LWRRDC Project CPI 9

Linking GRAZPLAN pasture/animal models to APSIM crop/soil models and the SWIM water balance

AD Moore, MJ Robertson, DP Holzworth,
NI Herrmann & NI Huth

Contents

Summary

A “common modelling protocol” has been developed by CSIRO Plant Industry (CPI) and the Agricultural Production Systems Research Unit (APSRU). The protocol is a software technology that enables models developed by different groups to be inter-connected. It is not itself a model, but is a framework, or set of rules, for building simulation models in a modular fashion.

The design of the protocol is outlined (the detailed specification is given in a separate report), and its implementation as software by APSRU and CPI discussed. A number of potential problems arise when models from different modelling groups are to be used together within the protocol; these issues are discussed and strategies for solving them are presented.

Several models of particular interest to Land and Water Australia's Redesigning Agriculture for Australian Landscapes Program have been implemented within the protocol. These models have been used together in sample simulations as a demonstration of project completion and to show some of the new capacities provided by using the protocol.

1. Introduction

Many Australian agricultural production systems are not ecologically sustainable because they are not able to make full use of available rain or soil moisture. As a consequence water containing salt and nutrients leaks from these systems. Drainage and nutrient loss are primary causes of dryland salinity, soil acidification and nutrient exports to watercourses. Land and Water Australia's Redesigning Agriculture for Australian Landscapes Program (RAAL) aims to compare current agricultural systems with the native plant communities that they have replaced, with four objectives in mind (Clarke 2000):

1. To understand by comparison, the key biophysical processes affecting leakage of water and nutrients in cropping, grazing and natural systems.
2. To benchmark criteria for redesigning agricultural systems in Australian landscapes.
3. To develop a toolbox of redesign options to modify current, or develop new, agricultural systems for Australian landscapes.
4. To facilitate implementation of redesign options in priority Australian landscapes by exploring the socio-economic, institutional, policy, marketing and technological requirements and implications of each option.

Meeting the first three of these objectives requires the use of simulation models to assess the benefits of alternative landscape design options. The high cost of mapping, monitoring and measurements mean that it is not feasible to have intensive field studies in every catchment; nor is it feasible to simply extrapolate results from a nearby area without some modelling framework. Also, the variability with time of different fluxes means that short-term field studies are unlikely to provide reliable estimates of fluxes without linkage with long-term climatic data via a model.

The key physical and biological processes have all been modelled. For example, the crop, tree and soil models developed by the Agricultural Production Systems Research Unit (APSRU; Keating *et al.* 2002), together with the SWIM soil water balance (Verburg *et al.* 1996) and GRAZPLAN pasture and ruminant biology models (Freer *et al.* 1997, Moore *et al.* 1997) developed by CSIRO, cover the main processes required to study the leakage of water and nutrients in cropping, grazing and natural systems. Until now, however, these models could only be put together in *ad hoc* ways (e.g. the study described by Stauffacher *et al.* 2000). As a result attempts to compare grazing and cropping land uses have been confounded by either the use of different models for a single process such as water flows, or the omission of potentially important feedbacks.

The RAAL program exemplifies a trend in agricultural and ecological modelling toward the analysis of more complex problems, the analysis of which requires a wider range of expertise than is available "in-house" in any one modelling group. There is therefore a need, both within and beyond the RAAL program, to develop technologies that enable models from different groups to be inter-connected. CSIRO has identified this need and has been working, with support from LWA, to resolve it through the development of a "common modelling protocol".

This report describes the design and implementation of the protocol by CSIRO Plant Industry (CPI) and APSRU. It presents the results of some example simulations that show the GRAZPLAN pasture and animal models, the SWIM water balance and APSRU's wheat model operating together to describe more complex agricultural systems than have been possible in Australia.

2. The common modelling protocol

The common modelling protocol is not itself a model; it is a framework, or set of rules, for building simulation models in a modular fashion. The protocol describes specific methods for carrying out the tasks involved in computing a simulation (initialization, integration, transfers of information, error handling etc). If a model developer implements an interface to their model that obeys the protocol, it can then be coupled to models of related processes, including models constructed by other organizations.

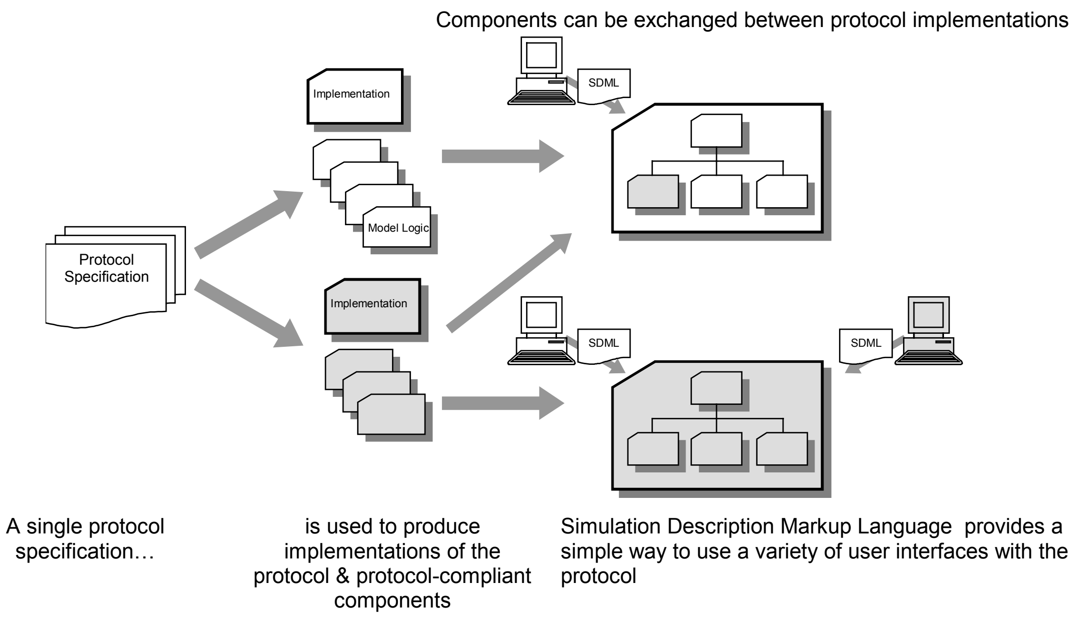

Components can be exchanged between protocol implementations
Implementation
SDML
Model Logic
Protocol Specification
Implementation
SDML
SDML
A single protocol specification…
is used to produce implementations of the protocol & protocol-compliant components
Simulation Description Markup Language provides a simple way to use a variety of user interfaces with the protocol

Figure 2.1. Using the common modelling protocol to develop flexible and interchangeable simulation software

2.1. Key requirements for a common modelling protocol

Simulation modelling in agriculture and resource management has now been under way for over 30 years (Arcus 1963; Brouwer & de Wit 1968). The desirability of generic simulation tools was recognized very early (e.g. Beek & Frissel 1973). As the technology of computing and the art of simulation have developed, the following attributes have been recognized as desirable in a generic agricultural simulation tool, and have guided the development of the common modelling protocol:

Modularity. Separating the parts of a model that are closely related into sub-models has advantages both in maintaining the computer software that implements the model, and in allowing scientists in research teams to specialize in modelling one part of the larger system (McCown *et al.* 1996). If modularity is implemented at the level of executable modules (e.g. dynamic link libraries in Windows), it also permits model developers to provide up-to-date versions of their models to other users while retaining control over model design.

Configurability. Once a simulation model is decomposed into sub-models, it becomes natural to arrange the sub-models in different configurations to reflect a range of different real-world situations (McCown *et al.* 1996). The practical advantage is that it becomes possible to use the simplest model required to analyze a problem. It also becomes possible to use the same model code more than once within a simulation (for example to represent the flows of soil water in each of several paddocks).

Interchangeability. Modular construction also permits the substitution of one representation of a process by another, depending on the needs of the modeller. This can be useful in comparing different representations of a process, or in configuring a simulation for efficient execution. In a well-designed modelling framework, interchangeability of models describing the same or similar processes will be limited only by the “science” of the models (i.e. the model developers’ working hypotheses as to how reality works), and not by the technology used to implement the science.

Interchangeability applies not only to sub-models, but also to modelling software. Ideally it should be possible to use the software implementing a model in conjunction with a range of different user interfaces for different purposes (e.g. Donnelly *et al.* 1997).

Representation of both continuous and discrete processes. Many processes in agricultural systems are fundamentally continuous in nature. Others, particularly management interventions, involve sharp changes in the state of the system, which may be thought of as instantaneous "events". Event-based representations of management have a long pedigree (e.g. Christian *et al.* 1978). To adequately capture key features of agricultural systems, a simulation tool must be able to accommodate both continuous and discontinuous processes.

Hierarchical structure. Ecological and hence agronomic systems are *medium-number* systems. They contain too many entities to be treated as *small-number* systems that can be solved by differential-equation techniques; and they have too few entities to be treated as *large-number* systems that are amenable to treatment as statistical assemblages. Current ecological theory suggests that the best way to analyze this kind of complexity is to take advantage of *organization* in these systems that arises from differences in the rates of different processes. This organization leads naturally to representations of reality that are hierarchically structured (O'Neill *et al.* 1986).

2.2. Dynamic models and simulations

Before describing the protocol, it is important to define the kind of models – "simulation" models – that it is intended to support. Our definition of a simulation model reflects the desiderata given in section 2.1.

A *simulation*[1] is a computation of a dynamic model between given start and end times, i.e. it is an integration over time. A *dynamic model* is defined by a set of equations; the equations of a given dynamic model may fall into natural groupings known as *submodels*. Some of these submodels may have equations and quantities of identical form, i.e. they belong to the same "class" of submodel. Each submodel is composed of a set of quantities, a set of rate equations, and a set of events.

All *quantities* can be expressed as real numbers, integers, or Boolean (true/false) values. Real-valued quantities have dimension and units; the units must conform to the dimension. A quantity is called a "state variable" if it may vary through time and its initial value must be known in order to compute the simulation. A quantity is called a "driving variable" if its value is stored externally to its submodel.

Changes in the values of state variables are computed according to the rate equations and events. Each real-valued state variable has a *rate equation* associated with it. The rate equation is an ordinary differential equation that gives the rate of change of that state variable over time.

An *event*[2] in a dynamic model is defined by
- a set of equations specifying an instantaneous change in one or more state variables; and
- a "trigger", which is a logical relation that, if satisfied at any time, causes the change(s) in state variables.

A simulation is therefore completely defined by:
- the model, i.e. the set of submodels it contains;
- the start and end times for the computation;
- the values of the state variables of each submodel at the start time; and
- the time course of the model's driving variables.

2.3. Design of the protocol

This section contains a relatively brief description of the common modelling protocol. A complete treatment is given in the protocol specification, which is included as an Appendix to this report.

2.3.1 Components, properties and events

In the common modelling protocol, simulations are constructed out of modular elements called **components**. Each sub-model of the dynamic model embodied by the simulation is represented by a component. Components may also perform tasks such as obtaining the values of the model's driving variables, controlling the time integration or storing outputs. Some components group other related components together; such components are called *systems*. The systems and components of a simulation are arranged in a hierarchy or tree, with a "simulation system" at its root.

The interface between a component and the rest of a simulation is made up of an identifying name; a number of properties; and zero or more event handlers.

1 No consistent terminology exists within the discipline of simulation modelling. The definitions used here are in relatively common use.

2 The term "event" is used in another sense within the modelling protocol. Model events will be represented by protocol events, but so will other parts of the computation. Unfortunately no good alternative term exists.

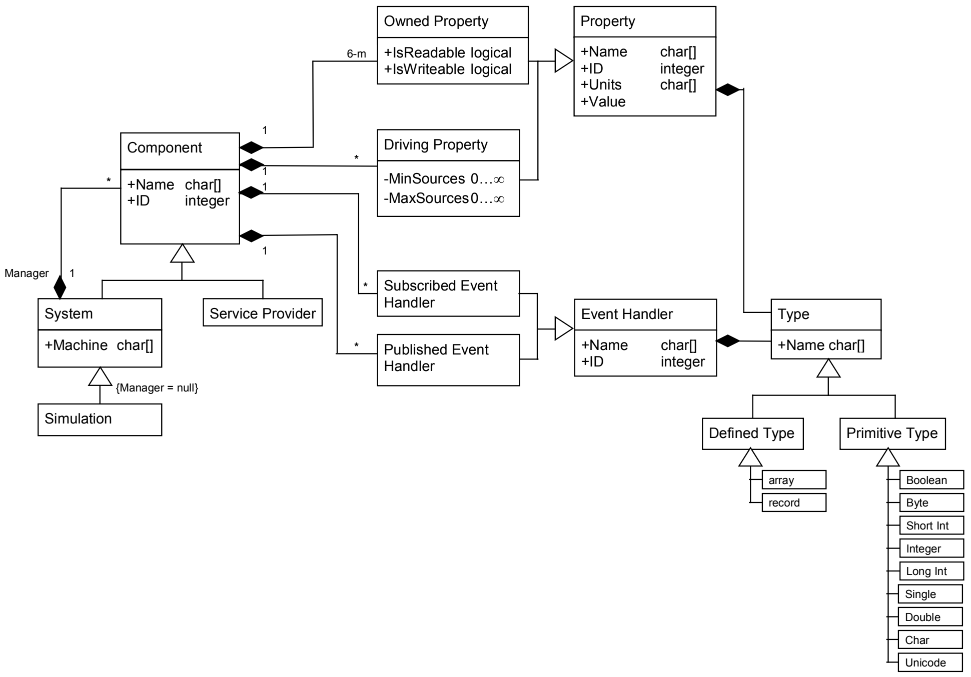


Figure 2.2. Class diagram describing the relationship between entities in the common modelling protocol. The diagram uses the conventions of a standard software design tool known as the Universal Modelling Language (Object Management Group 2001).

Properties correspond to the quantities in a dynamic model. Each property is defined by a name; a type; and a value. The type of a property determines the set of values it may take and the units of those values, where applicable. The type must be either one of a set of *primitive types* (see Figure 2.2) or else an array or record structure ultimately composed of these primitive data types. The value must conform to the type; for example an integer-valued property cannot take the value 2.5.

Two different kinds of properties are distinguished. *Driving properties* correspond to driving variables, i.e. quantities which are stored externally to a given component but which must be known in order to compute the component's logic. All other properties (i.e. those that are stored by the component) are known as *owned properties*. The systems in a simulation work together to ensure that the values of each component's driving properties are read from the properties of other components as needed.

Events in the common modelling protocol are used to signal the occurrence of activities (i.e. computations) and to pass instructions between components. Protocol "events" are therefore used to drive the computation of both continuous processes and logical "events" (discontinuous processes). Every event has a name; it may also have a type and data that conforms to the type. When an event is passed to a component, an *event handler* is executed.

Simulations are configured by specifying which components form the simulation and their arrangement into a hierarchy. This is done using Simulation Description Markup Language (SDML), a language defined using Extensible Markup Language (XML; World Wide Web Consortium 2000). A valid SDML document also contains enough information to specify the initial values of all state variables of a model, so that it completely specifies a simulation. The use of SDML to define individual simulations provides two important benefits:

- An SDML document provides a record of exactly how a given simulation was carried out, using a standard format. This is of use when commercial or legal considerations make the reproducibility and defensibility of a simulation analysis important.
- Protocol-compliant simulations are carried out by constructing an SDML document and submitting it to the protocol implementation for execution. Consequently a single simulation "engine" can be used readily with a variety of different user interfaces; all the interface software needs to do is to write the appropriate SDML documents.

The protocol achieves modularity through the use of components; configurability through the use of SDML; and hierarchical structure through the use of nested systems. Event handlers can be used to implement either continuous or discontinuous processes. Interchangeability (at the technical level) has been achieved by specifying rules that must be followed by protocol implementations; see section 3.1.

2.3.2 Low-level design of the protocol

In a simulation that uses the protocol, all information transfers are made by using *messages*. The protocol defined a set of 31 distinct messages. Every message has a common header that contains the type of the message (as a numeric index), the component from which it originated and the component to which it is to be sent. Most of the messages also contain further data, which are laid out in a specified fashion so that any protocol-compliant component can read them.

The protocol specifies the circumstances in which messages may be sent by a component, the component or components to which it may send each message, and the behaviour required of the receiving component (which may be to carry out some internal computation, or to send one or more messages, or both). Defined sequences of messages are used to carry out the various tasks involved in carrying out a simulation, such as:

- initializing the simulation;
- terminating the simulation;
- integrating the model over a time step;
- transmitting the current value(s) of a driving property;
- changing the value of another component's owned property;
- transmitting a protocol event; or
- recording the current model state.

The protocol is specified so as to permit an implementation to carry out asynchronous computations, thereby making possible the execution of parts of a single simulation on different computers. Every component, property and event handler is assigned an identifying number, which is used in place of its name within messages. Type information in messages is expressed using a Data Description Markup Language (DDML), which is also based on XML[3].

Table 2.1. The set of messages defined by the protocol.
For more detail, see the protocol specification, which is included as an Appendix to this report.

Message Name	Sent by	Received by	Acknowledge Completion
activateComponent	Component	System managing component	Optional
addComponent	Component	System	Optional
checkpoint	Component	Component	Futile
commence	Simulation system	Simulation system (sequencer)	Optional
complete	Component	Component	Never
deactivateComponent	Component	System managing component	Optional
deleteComponent	Component	System managing component	Optional
deregister	Component	System managing component	Optional
event	Component	Component	Optional
getValue	Component	System managing component	Mandatory
init1	System	Component managed by system	Mandatory
init2	System	Component managed by system	Mandatory
notifyAboutToDelete	System	Component managed by system	Mandatory
notifyRegistrationChange	System	Systems	Optional
notifySetValueSuccess	Component	Component	Optional
notifyTermination	Simulation system, system	System, component	Mandatory
pauseSimulation	Component	Simulation system (sequencer)	Optional
publishEvent	Component	System managing component	Optional
queryInfo	Component	System	Mandatory
queryType	Component	Simulation system	Futile
queryValue	System	Component	Futile
reinstateCheckpoint	Component	Component	Optional
register	Component	System managing component	Optional
requestComponentID	System	Simulation system	Futile
requestSetValue	Component	Component	Futile
resumeSimulation	Component	Simulation system (sequencer)	Optional
returnComponentID	Simulation system	Component	Optional
returnInfo	System	Component	Optional
returnType	Simulation system	Component	Optional
returnValue	Component	Component	Optional
terminateSimulation	Component	Simulation system	Never

[3] For detailed information about DDML, see section 6 of the protocol specification.

3. Implementing the protocol

APSRU and CPI have implemented the protocol independently. Separate implementations offer benefits to the project partners:

- intellectual property issues related to distribution of protocol implementations are avoided,
- maintenance of the computer code is more straightforward, and
- it has been possible to adapt existing model interfaces into the protocol, rather than having to write the interfaces to models from scratch.

By building separate implementations, we have also shown that it is possible for other parties to implement the protocol for themselves, thereby maintaining independence of CSIRO software while still making use of the collaborative benefits that the protocol offers.

3.1. Implementation technologies

3.1.1. Implementation under Windows

Protocol-compliant components within Windows must be implemented as dynamic link libraries (DLLs), using a common set of exported routines. The common interface assumes that a "wrapper" DLL interposes between the simulation implementation and the component DLL proper (Figure 3.1). In all protocol implementations, information and instructions to components must be conveyed by passing a message to the wrapper DLL; the wrapper then passes the message contents to the component DLL by implementation-specific means. A component DLL may act as its own wrapper (and does so in the APSRU implementation).

In addition to the message-passing interface, a Windows DLL containing component must export two routines that provide a description of the component (in a standard, XML-based format); another set of routines that allow a calling program to generate SDML that can be read by the component to specify its initial values; and a final routine that gives the name of the wrapper DLL.[4]

The result is that any Windows protocol implementation is able to load and execute any protocol-compliant Windows component. It is possible to implement the protocol so as to have parts of the simulation execute on different computers across a network.

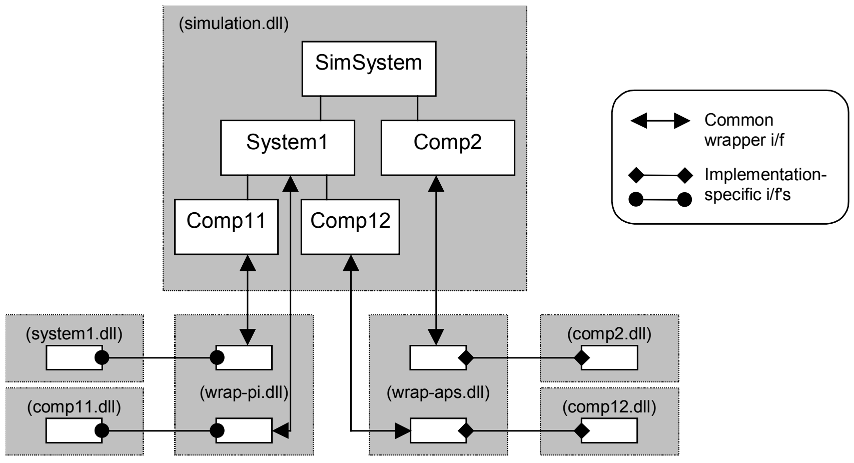


Figure 3.1. Component DLLs and their wrappers interacting with the protocol.

3.1.2. The APSRU implementation

The APSRU implementation of the common modeling protocol has evolved steadily over the last couple of years. Each release of APSIM has built on previous releases by further supporting the protocol. APSIM version 2 (released Feb 2001) was partially protocol compliant. It contained the following protocol features:

- it used an early version of SDML as the input file format;

[4] For detailed information about these exported routines, see section 10.1 of the protocol specification.

- several of the messages types (e.g. initialization, termination) were close to protocol-compliant;
- multiple instantiation of APSIM modules was possible, allowing for multi-point simulations. This was the first version of APSIM to have this capability.

At the time of this report, APSIM complies with most of the protocol specification. The two exceptions are checkpointing and dynamically modifying the simulation structure at run-time. These two protocol features are not required to successfully run components developed by CPI in APSIM, and as such are scheduled for future implementation. The version of APSIM developed as part of this project will be released as version 3 around the middle of 2002.

Each component in APSIM is implemented as a single DLL. This DLL is split into two functional parts: component interface (written in a C++) and a component "science" part. The component interface provides the necessary code to communicate with rest of the simulation in a protocol-compliant way. It also provides an "applications programming interface"; this hides the details of the protocol, so that the developer of the science part is free to focus on the simulation science without the complexity of the protocol getting in the way. The science part for most APSIM modules is written in FORTRAN; the remainder are written in C++.

The APSRU protocol implementation is coded as a "protocol manager", which has been coded as a normal component. In essence, the protocol manager is a router of messages and information from one component to another. It resides in its own DLL and has a component interface and a science part. The "science" part of the protocol manager component (written in C++) maintains registration information and resolves value, and event messages to their final destination.

The APSIM executable (apsim.exe) takes a single command line argument (an SDML script), creates an instance of a very small transport layer class (that is capable of delivering messages) and creates a top-level protocol manager. It then passes the SDML script to the protocol manager. The protocol manager then commences the simulation.

In the APSRU implementation, each logical system of the simulation is, by default, *closed;* variables and events remain within that system and are not visible to the wider simulation. The simulation designer must explicitly expose (and perhaps alias) variables and events as required. This is analogous to a C++ class where variables and methods are, by default, private to that class. This system level scoping provides APSIM with a capability to simulate multiple independent points, with the data flows between the points explicitly defined by the simulation designer.

3.1.3. The CSIRO Plant Industry implementation

CPI has implemented the protocol proper using standard C++ code. A freely-available XML parser has been used to read and write SDML and DDML. The code implementing the protocol resides within a single dynamic-link library that may be called by any program to execute a simulation. A separate "wrapper" DLL has also been written in C++. At the time of this report, the CPI implementation complies with most of the protocol specification; the main exception is checkpointing. This protocol feature is not required to run simulations using APSRU components.

Some of the components are also implemented in C++, but most are written in the Object Pascal language that forms part of the Borland Delphi software development tool (Inprise Corporation 1999). This strategy was adopted to make use of a large quantity of model code already implemented as Object Pascal classes.

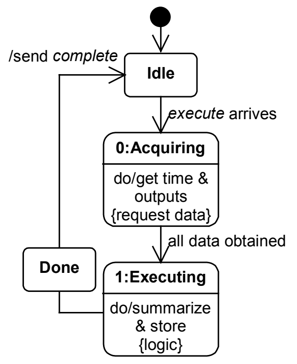


Figure 3.2. States and transitions for an event of the output component in .the CPI implementation of the protocol.

The interface code for each model component is implemented as a class, descended from a base "model instance" class that carries out the basic message-handling functionality. Event handling is implemented by treating each event as a state machine. The computation waits in some of the states while information (e.g. driving variable values) is acquired from other parts of the simulation, and alternative paths of computation can be followed depending on the results of computations in the current state. The simulation writer controls the order of computations within each time step, by giving the order of a set of computation events to a "sequencer" component.

Within the CPI components, a convention is followed that flows of information between components are represented as requests for property values, while flows of mass, energy or other conserved quantities are represented as events.

3.2. The process of implementing a sub-model

Adding a new model of biological or physical processes to the protocol typically follows three key steps:

1. **Define the "boundaries" of the sub-model(s).** This step involves making an assessment of how the new processes relate to one another and to existing processes that are already modelled in the protocol. Equations that are tightly coupled (in the sense of sharing common quantities) should generally form part of the same sub-model. The boundary of an existing submodel needs to be made consistent with the boundaries of pre-existing components – it is poor practice to have different components own properties that represent exactly the same state variable.

   There is a tradeoff between the flexibility gained by having many small components, and the efficiency of having fewer, larger components. Making this tradeoff is part of the modeller's art.

2. **Specify the interface of the sub-model.** Once a conceptual boundary has been drawn around a sub-model, it is a straightforward matter to identify which of the quantities in the model equations are driving variables, which are state variables, and which are parameters that will require initialization. The definition of the component's properties follows immediately from this inspection of the equations. The choice of property names and types will usually be guided largely by the definition of corresponding properties in existing components, or by conventions followed by the modelling group.

   The identification and definition of a component's event handlers may not be as straightforward. A component may have an event handler or handlers that will be called during every time step to carry out integration of its rate equations. It may also have event handlers that correspond to the triggers of logical events. Lastly, other components may transmit information (for example the amount of a flow of organic matter) by means of events.

   Maintaining consistency between components is of primary importance during this step.

3. **Implement the sub-model as a component.** In a well-designed protocol implementation, the implementation of a component interface as computer code will be a routine task once the submodel's interface is specified and its equations are correctly coded. It is good practice to keep the interface code and the code containing model logic separate.

In practice, of course, the component development process tends to be iterative. the component interface evolves as the science of interactions with other sub-models is resolved.

3.3. Issues arising when models from different sources are coupled

Making sub-models into protocol-compliant software components is only part of the process of making them genuinely interchangeable between modelling groups. The components must have interfaces that are sufficiently compatible to permit exchange of all the necessary information needed to compute their dynamics.

During this project we have reached a point where the GRAZPLAN pasture and animals, the SWIM water balance and the APSRU wheat model are scientifically as well as technically compatible. Along the way we have had to resolve issues at each stage of the component implementation process:

- **Differences in sub-model boundaries.** Different perspectives of the system lead to different (but equally valid) choices about where the boundaries between sub-models lie. For example, crop models have evolved from representing only the growing crop to more comprehensive models that follow the fate of surface residues. In this context it is natural to consider the equations describing the living crop and those for dead surface residues separately. When animal production is the primary aspect under consideration, however, live and dead shoot material are both food for the livestock; their similarities are seen as more important than their differences and it is natural to include them in a single component. Consequently the GRAZPLAN pasture model may be viewed as a combined "crop growth" model and "surface residue" model.

  In all but one case, the differences in boundaries between APSRU and CPI have been such that a single component from one group corresponds exactly to two or more components from the other. Such differences cause few problems of interchangeability. The exception was the question of whether the rate of soil water uptake should be treated as a soil water process (as assumed by SWIM) or as a plant process (as assumed by the pasture model). We resolved this issue by changing the pasture model so that it could be configured to either handle externally-computed transpiration rates, or compute them itself.

- **Differences in conceptualization of processes.** These lead to different state variable sets and often to different driving variables for otherwise similar components. Such differences were remarkably few; perhaps the most important is the treatment of growth restriction due to water deficit between the pasture and crop growth model, but this did not impact on the component interfaces

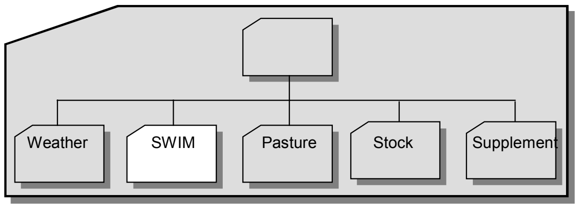
- **Keeping the representation of quantities and flows consistent.** Most of the potential issues here were circumvented by holding a meeting between CPI and APSRU, during which the interactions between components were identified and methods for ensuring consistency were resolved upon. Three approaches were used, depending on the degree of difference in the original implementations:
  (i) In the majority of cases, a common name and type was agreed for properties and events conveying the same information.
  (ii) APSRU converted some of the property values and events from the pasture component into forms recognised by APSRU soil and water components by writing a small "translator" component.
  (iii) CPI took an alternative approach, in which components are capable of first detecting the presence of APSRU components by querying the structure of the simulation, and then setting up their interfaces dynamically. The necessary translations are then performed internally.

Differences in component implementations often led to lively and productive debate over how processes should best be conceptualized and implemented. Both the APSRU and the CPI components are better implemented as a result of the challenge posed by the need to link to models developed with a different mind-set.

4. Example simulations

This section describes three simulation exercises in which the common modelling protocol has been used to couple models from different modelling groups. The specific model outputs should not be taken as definitive analyses of the systems simulated, but instead be seen as illustrative of the capability of the protocol. Further refinement in soil, genotype and management inputs will influence the simulation results.

4.1. SWIM water balance under grazed pasture



The first set of simulations demonstrates the use of the protocol to examine interactions between deep drainage and production from grazed pastures.

A simulation experiment was carried out to analyze the effects of pasture type and stocking rate on wool production and the amount of water draining past the root zone for hypothetical grazed pastures at Wagga Wagga, New South Wales. The soil was a reasonably well-drained red Kandosol (K. Verburg, *pers. comm.*). Fertility of the soil was assumed to be moderately high. Pastures were grazed continuously by medium Merino wethers; 25% of animals were replaced each year, and sheep were fed a ration of 500g/head/day of wheat if their condition score fell below 1. Two pasture types (annual grasses and *Phalaris aquatica*) and five stocking rates (0, 4, 8, 12 and 16 wethers/ha) were compared in a factorial design. Annual grasses were assumed to have a maximum rooting depth of 600mm, while phalaris roots were assumed to extend to 800mm.

Each simulation was computed over the 21 years from 1979 to 1999; the first year was omitted from the results to remove any effect of initial conditions. A component containing the SWIM water balance, implemented by APSRU, was coupled with CPI's weather, pasture, ruminant biology and supplementary feed components within the CPI implementation of the protocol (FarmWi$e).

Figure 4.1 shows time courses of the simulation for annual pasture stocked at 8 wethers/ha. Figure 4.2 shows the long-term average values of three key outputs: the amount of water draining past a depth of 1.5m, wool produced per area of land and the amount of supplementary feeding required. The greater water use by the phalaris pasture is reflected in higher levels of wool production and lower feeding requirements.

Stocking rate has only relatively small effects on drainage in this particular example (Figure 4.2). Somewhat unexpectedly, the coupled model predicts higher drainage at the lowest stocking rates. At low stocking rates, large quantities of dead herbage remain uneaten. They act as a mulch that reduces soil evaporation, leading to higher average soil water contents and hence to higher drainage amounts.

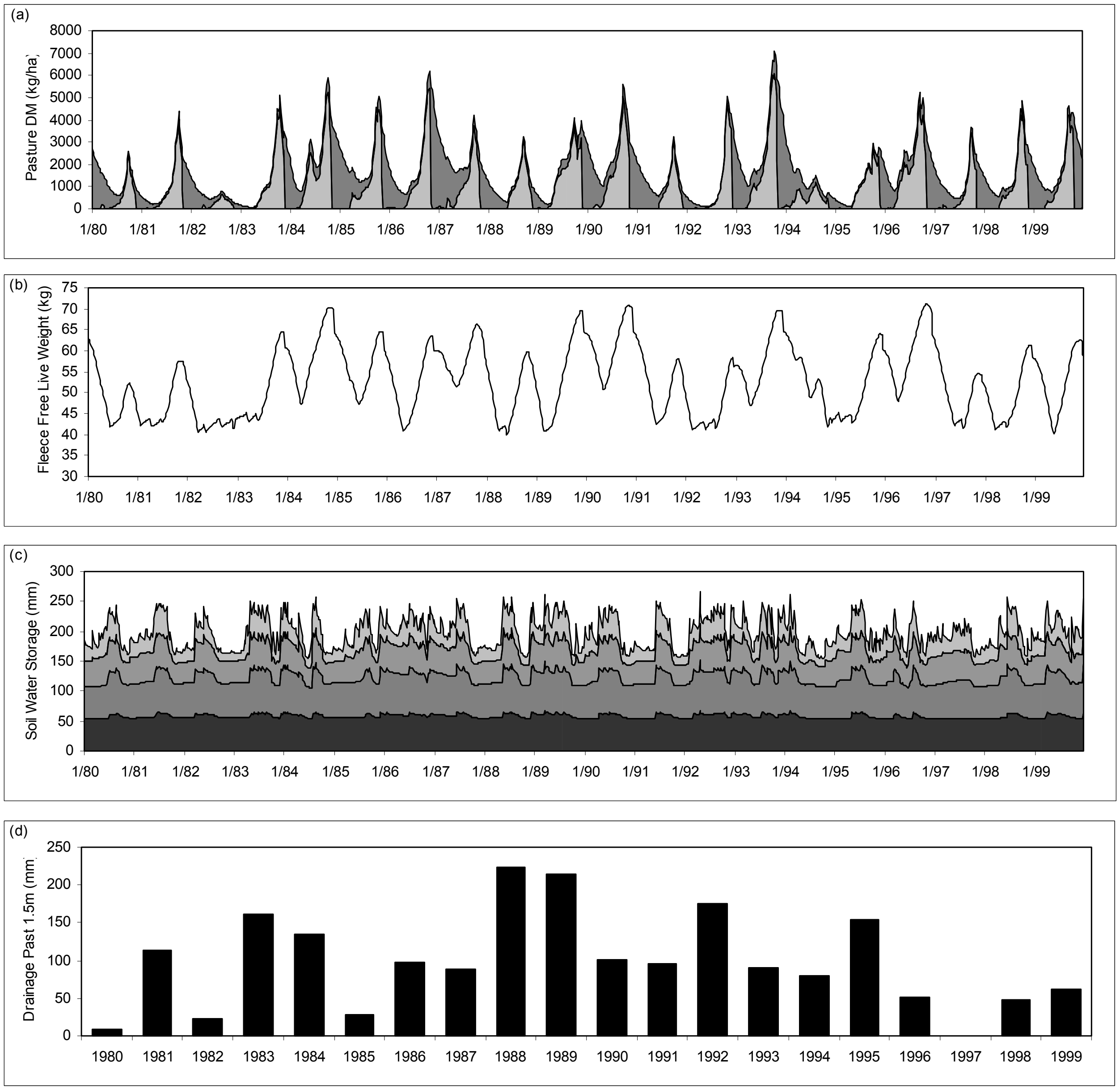
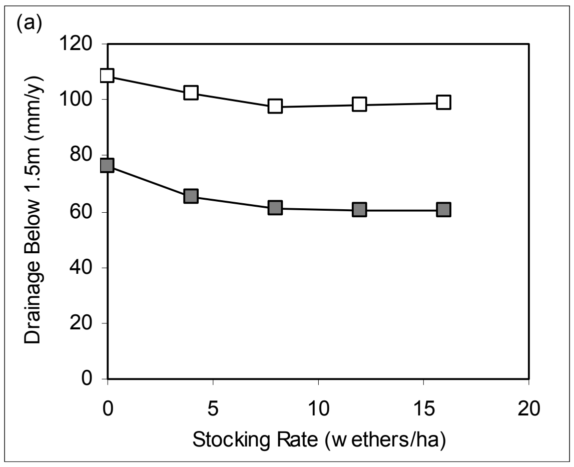
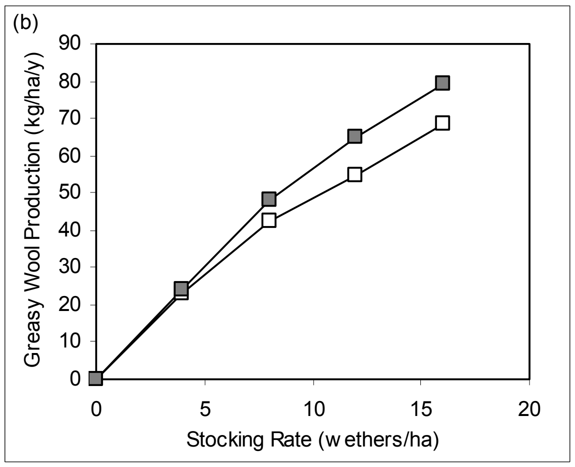


Figure 4.1. The SWIM water balance coupled to the GRAZPLAN pasture and animal models. Simulated time course over the period 1980-1999 of
(a) pasture dry matter available for grazing (▢ live; ▨ standing dead+litter),
(b) fleece-free wether live weight,
(c) soil water storage in the rooting zone (▢ 0-150mm; ▨ 150-350mm; ▨ 350-600mm; ■ 600-800mm), and
(d) annual drainage of water below a depth of 1500mm
for a hypothetical grazing system at Wagga Wagga, New South Wales in which annual grass pasture is grazed by medium Merino wethers at a stocking rate of 8/ha.





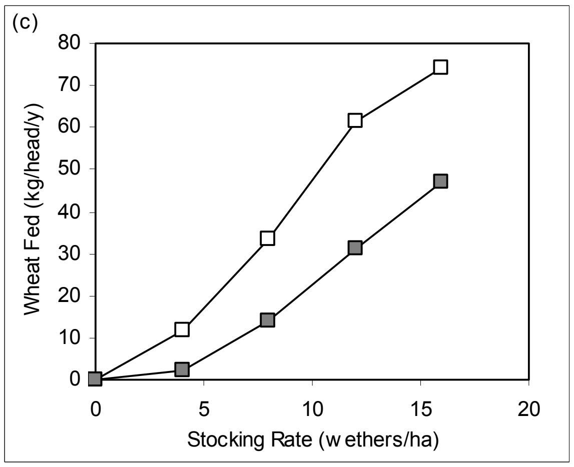
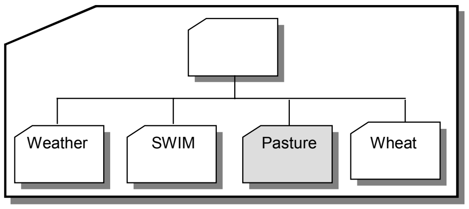


Figure 4.2. The SWIM water balance coupled to the GRAZPLAN pasture and animal models. Simulated effect of pasture type and stocking rate on
(a) long-term average drainage of water past 1.5m,
(b) production of greasy wool per hectare, and
(c) the average supplementary feeding requirement for a hypothetical grazing system at Wagga Wagga, New South Wales. □ annual grass; ■ phalaris.

4.2. Crop-pasture simulation using the APSRU implementation



The second simulation demonstrates a model configuration in which the APSRU wheat model, GRAZPLAN pasture model and the SWIM water balance are used together to simulate a crop-pasture system.

The simulated system was a wheat-pasture phase farming system at Gunnedah, New South Wales. The soil was parameterized to represent a red-brown earth. Each wheat and pasture phase had a set duration of three years (i.e. PPPWWW). Phalaris was sown in early April and removed by cultivation after three growing seasons. Crops during the wheat phase were sown in early June and harvested at physiological maturity.

The simulation was run over the twelve-year period from January 1960 to December 1971. The component containing the GRAZPLAN pasture model, implemented by CPI, was coupled with APSRU's weather and wheat components and the APSRU implementation of SWIM within the APSRU implementation of the protocol (APSIM).

A wheat-pasture phase farming system has been simulated using the Farmwi$e pasture model operating within APSIM. Each wheat and pasture phase has a set duration of three years. Phalaris is sown in early April and cultivated out after three growing seasons. Crops during the wheat phase are sown in early June and harvested at physiological maturity. The simulation uses climate data from Gunnedah NSW for the twelve year period from January 1960 to December 1971. The soil is parameterised to represent a red-brown earth using the APSIM-SWIM water balance module. Results of this simulation are presented in Figure 4.2.

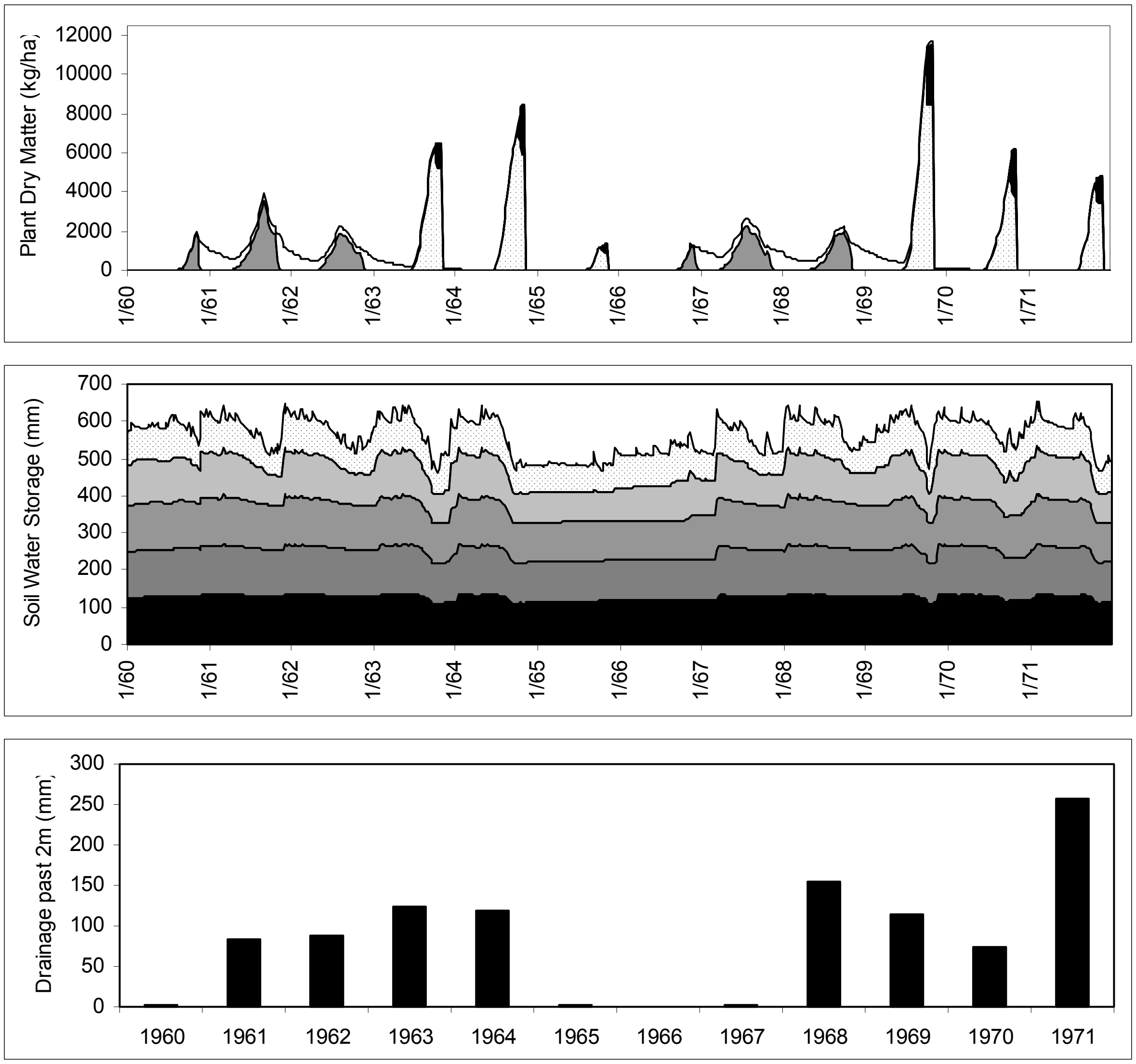
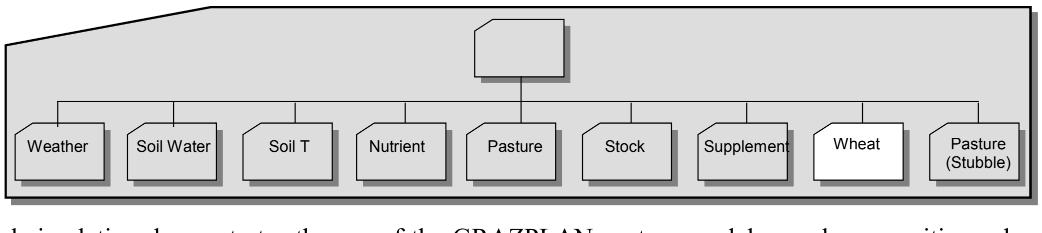


Figure 4.3. The APSRU wheat model, SWIM water balance and GRAZPLAN pasture model coupled together.
(a) Total plant dry matter (▩ live pasture; ☐ dry pasture; ⬚ wheat stover; ■ wheat grain),
(b) soil water storage (⬚ 0-400mm; ▤ 400-800mm; ▦ 800-1200mm; ▩ 1200-1600mm; ■ 1600-2000mm), and
(c) annual drainage of water below a depth of 2000mm
for a simulated pasture-wheat phase farming system at Gunnedah. Phalaris pasture is sown in April of every sixth year; at the end of three years the phalaris is killed by cultivation and three successive wheat crops are sown in early June.

4.3. Crop-pasture simulation using the CSIRO Plant Industry implementation



The final simulation demonstrates the use of the GRAZPLAN pasture model as a decomposition submodel for wheat residues, thereby enabling grazing of wheat stubbles to be simulated. The simulated system was a short pasture-crop sequence at Roseworthy, South Australia. The soil was a red-brown earth with relatively low soil organic carbon content (soil 32 of Forrest *et al.* 1985). At the start of the simulation, wheat stubbles were grazed by medium Merino wethers. In each of two years, a volunteer annual grass pasture germinated and was grazed by the sheep. In June of the second year, once soil moisture in the top 0.6m reached a specified level, the paddock was cultivated and a wheat crop (cv Janz) was sown. At harvest of the wheat crop, livestock were returned to the paddock and grazed the stubbles once more.

The simulation was computed from 1 March 1995 to 28 February 1997. A component containing the APSRU wheat model, was coupled with CPI's weather, soil water, soil temperature, soil nutrient, pasture, ruminant biology and supplementary feed components within the CPI implementation of the protocol. One copy of the pasture model was used to represent the volunteer annual grass, while the other was used to represent the wheat stubble.

Results of the simulation are presented in Figure 4.4. Note (i) the appearance of volunteer grasses in the second year before the wheat crop is sown, and (ii) the rapid weight loss of the sheep grazing stubble in summer 1996-7. The latter reflects a assumption in the wheat model that no grain is lost during harvest; in reality the small quantity of fallen grain represents a significant element of the diet in these circumstances.

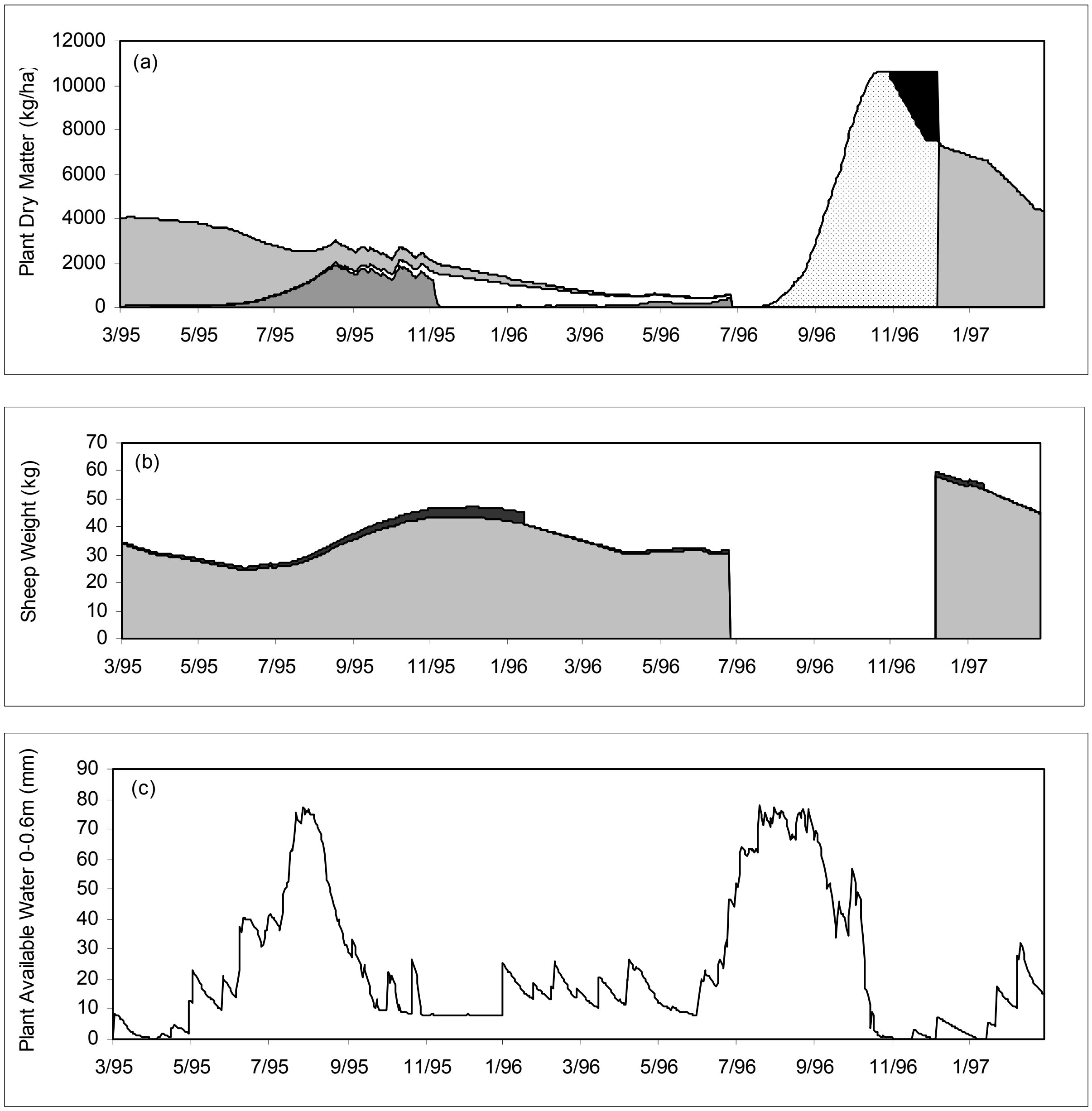


Figure 4.4. The GRAZPLAN pasture and animal models coupled to the APSRU wheat model. Simulation of a grazed annual grass pasture followed by a wheat crop at Roseworthy, with wheat stubble grazed by sheep:
(a) total plant dry matter (▨ wheat stubble; ■ live pasture; ❑ dry pasture; ❏ wheat stover; ■ wheat grain),
(b) wether live weight (▨ fleece-free weight; ■ greasy fleece weight), and
(c) plant-available water to 0.6m depth.
Annual grass pasture and wheat stubbles are grazed by medium Merino wethers at a stocking rate of 6/ha; a wheat crop (cv Janz) is sown in the second year when plant-available water reaches a specified threshold.

5. Opportunities for future work

Opportunities for future work with the protocol were outlined in an earlier project report (Moore & Keating 2001) This report sets out plans from the perspective of APSRU and CPI, as well as the project partners' common views on issues such as commercialization and access to models. An important dimension of this is promoting the adoption of the protocol beyond CSIRO by other modelling groups. The report also contains a description of principles for guiding access to the APSIM and FarmWi$e software that is based on the protocol.

6. Acknowledgements

We wish to acknowledge the substantial contributions made to the development of the common modelling protocol by

- Sid Wright (formerly of APSRU and CSIRO Mathematical & Information Sciences),
- Ben Higgins (formerly of CSIRO Plant Industry), and
- Brian Keating (APSRU).

This work has been funded by Land and Water Australia as part of the Redesigning Agriculture for Australian Landscapes Program.

7. References

Arcus PL (1963) An introduction to the use of simulation in the study of grazing management problems. *Proceedings of the New Zealand Society of Animal Production* **23**, 159-168.

Beek J & Frissel MJ (1973). *Simulation of nitrogen behaviour in soils.* PUDOC, Wageningen.

Brouwer R & de Wit CT (1968). A simulation model of plant growth with special attention to root growth and its consequences. *Proceedings of the 15th Easter School of Agriculture Science,* 224-242.

Christian KR, Freer M, Davidson JL, Donnelly JR & Armstrong JS (1978). *Simulation of grazing systems.* PUDOC, Wageningen.

Clarke D (2000) *Redesigning agriculture for Australian landscapes research & development program - phase 2 program strategy 2000–2002*. Land and Water Resources Research and Development Corporation, Canberra

Donnelly JR, Moore AD & Freer M (1997). GRAZPLAN: decision support systems for Australian grazing enterprises. I. Overview of the GRAZPLAN project, and a description of the MetAccess and LambAlive DSS. *Agricultural Systems* **54**, 57-76.

Freer M, Moore AD & Donnelly JR (1997) GRAZPLAN: decision support systems for Australian grazing enterprises. II. The animal biology model for feed intake, production and reproduction and the GrazFeed DSS. *Agricultural Systems* **54**, 77-126.

Inprise Corporation (1999) *Borland Delphi 5 developrs' guide.* Inprise Corporation, Scotts Valley, California.

Keating BA, Carberry PS, Hammer GL, Probert ME, Robertson MJ, Holzworth DP, Huth NI, Hargreaves JNG, Meinke H, Verburg K, Snow V, Dimes JP, Silburn M, Wang E, Brown S, Hochman Z, McLean G, Asseng S, Chapman S, McCown RL, Freebairn DM & Smith CJ (2002) The Agricultural Production Systems Simulator (APSIM): its history and current capability. *European Journal of Agronomy* (in press)

McCown RL, Hammer GL, Hargreaves JNG, Holzworth DP & Freebairn DM (1996). APSIM: a novel software system for model development, model testing, and simulation in agricultural systems research. *Agricultural Systems* **50**, 255–71.

Moore AD, Donnelly JR & Freer M (1997) GRAZPLAN: decision support systems for Australian grazing enterprises. III. Pasture growth and soil moisture submodels and the GrassGro DSS. *Agricultural Systems* **55**, 535-582.

Moore AD & Keating BA (2001) *Plan for the use, extension and adoption of software systems based on the CSIRO common modelling protocol*. Unpublished report to Land and Water Australia, project CPI9.

Object Management Group (2001). *OMG Unified Modeling Language specification, version 1.4.* Published electronically at http://www.omg.org/technology/documents/formal/uml.htm.

O'Neill RV, De Angelis DL, Waide JB & Allen TFH (1986). *A hierarchical concept of ecosystems*. Princeton University Press, Princeton NJ.

Stauffacher M, Bond WJ, Bradford AS, Coram J, Cresswell HP, Dawes WR, Gilfedder M, Huth NI, Keating BA, Moore AD, Paydar Z, Probert ME, Simpson RJ, Stefanski A & Walker GR (2000) *Assessment of salinity management options for Wanilla, Eyre Peninsula: groundwater and crop water balance modelling.* Technical Report 1/00, CSIRO Land and Water, Canberra.

Verburg K, Ross PJ & Bristow KL (1996). *SWIM v2.1 user manual*. Divisional Report No. 130, CSIRO Division of Soils, Canberra.

World Wide Web Consortium (2000) *Extensible markup language (XML) 1.0* (2nd ed). W3C Recommendation, 6 Oct 2000. Published electronically at http://www.w3.org/TR/REC-xml.