensional Richards' equation	Šimůnek et al., 1992
SoluteMover	Convective-dispersive transport of several solutes	Šimůnek et al., 1992; Istok, 1989
HeatMover	Diffusive-convective redistribution of heat	Šimůnek et al., 1992
GasMover	Transport of gases by diffusion in air-filled pore space	-†
<u>Soil-atmosphere boundary</u>		
SetSurf01	Hourly fluxes of water, solutes, heat and CO ₂ at the soil surface, governed by parameters of atmosphere and plant shade	Acock and Trent, 1991
SetSurf02	Piecewise dependencies of boundary fluxes or state variables on time.	-
<u>Root activity</u>		
RootUptake	Root growth, and water and solute uptake based on the trade of carbon and water between shoot and root. Root respiration.	Acock and Trent, 1991
<u>Interphase mass exchange</u>		
MacroChem	Cation exchange Ca-Mg-Na; dissolution precipitation of gypsum and carbonates.	Pachepsky, 1990
<u>Biotransformation</u>		
NitroChem	Mineralization and/or immobilization of soil organic matter. Nitrification and denitrification.	Bergstrom et al., 1991
<u>Management</u>		
Tillage	Soil disruption and subsidence.	-
Fertilizer application	Addition of organic and inorganic fertilizers.	-
Irrigation	Addition of water as needed.	-

† The module has been developed by us to complete an example simulator.

A simple output module is available that prints soil and state variables, together with nodal coordinates, at prescribed times in the simulation.

APPLICATION AND EXAMPLES

To test and apply the modular design, we assembled a representative simulator called 2DSOIL. The process modules used in 2DSOIL and their sources are summarized in Table 1. The modules were adopted from other computer models that were in the public domain and had been written in FORTRAN. The code was chosen on the basis of its reliability and understandability because we had to restructure the code to conform with the modular structure described above. The restructuring required only moderate effort since the code was well documented. The

resulting 2DSOIL simulator is fully documented in Pachepsky et al. (1993). We chose to use two-dimensional water transport because the importance of the interplay between vertical and horizontal transport in soils has been demonstrated elsewhere, and many crop models already rely on a two-dimensional representation of the soil profile. The simulator is not limited to two-dimensional transport, however. Because 2DSOIL is a generic simulator, any of the modules can be replaced as necessary following the design described above. For instance, we developed a one-dimensional water transport module for use in another application (Kemp et al., 1995). Concentrations and water contents per unit of soil volume are computed by both one-dimensional and two-dimensional transport modules, assuming a standard thickness of 1 cm of a two-dimensional layer or a standard cross-sectional area of 1 cm² of a one-dimensional soil column.

To develop examples of applications, we needed to assemble a crop model and a boundary interface. We chose the atmosphere simulator described by Acock and Trent (1991). This simulator derives hourly potential fluxes of water, solutes, heat, and CO₂ at the soil surface from weather parameters (radiation, precipitation, minimum and maximum daily air temperature, and wind speed) and surface shading governed by the plant height, plant row orientation, and row spacing. To provide shoot-root interactions, we used a simple shoot imitator that calculates plant height and available C from an index of soil water availability and plant age. Both boundary interfaces are described in Pachepsky et al. (1993).

Two examples are presented below. The first example was chosen to demonstrate that complex interactions of soil transport processes, root activity, and canopy influence may occur, and that the 2DSOIL simulator with a crop simulator can be useful for understanding these interactions. The second example was chosen to show that the 2DSOIL simulator can be readily expanded without any changes in structure or in control modules.

The first example describes the soil temperature regime in ridge and interridge zones as affected by plant canopy, water table, and water uptake by roots. The grid used in this example has a shape similar to the one in Fig. 2. The width of the top of the ridge is 0.1 m, and the width of the bottom of the ridge is 0.3 m; the furrow width is 0.5 m. The matric potential was kept constant (-30 kPa) at the 0.8-m depth. This simulates a water table at approximately 4 m in depth. The soil parameters used in the model are listed in Table 2.

The results, in Fig. 5, show how simulated surface (0.05 m) soil temperature responded in the presence of water uptake by roots under a fully developed canopy. In

Table 2. Parameters used in the heat and solute transport simulations.†

Layer	Sand	Silt	Clay	Diffusion constant	θ_s	θ_r	α	η	K_{sat}	ρ_b
	%			cm ² d ⁻¹	m ³ m ⁻³		1 m ⁻¹		mm h ⁻¹	Mg m ⁻³
<u>Heat transport simulation</u>										
1	45	30	25	-	0.426	0.001	0.023	1.103	25.0	1.52
2	45	37	18	-	0.390	0.001	0.016	1.240	21.7	1.62
<u>Solute transport simulation</u>										
1	-	-	-	12.0	0.399	0.001	0.017	1.300	12.4	1.52

† θ_s , saturated water content; θ_r , residual water content; ρ_b , bulk density; K_{sat} , saturated hydraulic conductivity; α and η , coefficients for the van Genuchten water retention equation.

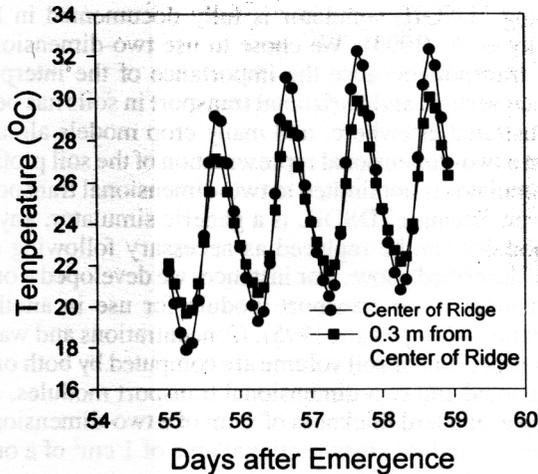


Fig. 5. Simulated soil temperature at 0.05-m depth in a ridged soil when the plant canopy covers the entire soil surface.

this case, water uptake occurred primarily in the ridge, directly below the crop. After 35 d after emergence, there was no more precipitation; there was some contribution of water moving up from the water table, however. The data shown are from 54 to 60 d after emergence. Plant shade has covered all of the interridge zone. The average simulated water contents were $0.25 \text{ m}^3 \text{ m}^{-3}$ in the ridge and $0.30 \text{ m}^3 \text{ m}^{-3}$ in the furrow at the 0.05-m depth. As shown in Fig. 5, the surface soil in the ridge had a wider range of temperature fluctuation. The soil there was warmer at midday and cooler at night. This occurred because the ridge was drier than the interridge zone due to higher elevation above the water table, and had greater root mass, which resulted in more water uptake. Benjamin et al. (1990) showed similar results for field data from an experiment with two ridge shapes, one ridge more peaked than the other. The temperature amplitude was larger in the more peaked ridge, which had the smaller water content.

The second example demonstrates the effect of root water

uptake on solute distributions in the upper 0.8 m of the profile. This example also demonstrates how a management module can be used to add a soluble salt to selected surface nodes at a specified time. By placing the module at the same level as the other process modules, some of its code will be executed at every time step. A data file was also created for the module. The data included the time of application, amount of solute to be applied, and the node numbers to receive the solute. The new module was compiled and linked with the other modules. No other changes in any other modules were required. The FORTRAN code for this module is in Fig. 4.

In the simulation, a solute in the amount of $300 \mu\text{g cm}^{-2}$ was added to the soil surface 44 d after emergence and then rainfall was applied. The net amount of water added to the soil on Day 44 was 22.5 mm. After Day 44, an additional 73 mm of rainfall was applied at four different times. Other parameters for the soil hydraulic properties and solute are given in Table 2.

In Fig. 6, we show the root distribution at 44 d after emergence and the two-dimensional pattern of water content in the profile that results from water uptake by roots. The root distribution is concentrated in the soil under the plant (Fig. 6a). Water contents were lowest under the plant, where the root distribution was highest (Fig. 6b). Water content increased with depth and horizontal distance from the plant. After rainfall on Day 44, water has infiltrated deeper in the interrow position, because the initial water content was greater in this position (Fig. 6c). Transpiration and several periods of rainfall continue after Day 44 until the simulation ends at Day 56. The chemical concentrations directly below the plant row (at $x = 0.01 \text{ m}$) and midway between rows ($x = 0.35 \text{ m}$) at Day 56 are plotted in Fig. 7. The solute mass in the soil under the row position was greater than under the interrow position. Furthermore, the amount of solute near the surface below the plant was still rather high, despite two rainfalls after solute application. Evaporation and transpiration act to move the water with the solute toward the soil surface

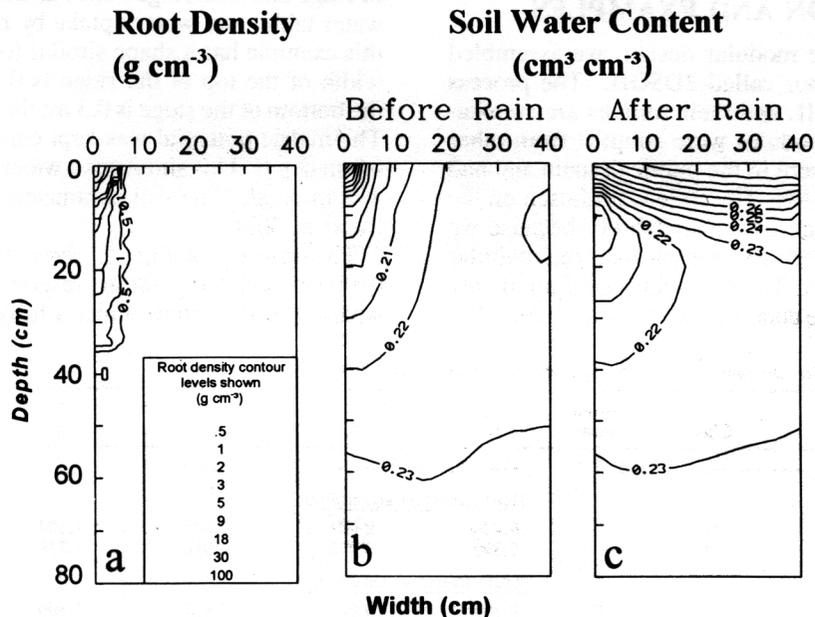


Fig. 6. Simulated root density and water content at Day 44 before chemical application and rainfall, and water content after chemical application and rainfall for the second example.

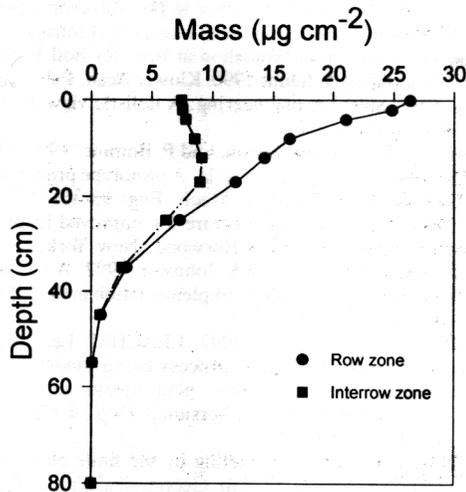


Fig. 7. Simulated mass of solute in the upper 0.8 m vs. soil depth in the row zone ($x = 0.01$ m) and in the interrow zone ($x = 0.35$ m) for the second example.

and toward the plant. This results in concentration of solute in the soil under the plant and depletion of solute in the interrow zone.

The results of this simulation are in qualitative agreement with observed effects of plant water uptake or management practices on solute distributions in soils. Timlin et al. (1992) measured higher levels of bromide in the upper 0.5 m of soil under plant rows than in soil in interrow zones. The bromide had been applied uniformly to the soil surface and there were several periods of rainfall during the season. Benjamin et al. (1994), using SWMS_2D (Šimůnek et al., 1992), predicted less downward displacement under furrow irrigation of a tracer placed in the ridge as compared with furrow placement. This compared favorably with experiments carried out by Kemper et al. (1975).

DISCUSSION AND CONCLUSIONS

The paradigm of modularity used in this paper is much more than dividing a program into a number of subroutines and then calling them in sequence. Modules should be loosely coupled (i.e., with a simple interface between modules and a minimum of information passing). They should also have high cohesion (i.e., interdependent elements packaged together and information hiding). Because the ability to hide information is critical, the questions of what comprises a module and how to best encapsulate the information must be answered. The answers depend on several factors, including the scale (i.e., plant, field, or landscape scale), the desired level of detail of the system to be modeled, and the programming environment (e.g., FORTRAN, C⁺⁺).

In the application 2DSOIL, the modules were organized around processes primarily because the structures of the existing programs incorporated into 2DSOIL were already based on processes and because this kind of structure is suited to FORTRAN. A modular soil simulator could also be organized on the level of a soil cell (Dubois-Pelerin et al., 1992). Here a cell is defined as a polygonal element with a boundary, as shown in Fig. 2. Calculations for mass

transport, root growth, and transformations can be encapsulated in a cell module. This type of structure is much more easily implemented in an object-oriented language than in FORTRAN. Soil cell objects (modules) will have methods (functions) depending on their role in the plant-soil system. New types of cells could easily be derived from a base class (e.g., cells with macropores, or boundary cells).

The generic, modular structure described in this paper offers many advantages to the development and maintenance of agricultural management models. These advantages include:

1. Ability to reuse code. Our example generic simulator, 2DSOIL, used code from several models.
2. Computer code can be easily modified and evaluated. This simplifies the process of incorporating into models ideas derived from experiments.
3. Submodels, which are components of a larger model, can easily be tested and validated in more than one model. For example, models of N dynamics can be evaluated in both crop growth and water transport models. The evaluation would not be limited to one particular model.
4. Users can add management practices as modules to build models for specific tasks from program units developed by other researchers.
5. Because the developer of a particular process model does not need to know the details of the other modules, developers gain access to a wide range of code in areas outside their fields of expertise.

Code and documentation for the sample generic simulator, 2DSOIL, are available from the authors.

The modular design presented here was developed mainly as a framework for crop modelers to interface their plant and atmosphere codes with reliable soil code. To do this, the crop modeler needs to concentrate only on the boundary interface. The modeler must (i) assign potential boundary fluxes of water, solutes, heat and gases from their atmosphere module to the nodal boundary fluxes, (ii) receive actual boundary flux values from the water transport module and use them as needed, (iii) pass potential transpiration and C flux values from the plant module to the root module, (iv) receive actual transpiration, water and nutrient fluxes from the root module and use them as needed, and (v) provide simulated times when the plant and atmosphere modules will be ready to exchange information. All variables needed for this exchange are global public variables, and no code has to be changed in the other modules of the soil simulator. If some other soil variables are needed by the plant and atmosphere components, they are accessible through the global COMMON block since all global public data for the boundary interface are available there. If some private root variables are needed by the shoot module, they can be made accessible by inserting a local public COMMON block of the root module into the boundary interface. Finally, if the crop modeler wants to use his or her own root module, the module must be rearranged to fit into the data structure of the design described here.

We have shown how existing FORTRAN program units, when clearly written, can be combined into larger units

using a modular structure. These units are easily modified or replaced. FORTRAN was used because it has been an important programming language for the scientific community and there is a large amount of available code. True object-oriented programming languages, however, such as SmallTalk and C++, offer many advantages for programming complex models and should be investigated. A finite element model written in SmallTalk has recently been published (Dubois-Pelerin et al., 1992), and a cotton (*Gossypium* spp.) model written in C++ incorporating 2DSOIL is under development (Hal Lemmon, personal communication, 1995).

The segregation of a process or group of processes into a particular module is not always straightforward, however, and will probably always be, to some degree, arbitrary. If we are to develop generic, modular simulators as Reynolds et al. (1992) suggested, we will need to further explore the issue of modularity and the basis for creating modules. This is similar to the problems facing taxonomists: i.e., what should be included in a group and what should be excluded.

ACKNOWLEDGMENTS

The authors wish to thank Drs. Jerka Šimunek, Rien van Genuchten, and Tomas Vogel for providing the finite element model, SWMS_2D, which was used as the water mover module and for their helpful comments while the code was being modified.

REFERENCES

- Acock, B., and A. Trent. 1991. The soybean simulator, GLYCIM: Documentation for the modular version 91. Agric. Exp. Stn., Univ. of Idaho, Moscow.
- Ahuja, L.R., D.G. Decoursey, B.B. Barnes, and K.W. Rojas. 1991. Characteristics and importance of preferential macropore transport studied with the ARS root zone water quality models. p. 32-42. *In Proc. Natl. Symp. Preferential Flow*, Chicago, IL. 16-17 Dec. 1991. ASAE Publ. 9. ASAE, St. Joseph, MI.
- Alessi, S., L. Prunty, and W.M. Schuh. 1992. Infiltration simulations among five hydraulic property models. *Soil Sci. Soc. Am. J.* 56: 657-682.
- Benjamin, J.G., M.R. Ghaffarzadeh, and R.M. Cruse. 1990. Coupled water and heat movement in ridged soil. *Soil Sci. Soc. Am. J.* 54:963-969.
- Benjamin, J.G., H.R. Havis, L.R. Ahuja, and C.V. Alonso. 1994. Leaching and water flow patterns in every-furrow and alternate-furrow irrigation. *Soil Sci. Soc. Am. J.* 58:1511-1517.
- Bergstrom, L., H. Johnsson, and G. Tortensson. 1991. Simulation of soil nitrogen dynamics using the SOILN model. p. 181-198. *In J.J.R. Groot et al. (ed.) Nitrogen turnover in the soil-crop system: Modelling of biological transformations, transport of nitrogen and nitrogen use efficiency. Proc. Workshop at Inst. for Soil Fertility Res., Haren, Netherlands. 5-6 June 1990. Kluwer Acad. Publ., Dordrecht.*
- Blum, B.I. 1992. *Software engineering: A holistic view.* Oxford Univ. Press, New York.
- Dubois-Pelerin, Y., T. Zimmerman, and P. Bomme. 1992. Object oriented finite element programming: II. A prototype program in SmallTalk. *Comput. Methods Appl. Mech. Eng.* 98:361-397.
- Einbu, J. 1989. *A program architecture for improved maintainability in software engineering.* Ellis Horwood, New York.
- Hodges, T., S.L. Johnson, and B.S. Johnson. 1992. A modular structure for crop simulation models: Implementation in the SIMPOTATO model. *Agron. J.* 84:911-915.
- Hutson, J.L., and R.J. Wagenet. 1992. LEACHM. Leaching Estimation and Chemistry Model: A process based model of water and solute movement, transformations, plant uptake, and chemical reactions in the unsaturated zone. Version 3. Dep. of Agronomy, Cornell Univ., Ithaca, NY.
- Istok, J. 1989. *Groundwater modeling by the finite element method.* Water Resources Monogr. 13. Am. Geophysical Union, Washington, DC.
- Kemper, W.D., J. Olsen, and A. Hodgdon. 1975. Fertilizer or salt leaching as affected by surface shaping and placement of fertilizer and irrigation water. *Soil Sci. Soc. Am. Proc.* 39:115-119.
- Kirk, B.R. 1990. Designing systems with objects, processes, and modules. p. 387-404. *In P. Hall (ed.) SE90: Proc. Software Engineering 90*, Brighton, UK. July 1990. Cambridge Univ. Press, Cambridge.
- Pachepsky, Y. 1990. *Mathematical models of physical chemistry in soil science.* Nauka, Moscow.
- Pachepsky, Y., D. Timlin, B. Acock, H. Lemmon, and A. Trent. 1993. 2DSOIL: A new simulator of soil and root processes. Version 02. Systems Res. Lab. Publ. 2. USDA-ARS, Beltsville, MD.
- Reynolds, J.F., B. Acock, and R. Whitney. 1992. Linking CO₂ experiments and modeling. p. 93-106. *In E.-D. Schulze and H.A. Mooney (ed.) Design and execution of experiments on CO₂ enrichment.* Springer-Verlag, Berlin.
- Šimunek, J., T. Vogel, and M.T. van Genuchten. 1992. The SWMS_2D code for simulating water flow and solute transport in two-dimensional variably saturated media. Version 1.2. Res. Rep. no. 132. U.S. Salinity Lab., USDA-ARS, Riverside, CA.
- ten Berge, H.F.M., D.M. Jansen, K. Rappoldt, and W. Stol. 1992. The soil water balance model SAWAH: Description and users guide. Simulation Rep. CABO-TT No. 22. DLO-Ctr. for Agrobiological Res. (CABO-DLO) and Dep. of Theoretical Production Ecology (TPE), Wageningen, Netherlands.
- Thomas, D. 1989. What's in an object? *Byte* 14:231-240.
- Timlin, D.J., G.C. Heathman, and L.R. Ahuja. 1992. Solute leaching in row vs. interrow zones. *Soil Sci. Soc. Am. J.* 56:384-392.
- Wirfs-Brock, R., B. Wilkerson, and L. Weiner. 1990. *Designing object oriented software.* Prentice Hall Publ. Co., Englewood Cliffs, NJ.
- Witt, B.I., F.T. Baker, and E.W. Merritt. 1994. *Software architecture and design: Principles, models, and methods.* Van Nostrand Reinhold, New York.
- Yeh, G.T., and V.S. Tripathi. 1991. A model for simulating transport of reactive multispecies components: Model development and demonstration. *Water Resour. Res.* 27:3075-3094.