ELSEVIER Ecological Modelling 94 (1997) 67-80

ECOLOGICAL MODELLING

A modular soil and root process simulator

Dennis J. Timlin[a,*], Yakov A. Pachepsky[a,b]

[a] *USDA-ARS, Systems Research Laboratory, Building 007, Room 008, BARC-W, 10300 Baltimore Avenue, Beltsville, MD 20705, USA*

[b] *Duke University Phytotron, Department of Botany, Durham, NC 27710, USA*

Abstract

The ability to build models for various crop management scenarios can be enhanced by using modules corresponding to soil, root atmosphere and management processes. In this paper we describe the design of a generic modular soil and root process simulator (2DSOIL) for use in crop modelling. Coded in Fortran 77, it uses a three-level hierarchy to organize soil processes and their parameters into a modularized structure. Decoupling of processes and re-arrangement of boundary condition formulations are used to facilitate independence of modules and the encapsulation of information is consistently implemented. Variable time steps are selected using restrictions imposed by non-linearity of models. The use of this modular design allows modellers to reuse well-tested codes, and results in a decrease of effort in input data preparation and in expandability to encompass various management practices. The modular design of 2DSOIL allows it to be readily modified and easily incorporated into crop models. © 1997 Elsevier Science B.V. All rights reserved

Keywords: Soil modeling; Modularity; Encapsulation; Generic simulator; Crop modeling

1. Introduction

Agricultural chemicals represent a serious threat to surface and ground water supplies. While it is imperative that researchers find ways to minimize this risk, it is impossible to directly test the consequences of a large number of management practices and techniques under natural conditions. Crop modelling offers an alternative that is both efficient and quick and enables researchers to explore a wide range of management scenarios to assess their potential without risk to the environment.

To date, the emphasis in agricultural modelling has been primarily on crop productivity. As a result, plant simulators typically have been developed with comprehensive descriptions of plant processes such as leaf and canopy photosynthesis, growth, nutrient uptake, light relations in canopies, and respiration (e.g. Baker et al., 1983; Reynolds and Acock, 1985; Thornley and Johnson, 1990; Acock and Trent, 1991; Hodges et al.,

* Corresponding author.

1992). On the other hand, soil processes per se have often been treated in an overly simplistic manner. It is now clear that, in order to use crop models to explore management scenarios that consider both crop productivity and the fate and distribution of chemicals, comprehensive descriptions of key soil processes are also needed.

There are several comprehensive soil simulators available that reflect the state-of-art in soil science, e.g. Ahuja et al. (1991); Hutson and Wagenet (1992), and Šimünek et al. (1994). These models, however, are complex stand-alone models that were not designed to be expanded and/or interfaced with plant simulators. There is a need to design and implement soil simulators that can easily be incorporated into crop models.

The design of a framework that will enable us to interface comprehensive soil simulators with crop models must be guided by two important factors. First, is the ability to choose a particular soil model from the large number available. The existence of a multitude of mathematical models for each soil process reflects differences in scientific concepts and available data. There is no way to ascertain which model will perform better without testing them against independent data. Moreover, a model of a particular soil process usually can not be tested independently of models of other processes because of interactions. The second guiding factor is the ability to model a range of management practices and easily add new ones. Therefore, the design of a generic soil simulator should emphasize: (1) replaceability of sub-models of soil and management processes; and (2) expandability. The availability of generic software with these properties would have other advantages for the development of complex simulation models as well.

It is neither practical to develop ad hoc models for each question of interest nor is it possible to build a single, all-purpose model to answer all questions (Acock and Reynolds, 1989, 1990; Reynolds et al., 1993). Reynolds et al. (1989) suggested that the goal should be to build a suite of models that are based on general principles derived from structures and behaviours that are fundamentally similar in different plants and ecosystems. To accomplish this, they proposed the development of 'generic' models that have a well defined and uniform modular structure. These generic models also use similar modules, some of which may be used in several models with appropriate parameter changes. In a generic, modular structure, related processes are grouped in sub-model components, and the function of each module and the variables are explicitly defined. Numerous advantages are potentially realized including improved understanding and maintenance, reduced duplication of efforts, and easier inclusion of experimentalists in modelling (also see Reynolds and Acock, 1997).

The objective of this work was to develop a generic soil simulator that is based on the principles of modularity. The generic soil simulator is not a single model but rather represents an approach that establishes the basis for a suite of sub-modules that can incorporate various descriptions of soil processes and management practices.

2. Conceptual development

2.1. Modular design

Three considerations governed the design of the generic simulator:

(1) it should represent existing scientific concepts of the processes and their interactions;
(2) As software under development, it should be designed as a system of interacting pieces that can be independently tested and provide intellectual control of the development (Mills, 1980);
(3) As software subject to modifications by users, it should be malleable (Witt et al., 1994), i.e. it should facilitate adaptation to changing end-user requirements. As pointed out by Kirk (1990), we have to optimize ease of comprehension, ease of development by teams, and ease of extension.

The use of a modular structure allows us to satisfy these requirements.

Modularity is an acknowledged principle of computer software design (Radice and Philips, 1992; Witt et al., 1994; Blum, 1992; Parnas, 1972). There is general agreement about desirable attributes of modules: modules have to be independently testable, modules must have high internal cohesion (i.e. represent one logical self-contained task) and finally, modules must have loose inter-unit coupling. Loose coupling is characterized by having a single entry point and a single exit point in the computer code, a limited number of well-defined interfaces, and a minimum of data passing. However, there is no unique recipe of how to subdivide a system into modules. One general guideline is to divide 'large aggregates of components into units having loose inter-unit coupling and high internal cohesion...' (Witt et al., 1994).

From this point of view, models of the various soil and root processes can be seen as natural candidates to be coded in separate modules. It is easy to assign correspondence between model and system processes where the process is used to define the structure and activity of a component of the system being modeled. Our modular design is based on a representation of the system as a set of 'cooperating elements' where elements represent major soil and plant processes (Kirk, 1990). One element, for example would be water uptake by roots, another would be water flow. In this way, the system requirements are analyzed to produce a non-hierarchical logical model of the system in terms of these elements. In addition to process modules there are control modules that oversee the activities of the process modules and manage common tasks.

2.2. Implementation of the modular design

To satisfy the requirements for information exchange and independence, the soil-root system is represented as a non-hierarchical system of objects or elements corresponding to real processes that belong to one of the following groups:

(a) transport processes, e.g. water flow, heat movement, nitrate transport, and oxygen movement;

(b) atmosphere processes, e.g. surface evaporation, nitrogen influx with precipitation;

(c) root processes, e.g. water uptake, NH_4 uptake, root respiration;

(d) management processes, e.g. subsoil irrigation, tillage, nitrogen fertilizer application;

(e) interphase exchange processes, e.g. reversible Ca–Mg–Na exchange;

(f) biotransformation processes, e.g. denitrification, CO_2 production.

Each process listed above is represented by a process module. We did not use instances (multiple appearances) of modules in this structure. For example, to simulate transport of several solutes we repeat calculations sequentially for each solute in the single solute transport module (instead of creating a separate module for each solute). Simulation of the activity of several root systems in one soil domain is done in a single module, i.e. there are not separate modules for each root system. Furthermore, each process module opens and reads its own data file or files, there are no input routines that read and manage a global pool of data read from one or more files.

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 object oriented programming (OOP) language. In OOP, an object (module) contains specific information (data) and is coded to perform certain operations (methods). Our design, coded in Fortran 77, follows OOP precepts, but within the constraints of Fortran. We used Fortran because the majority of existing plant models use this language, and we wanted to use code from available and tested soil and root process models.

The simulator uses a three-level hierarchy to structure the components (Fig. 1). This hierarchy determines the structure of the data fields in order to facilitate encapsulation of information. 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). At the highest level is the program itself: the simulation model that has been assembled to represent a particular system. The second level is represented by process modules that are implemented as subroutines. The arrangement of the process modules is non-hierarchical, i.e. all process modules are on the same level. No one process module (subroutine) will call another process module. Each process module may contain one or more sub-modules, however. These

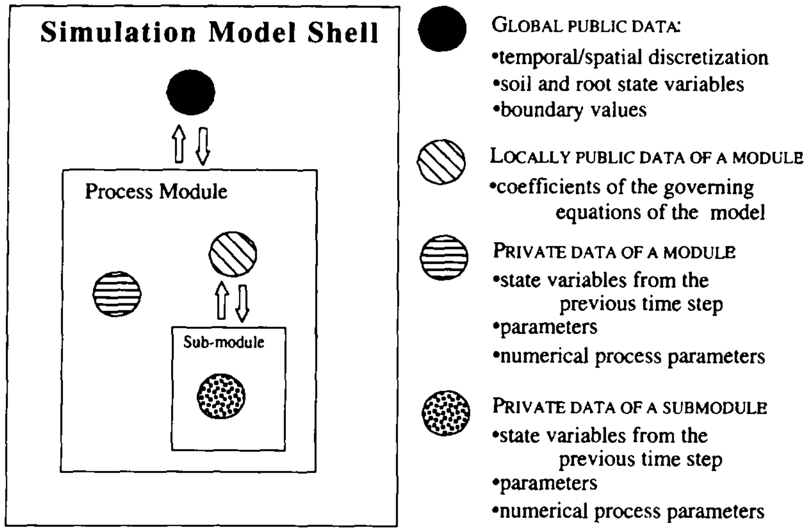
Simulation Model Shell
Process Module
Sub-module
GLOBAL PUBLIC DATA:
•temporal/spatial discretization
•soil and root state variables
•boundary values
LOCALLY PUBLIC DATA OF A MODULE
•coefficients of the governing equations of the model
PRIVATE DATA OF A MODULE
•state variables from the previous time step
•parameters
•numerical process parameters
PRIVATE DATA OF A SUBMODULE
•state variables from the previous time step
•parameters
•numerical process parameters

Fig. 1. Three-level data structure that facilitates data encapsulation.

sub-modules, which represent the third level of the hierarchy, may include algorithms to calculate coefficients for particular equations to represent a model in a process module.

Neither data nor equations that are contained in these sub-modules are available to other process modules. These data and operations, encapsulated in a single module, can be hidden from the developer of other program units. An example is a sub-module in the water transport module that contains equations to calculate soil water content as a function of matric potential. This sub-module receives matric potentials from the water transport module and returns water contents. It reads its own data file that contains parameters for the equations to calculate water content for a particular soil type. If a programmer desires to use a different method to make these calculations he or she only need to replace the submodule and its data file, no other changes in other modules, in other data files or in the water transport module are necessary.

2.3. Sharing data in the context of modularity

The minimum data set available to all modules 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–atmosphere 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 rates of transport of substances through boundaries (Fig. 2) are referred to as boundary fluxes. These include fluxes of heat, solutes, water and gases, and rates of carbon and nutrient exchange.

Modules interact by sharing data, therefore, the framework must be designed to provide a means to share data and maintain loose coupling and high internal cohesion. Loose-coupling requires that the amount of data available to all modules be minimized and represented consistently.

Values of the soil and root state variables are recorded and calculated for specific locations within a soil profile, and these locations have to be the same for all modules. A grid is used for the spatial reference of soil state variables and boundary fluxes. 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. A two-dimensional grid is illustrated in Fig. 3. In this example, triangular and rectangular elements were used together to approximate a soil surface with a ridge. The spatial locations, at the intersections of the grid lines, where values of state variables and fluxes are known are called nodes (Fig. 3). 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. A control module manages the input of nodal coordinates and locations of boundary coordinates.

2.4. Data structure

We used the concept of encapsulation of information to facilitate independence of modules and the data structure follows the three-level hierarchy of the simulator (Fig. 1). The variables are divided

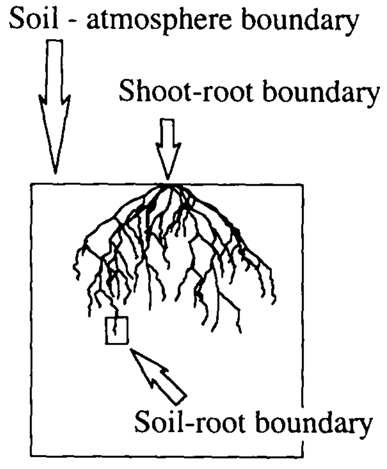

Fig. 2. Information exchange consists of transfer of information across these boundaries of the components of a soil–plant–atmosphere–management system.

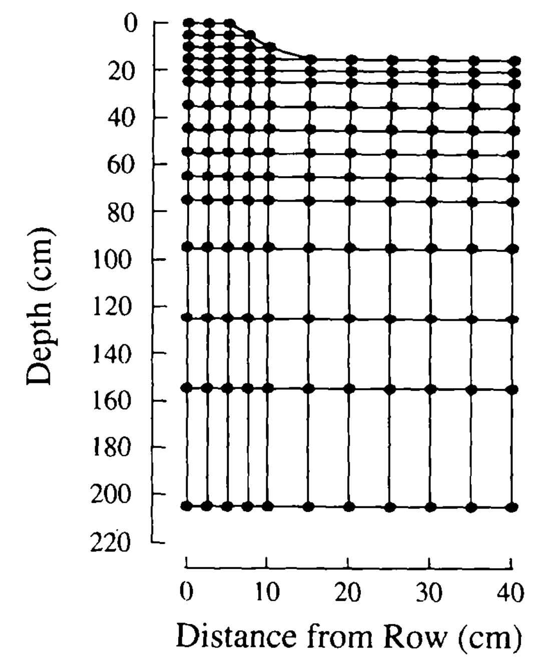

Fig. 3. An illustration of a spatial grid that includes an uneven soil surface in the form of a ridge and furrow. The soil domain includes the area delineated by the x and y-axes. The nodes are indicated by the filled circles.

into public and private fields. Variables that are public are available to two or more modules and may be global public or local public. Global public variables (highest level) are available to all process modules and include variables such as soil state variables, nodal coordinates, and boundary fluxes. If a process module must initialize a particular state variable or any other variable, it opens and reads the data from its own file. Generally, initialization and/or input of a variable is carried out by the module that first modifies that variable. Local public variables (level two) are shared between a process module and one of its sub-modules. These include matrices passed to a sub-module to solve a system of simultaneous equations, or the coefficients for a model equation, such as thermal conductivities, for a heat transport equation. The private data field (level three) is not available to other modules and includes, for example, control variables to read or output data, and state variables and fluxes from the previous time step. If a module or sub-module requires data to fill its private data file, it opens and reads it's own data file.

As a part of its specification, each process module has a precisely defined list of global public variables that it can change during invocation. To do so, a process module may use any of the available global variables (and, of course its private variables) (Fig. 1). Similarly, the list of the process module's locally public variables to be changed is defined for each submodule in this module. Finally, there are no limitations on which available private data a submodule may use.

Components of the soil-root-atmosphere system are separated by boundaries (Fig. 2) and information exchange concerns mass and energy exchange across these boundaries. This information includes boundary fluxes. An atmosphere module provides values of potential atmospheric boundary fluxes such as precipitation, evaporation, heat flux, etc. The plant module may provide a value of carbon available for root growth to the root module. The soil and root modules also provide soil and root state variables, i.e. vertical (and in the 2D case, lateral) distribution of water content, concentration of solutes and gases, temperature, and root densities. In practice, the atmosphere module using information on plant status (leaf area index, height, etc.) supplies the value of potential root water uptake and the plant module supplies the value of the potential carbon supply to roots. The root and soil modules return the actual water and nutrient uptake together with actual carbon use to the plant module which then uses this information to determine stresses.

The potential values of boundary fluxes are supplied by a specific module and are generally constant over a time step. The potential values are passed to the soil and root modules where they are adjusted based on soil or plant conditions. Transport modules do not read data on atmospheric boundary fluxes or concentrations, etc. These values are supplied by the atmosphere or management modules as public variables that can be used by the transport modules. Modularity is enhanced by keeping the source of the value separate from the module where the value is modified or used. For example, several models are available to calculate potential evapotranspiration (PET). By maintaining the code to calculate PET in a separate atmosphere module only changes to that module have to be made if a new model of PET is chosen. The water flow code which uses PET to calculate actual evaporation does not have to be modified. Potential values of fluxes may also be given as a function of a state variable, i.e. $q = -ay + b$ where y can be a matric potential, solute concentration, temperature or gas content and q can be flux of water, solute, heat or gas, respectively; and a and b are coefficients that do not depend on y. In this case, the coefficients of the flux equation, a and b, (and not the value of q) are passed to the transport process module.

2.5. Time synchronization

Modules have to be synchronized to calculate state variables and fluxes for simulated times that are the same for all modules. This synchronization is provided for by control modules. The process modules are invoked using a sequential iteration approach (Yeh and Tripathi, 1991). The processes are numerically decoupled from each other and process modules execute sequentially, always in the same order. This design uses a specific sequence of module invocation as shown in Fig. 4. This sequence reflects the method by which transport processes and intra-soil interactions are decoupled during a time increment. For time step calculations, transport processes use values of sources and sinks from the previous, or 'old', time level. Intra-soil and surface interactions may use values of soil state variables from both the current, or 'new' time level or from the old time level. It is for this reason transport calculations precede any calculations of intra-soil interactions (Fig. 4). There is also a specific sequence of transport module invocation. The water transport module must be called first because it produces the information that other transport modules require such as water flow velocities.

Time stepping is synchronized to meet the requirements of all modules, although some modules may not have any limitations on the time step. For example, an equilibrium interphase exchange module can calculate a new equilibrium between solid and liquid phases that results from changes of concentration in solution. The changes can be independent of the time interval during which the changes have occurred. Some modules that use a time step, like water flow, usually make some restrictions on the time step to provide for convergence of non-linear numerical calculations. It is desirable to decrease time steps to enhance convergence of iterations or to increase time step, when the convergence is achieved quickly. The number of iterations required for convergence is used to alter the time step. Further details are given in Pachepsky et al. (1993a,b). Modules that simulate management- or weather-related events, like tillage, require the event to occur at a specific time.

Control modules form the backbone of the generic simulator and are used to oversee the operations of the program. These modules remain the same for any selection of process modules. Control modules manage the spatial discretization of the soil domain (i.e. the area delineated by the x and y coordinates in Fig. 3), types of boundary conditions and locations of boundary nodes, information on soil layering, and calculation of time steps.

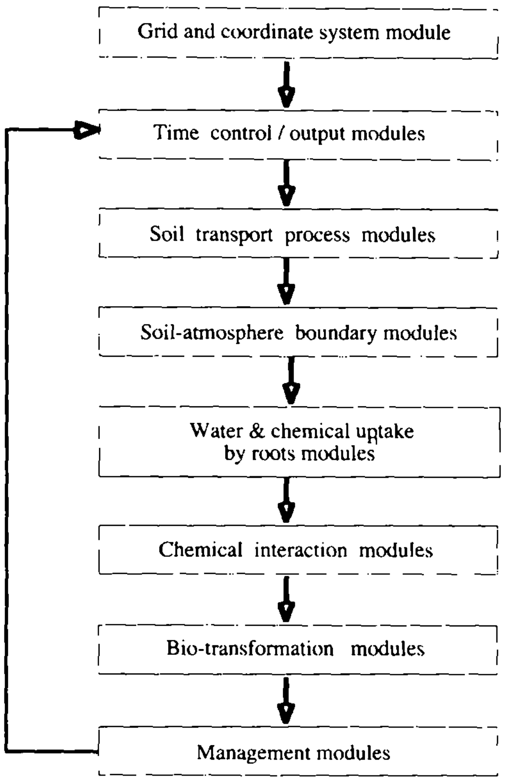
Grid and coordinate system module
Time control / output modules
Soil transport process modules
Soil-atmosphere boundary modules
Water & chemical uptake by roots modules
Chemical interaction modules
Bio-transformation modules
Management modules

Fig. 4. Sequence of operations when the process and control modules are combined to form a simulation model.

2.6. Structure of a process module

At the uppermost level of the hierarchy, global public variables are passed to process modules (subroutines) in Fortran COMMON blocks; no arguments are used in CALL statements for process level modules. 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 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.

Fortran COMMON blocks are also used to store private variables within a module or submodule. The primary reason for using COMMON blocks for private variables is 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.

The general sequence of operations is similar for all process level modules. The operations begin by reading the time-independent data and/or initial data. Next, logic statements are used to determine whether it is time to execute, and if it is time to execute, the module carries out it's designated operations. These include reading the time-dependent data, changing certain public variables and private variables, calculating the requirements for time increments, and writing output data. Some steps may be absent. It also includes the user's interface which, in our case, is the list of input and output variables either in CALL statements or COMMON blocks.

3. Representative example of the simulator-2DSOIL

In this section we present an example of the application of the modular design to build a model of solute movement in row and interrow zones of a soybean crop. The example also illustrates how a model can be expanded to include a management practice through the addition of a module. The objective is to illustrate the influence of root systems on chemical transport. The roots of a crop planted in rows will dry the soil under the rows more rapidly than between the rows by nature of the uneven root distribution. If the soil is more wet between the rows than in the row zone, the wetting front during infiltration will penetrate deeper in the interrow zone than in the row zone. As a result, there will also be higher solute fluxes in the interrow zone than in the row zone.

In order to simulate the processes in this example we needed to assemble a crop model and atmosphere boundary interface with the soil and root process modules. Some modules were adapted from existing (Fortran) code and some developed by us. The adapted modules were rearranged to fit into the data encapsulation structure illustrated in Fig. 1.

A list of modules included in the representative simulator, called 2DSOIL, is presented in Table 1. We used a two-dimensional representation of the soil to model crop row-interrow processes and a finite element representation of the governing equations (Šimůnek et al., 1994). We chose the atmosphere simulator described by Acock and Trent (1991). This simulator derives hourly potential fluxes of water, solutes, and heat at the soil surface from the 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 carbon from an index of soil water availability and plant age. This simplified shoot module can be easily replaced by a comprehensive plant model. Both boundary interfaces are described in Pachepsky et al. (1993b). The global public variables that are modified by process modules in 2DSOIL are shown in Table 2. The information in this table demonstrates that the connection between process modules is loose since each module modifies only a small part of the public data. All modules are thoroughly described in the documentation (Pachepsky et al., 1993b) which is available upon request.

The new chemical application module Mngm is shown in Fig. 5. The structure of the module shown in Fig. 5 corresponds to the general structure for process modules listed and to the data structure of Fig. 1. The module does not have time-independent auxiliary variables nor does it read initial distributions of state variables. The variable Nodes_Where_Applied is private. When in operation, the module changes a public variable (conc), using its private data and other public variables. The addition of this module did not require any changes in other modules of 2DSOIL.

In simulations, the soil consisted of two layers and the soil texture was a sandy loam over a loam. The grid shown in Fig. 3 was used. The initial chemical concentration was zero throughout the profile, and there was no movement of chemical across the left or right boundaries. The bottom boundary was impermeable to both chemical and water. The plant seed was placed at the left end of the grid at $x = 0$ cm, and 5 cm deep. Soybean root parameters were used. Simulated time began at day 1 and ran a total of 57 days. The chemical was a non-reactive tracer similar in properties to bromide. Chemical was applied on day 44 at a rate of 300 mg cm^{-2}. During the preceding 44 days, 18 cm of rainfall was applied and evapotranspiration was 30 cm. This simulates a moderately dry period. Immediately after chemical application 3 cm of water was applied.

The water content and root distribution before chemical application and water distribution after rainfall are shown in Fig. 6. The root distribution is concentrated in the soil under the plant. It can be seen that values of water content are lowest

Table 1
Soil state variables, boundary fluxes, and control variables available to all modules of the 2DSOIL simulator

Variable	Module[a] that modifies the variable
Grid geometry	
Transverse coordinates of nodal points	D
Vertical coordinates of nodal points	D
Numbers of corner nodes for every element	D
Nodal values	
Nodal pressure heads	W
Nodal water contents	W
Nodal concentrations of solutes in soil solution	S
Nodal temperature values	H
Nodal gas contents in soil air	G
Number of soil layer in which nodes occur	D
Element values	
Water extraction rates for elements	R
Solute extraction rates for elements	R
Gas extraction rates for elements	R
Root mass in the elements (soil cells)	R
Boundary values	
Nodal numbers of boundary nodes at seepage faces	D
List of nodal numbers for boundary nodes	D
Width of strips associated with boundary nodes	D
Codes of boundary condition for water movement	D
Same as above for the solute movement	D
Same as above for the heat movement	D
Same as above for the gas movement	D
Boundary pressure heads	
Or components of the water flux equation	A,E
Boundary concentrations	
Or components of the solute flux equation	A,E
Boundary temperatures	
Or components of the heat flux equation	A,E
Boundary gas contents	
Or components of the gas flux equation	A,E
Shoot-related values	
Potential and actual transpiration	P
Carbon available for the root growth	P
Nutrients supplied for the plant grows	R
Time control values	
Switch to show if initial data have to be read	T
Time of the beginning of calculations	T
Current value of the time step end time	T
Current time step	T,W
Time of the next event (soil-atmosphere boundary update, time-dependent boundary_ conditions update, root extraction update, output of results)	A,E,O,P,M
Maximum next time steps allowed by transport modules	W,S,H,G
Time of the end of calculations	T

[a] A, atmosphere module; B, Biotransformation; C, chemical transformation module; D, discretization module; E, engineered boundaries module; G, gas transport module; H, heat transport module; M, management modules; O, output module; P, plant shoot module; R, root uptake module; S, solute transport module; T, time control module; W, water transport module

under the plant where the root concentrations are high. Water content increases with depth and horizontal distance from the plant. After rainfall the water contents are still highest in the interrow zone (Fig. 6). The chemical concentration directly below the plant row (at $x = 0$) and under the interrow position ($x = 35$) at day 56 are shown in Fig. 7. There is more chemical in the soil under the row position than the interrow position. Furthermore, the chemical concentrations near the surface are still rather high in spite of two rainfall events after chemical application. Evaporation and transpiration act to move the water with chemical towards the soil surface and towards the plant. This results in accumulation of chemical in the soil under the plant and depletion of chemical in the interrow zone. These results are in qualitative agreement with the results of Timlin et al., 1992.

Table 2
Process modules used in the example of the 2DSOIL simulator

Module (module name)	Source
Water transport (WaterMover)	
1. Finite-element solution of the Richards' equation for two-dimensional Darcian water flow in variably saturated soil	Simünek et al., 1994
2. Finite-difference solution of the Richards' equation for one-dimensional Darcian flow in variably saturated soil	Scherbakov et al., 1981
Solute transport (SoluteMover)	
1. Finite-element solution of the convective-dispersive equation for two-dimensional transport of several solutes in variably saturated soil	Simünek et al., 1994; Istok, 1989;
2. Finite-difference solution of the convective-dispersive equation for one-dimensional transport of several solutes in variably saturated soil	Pachepsky, 1990
Heat transport (HeatMover)	
1. Finite-element solution of the convective-dispersive equation for two-dimensional transport of the heat in variably saturated soil	Simünek et al., 1994
Gas transport (GasMover)	
1. Finite-element solution of the molecular diffusion equation for two-dimensional gas transport in air-filled pore space	Pachepsky et al., 1993b
Root water uptake (RootUptake)	
1. Root water uptake based on the trade of carbon and water between shoot and root. Root growth dependent on soil water and oxygen contents, soil temperature and root-shoot carbon balance	Acock and Trent, 1991
2. Water uptake by a permanent root system based on the partitioning of the transpiration demand between roots in soil cells according the root mass, and calculating the actual uptake as a fraction of the potential uptake depending on soil-water potential	Wesseling and Brandyk, 1985
Macroelement equilibrium chemistry (MacroChem)	
1. Cation exchange Ca-Mg-Na, dissolution precipitation of gypsum and carbonates, speciation in solution and dissociation of carbonic acid and water	Pachepsky, 1990
Nitrogen transformation (NitroChem)	
1. Mineralization/immobilization of organic matter in soil. Nitrification and denitrification.	Bergström et al., 1991

4. Conclusions

The support of code extension and code reuse is a major advantage of the modular structure and application, 2DSOIL, described here. Since each module manages its own data, and the functions and data are kept together within the module, the design makes the modules highly independent. A library of modules can be easily accumulated. Several water transport and solute transport modules listed in Table 1 represent an initial step in this direction. Since the data only have to be prepared for modules for a particular application, it also reduces the effort needed to compile and prepare data sets. As in the example, 2DSOIL, shown here, it is very easy to add new modules for specific cases using the modular structure developed in this paper.

The modular design presented here was mainly developed as a framework for crop modellers to interface their plant and atmosphere codes with reliable soil code. To do this, the crop modeller needs to concentrate only on the boundary interface. He or she must: (a) assign potential boundary fluxes of water, solutes, heat and gases from their atmosphere module to the nodal boundary fluxes; (b) receive actual boundary flux values from the soil transport process modules to use them as needed; (c) pass carbon flux values from their shoot module to the root module; (d) receive actual transpiration, water and nutrients fluxes from the root module and to use them as

Subroutine Mngm ( )

Include 'Public.ins'	Global public variables are in 'Public.ins'
Common /MNG/ Application_time, & Chemical_Mass_in_a_Node_After_Application, & Number_of_Nodes_Where_Applied, & Nodes_Where_Applied(Max_Number_of_Boundary_nodes), & Number_of_This_Module	Private information is declared and stored in the **COMMON** field **MNG**
If (lInput.eq.1) then Open (40, file='Param_M.dat') Read (40,*,Err=20) Application_time, & Chemical_Mass_in_a_Node_After_Application, & Number_of_Nodes_Where Applied Read (40,*,Err=20) & (Nodes_Where_Applied(n), n= 1,Number_of_Nodes_Where_Applied) Close(40)	Initialization of the private information; **lInput** is the global public variable which is equal to 1 until all modules and submodules are invoked one time.

Global_Number=Global_Number+1
Number_of_This_Module=Global_Number
	The module gets its sequence number in the current configuration and keeps it as private information. **Global_Number** is the global counter of modules

tEvent(Number_of_This_Module)=Application_Time Endif	tEvent is the global array of times when events require to end a time step. **tEvent** of this module is set to application time.
If (Abs(Time-tEvent(11)).LT.0.1*Step) then	Check whether it is time to operate using simulated time Time and current time increment, **Step.**
Do n=1,Number_of_Nodes_Where_Applied Conc(Nodes_Where_Applied(n),1)= & Chemical_Mass_in_a_Node_After_Application/ & Water_Content(Nodes_Where_Applied(n)) Enddo	The module changes public global variable **Conc** which represents nodal concentrations of chemicals; second subscript shows that chemical #1 is applied. To calculate concentrations, nodal mass is divided by the nodal water content, represented by the public variable **Water_Content**.
tEvent(Number_of_This_Module)=1.0E+32 Endif Return 20 Stop 'Management data error' End	The next time to for this module to execute instructions is set to 1.0E+32 because the module will no longer be needed and therefore will not impose requirements on time steps anymore

Fig. 5. Computer code that illustrates the sequence of operations in a process module. This example is a management module to simulate an application of instantaneously soluble chemical such as a fertilizer.

needed; and (e) provide simulated times when the shoot and atmosphere simulators will be ready to exchange information. All variables needed for this exchange are global public, and no code has to be changed in other modules. If some other soil variables are used by the shoot and atmosphere simulator, they are accessible through the global COMMON block since all global public data for the boundary interface are available here. 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 modeller wants to use his or her own root module, the module must be rearranged to fit into the data structure described here.

The example simulator demonstrated in this paper, 2DSOIL, illustrates code reuse. We used significant pieces of code written by others and were able to combine them as necessary for a particular task. The modularity was a necessary precondition of this and we took full advantage of the fact that the codes we used were well-tested, robust and secure: necessary features for code that is to be reused (Radice and Philips, 1992).

The proposed three-level architecture of the generic simulator design (Figs. 1 and 5) represents a compromise between a modeller's and a software developer's views on modularity. The most fundamental criteria in module selection is that each module should be testable (Maynard, 1972). We could satisfy this criteria by generating public information needed by a module with the help of specially written 'stubs' or fictitious modules.

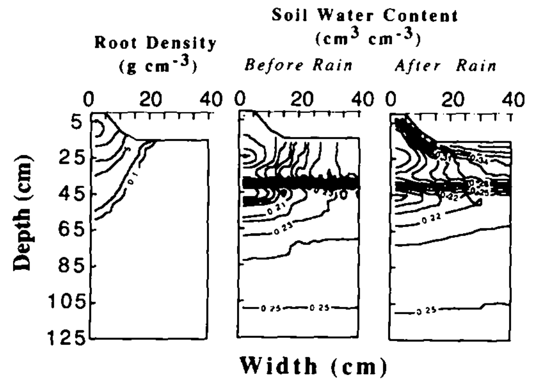

Fig. 6. Vertical and horizontal distributions of simulated root density and soil water content before and after a chemical application followed by rain.

The design of the generic simulator and its current implementation, 2DSOIL, has several disadvantages, however. The modules in 2DSOIL belong to the category of 'external coupled modules', i.e. they reference a global public data block (Einbu, 1991). Global data areas used as a communication between modules are known to be a source of errors (Einbu, 1991). However, the present arrangement together with numerical decoupling to allow each process module to be executed in sequence makes the connection among modules loose. Therefore the public field is relatively small, and the data are mostly encapsulated. If the use of public data by a module is thoroughly documented in its specification, and connections between modules are loose as they are in this design, the danger of errors is reduced.

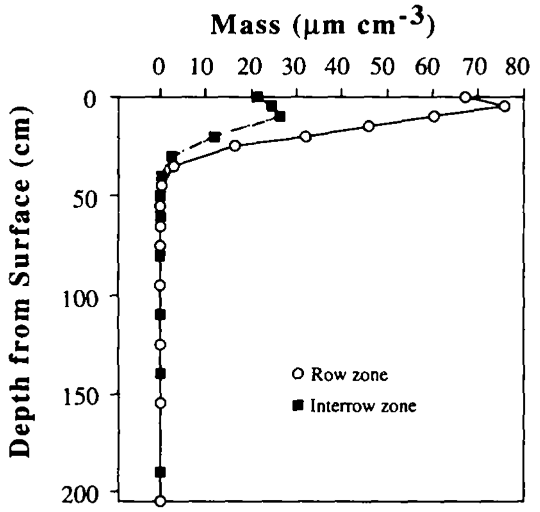

Fig. 7. Simulated mass of chemical in the upper 2 m of soil as a function of soil depth in the row zone ($x = 0.01$ m) and in the interrow zone ($x = 0.35$ m) after a chemical application followed by rain.

In our example, numerical decoupling of interrelated processes (e.g. example solute transport from water flow) did not present a problem since the only strong non-linearity in time dependent modules was encountered in the water transport module. However, in general, it may be necessary to iterate over all equations if several nonlinear equations with explicit dependence on time step will be involved. This problem has to be studied in future.

In the application of the modular design, 2DSOIL, the modules were organized around processes primarily because the structures of the existing programs that were incorporated into 2DSOIL were already based on processes, and this kind of structure was suited to Fortran. A modular soil simulator could also be organized on the level of a soil cell (Dúbois-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 OOP language such as Smalltalk 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, for example cells with macropores, or boundary cells. However, the need for variable time steps and iterations may preclude a consistent implementation of OOP. This problem was discussed by Dúbois-Pelerin et al. (1992) who could not achieve a fully OOP-consistent structure in their pilot attempt of finite-element programming in Smalltalk because of the necessity to iterate within a time step. This problem is also discussed by Acock and Reddy (1997) and Lemmon and Chuk (1997).

The process modules in our simulator are non-hierarchical relative to one another. The processes can be grouped in other arrangements and then be partitioned recursively into objects. Each object is then internally partitioned into lower level objects (for example a transport process module that contains solute transport and water flow as submodules). The result will be a multilevel hierarchy. There is an opinion, however, that a single level of objects is adequate for most purposes (Kirk, 1990). Our experience with 2DSOIL shows that this is true for a generic soil and root process simulator since the cohesion within processes is relatively stronger than between processes. Our objective was to design a simulator that could be expanded and modified by users. Therefore we gave the priority to information hiding at the expense of hierarchy development. As a result, a user can work with modules without detailed knowledge of their internal structure.

Since the coding of soil models has been done mostly in Fortran we used this language for the development of modules. Fortran 90, however, includes some OOP capabilities. As mentioned by Dúbois-Pelerin et al. (1992), the really successful implementation of OOP can be achieved by 'expanding the existing OOP environment', i.e. using existing classes and objects. The extension of the OOP environment of Fortran 90 can be a promising way of embedding existing computationally efficient and well-tested codes into an OOP simulator of soil and root processes.

Developers of agricultural and ecosystem models generally think of their models as integrated scientific knowledge and rarely consider the coded model as software. Since models are software, and as the number of models grows rapidly, software engineering issues are becoming more and more important in modelling. Modularity is one such issue. Our experience with 2DSOIL convinced us that more efficient simulators can be built for particular objectives if recommendations and methods of software engineering are used. In current and future applications of this modular structure we hope to demonstrate that modularity enhances the usefulness of models as scientific tools because it simplifies experimentation with different concepts. Current applications include modelling competition between two root systems (Caldwell et al., 1994), modelling evapotranspiration in a desert environment (Kemp et al., 1997) and the incorporation of the potato model, SIMPOTATO (Hodges et al., 1992) into 2DSOIL. We also hope that the modularity as illustrated by 2DSOIL will widen the potential ring of soil model users by making a particular process modeller independent of specialists in other processes.

References

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

Acock, B. and Reynolds, J.F., 1989. The rationale for adopting a modular generic structure for crop simulators. Acta Hortic., 248: 391-396.

Acock, B. and Reynolds, J.F., 1990. Model structure and database development. In: R.K. Dixon, R.S. Meldahl, G.A. Ruark and W.G. Warren (Editors), Process Modelling of Forest Growth Responses to Environmental Stress. Timber Press, Portland, Oregon, pp. 169-179.

Acock, B. and Trent, A., 1991. The soybean simulator, GLYCIM. Documentation for the modular version 91. Agric. Exp. Stn. Univ. Idaho, Moscow, ID.

Ahuja, L.R., Decoursey, D.G., Barnes, B.B. and Rojas, K.W., 1991. Characteristics and importance of preferential macropore transport studied with the ARS root zone water quality models. Proc. Nat. Symp. Preferential Flow. Amer. Soc. Agric. Eng. Publ. 9. Chicago, IL, pp. 32 42.

Baker, D.N., Lambert, J.R. and McKinion, J.M., 1983. GOSSYM: A simulator of cotton crop growth and yield. S.C. Agric. Exp. Sta. Tech. Bull., 1089: 1-134.

Bergström, L., Johnson, H. and Torstensson, G., 1991. Simulation of soil nitrogen dynamics using the SOILN model. In: J.J.R. Groot, P. de Willigen and E.L.J. Verberne (Editors), Nitrogen turnover in the soil-crop system. Cluver Academic Publishers, Dordrecht, pp. 181-188.

Blum, B.I., 1992. Software Engineering: A Holistic View. Oxford Univ. Press, New York, 588 pp.

Caldwell, R., Pachepsky, Y. and Timlin, D., 1994. Current research status on growth modelling in intercropping. Proc. Workshop on Nitrogen Dynamics of Intercropping Systems in the Semi-arid Tropics. Delhi, India.

Dúbois-Pelerin, Y., Zimmerman, T. and Bomme, P., 1992. Object oriented finite element programming. II. A prototype program in Smalltalk. Comput. Methods Appl. Mech. Eng., 98: 361-397.

Einbu, J., 1991. A program architecture for improved maintainability in software engineering. Ellis Hoorwood, Chichester, UK, 166 pp.

Hodges, T., Johnson, S.L. and Johnson, B.S., 1992. A modular structure for crop simulation models: Implementation in the SIMPOTATO model. Agron. J., 84: 911-915.

Hutson, J.T. and Wagenet, R.J., 1992. LEACHM: Leaching Estimation And Chemistry Model: A process based model of water and solute movement, transformation, plant uptake and chemical reactions in unsaturated zone, Version 3. Department of Agronomy, Cornell University, Ithaca, NY. 89 pp.

Istok, J., 1989. Groundwater modeling by the finite element method. Water Resources Monograph 13, American Geophysical Union, Washington, D.C.

Kemp, P.R., Reynolds, J.F., Pachepsky, Y. and Chen, J.-L., 1997. A comparative modeling study of soil water dynamics in a desert ecosystem. Water Resour. Res. (in press).

Kirk, B.R., 1990. Designing Systems with Objects, Processes, and Modules. SE90: Proc. Software Engineering, 90. Cambridge Univ. Press, Brighton, UK, pp. 387–404.

Lemmon, H. and Chuk, N., 1997. Object-oriented design of a cotton crop model. Ecol. Model., 94: 45–51.

Maynard, J., 1972. Modular programming. Petrocelli Books, New York. 100 pp.

Mills, H.D., 1980. Principles of software engineering. IBM Syst. J., 19: 415–420.

Pachepsky, Y., Acock, B., Lemmon, H., Timlin, D. and Trent, A., 1993a. 2DSOIL-a new modular simulator to assess the influence of management practices on water quality. Proc. Agricultural Research to Protect Water Quality, Minneapolis, MN.

Pachepsky, Y., Timlin, D., Acock, B., Lemmon, H. and Trent, A., 1993b. 2DSOIL-a new modular simulator of soil and root processes. Release 02, US Deptartment of Agriculture, Beltsville, MD.

Pachepsky, Y.A., 1990. Physico-chemical models in soil science. Nauka, Moscow. 187 pp.

Parnas, D., 1972. On the criteria to be used in decomposing systems into modules. Commun. ACM, 15: 1053–1058.

Radice, R.A. and Philips, R.W., 1992. Software Engineering: An Industrial Approach. Prentice Hall, Englewood Cliffs, NJ. 315 pp.

Reynolds, J.F. and Acock, B., 1985. Predicting the response of plants to increasing carbon dioxide: A critique of plant growth models. Ecol. Model., 29: 107–129.

Reynolds, J.F. and Acock, B., 1997. Modularity and genericness in plant and ecosystem models. Ecol. Model., 94: 7–16.

Reynolds, J.F., Acock, B., Dougherty, R. and Tenhunen, J.D., 1989. A modular structure for plant growth simulation models. In: J.S. Pereira and J.J. Landsberg (Editors), Biomass Production by Fast-Growing Trees. NATO ASI Series, Applied Science, Kluwer Academic Publishers, Boston, MA, pp. 135–168.

Reynolds, J.F., Acock, B. and Whitney, R., 1993. Linking CO_2 experiments and modeling. In: E.-D. Schulze and H.A. Mooney (Editors), Design and Execution of Experiments on CO_2 Enrichment. Report No. 6, Ecosystems Research Series of Environmental Research Programme, Commission of the European Communities, Brussels, pp. 93–106.

Scherbakov, R.A., Pachepsky, Y.A. and Kuznetsov, M.Y., 1981. Transport of ions and chemicals in soils: Water movement. Ecomodel software series, No 7, USSR Academy of Sciences, Pushchino, 47 pp.

Sĭmünek, J., Vogel, T. and van Genuchten, M.T., 1994. The SWMS_2D code for simulating water flow and solute transport in two-dimensional variably saturated media-Version 1.2. Research Report no. 132, U.S. Salinity Laboratory, USDA-ARS, Riverside, CA.

Thornley, J.H.M. and Johnson, I.R., 1990. Plant and Crop Modelling. Clarendon Press, Oxford. 669 pp.

Timlin, D.J., Heathman, G.C. and Ahuja, L.R., 1992. Solute leaching in row vs. interrow zones. Soil Sci. Soc. Am. J., 56: 384–392.

Wesseling, J.E. and Brandyk, T., 1985. Introduction of the occurrence of high groundwater level and surface water storage in computer program SWATRE. Note 1636, Institute for Land Water Management Research, Wageningen, The Netherlands. 68 pp.

Wirfs-Brock, R., Wilkerson, B. and Weiner, L., 1990. Designing Object Oriented Software. Prentice Hall, Englewood Cliffs, N.J. 341 pp.

Witt, B.I., Baker, F.T. and Meritt, E.W., 1994. Software architecture and design. Principles, Models, and Methods. Van Nostrand Reinhold, New York, 324 pp.

Yeh, G.T. and Tripathi, V.S., 1991. A model for simulating transport of reactive multi-species components: Model development and demonstration. Water Resour. Res., 27: 3075–3094.