Environmental Modelling & Software 25 (2010) 269–275

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

Simplifying environmental model reuse

Dean P. Holzworth [a,*], Neil I. Huth [a], Peter G. de Voil [b]

[a] CSIRO Sustainable Ecosystems, P.O. Box 102, Toowoomba 4350, Queensland, Australia
[b] Department of Primary Industries and Fisheries, P.O. Box 102, Toowoomba 4350, Queensland, Australia

ARTICLE INFO

Article history:
Received 14 May 2008
Received in revised form
2 October 2008
Accepted 20 October 2008
Available online 13 December 2008

Keywords:
Model
Simulation
Component
Interoperability
Framework independent
APSIM

ABSTRACT

The environmental modelling community has developed many models with varying levels of complexity and functionality. Many of these have overlapping problem domains, have very similar 'science' and yet are not compatible with each other. The modelling community recognises the benefits to model exchange and reuse, but often it is perceived to be easier to (re)create a new model than to take an existing one and adapt it to new needs.

Many of these third party models have been incorporated into the Agricultural Production Systems Simulator (APSIM), a farming systems modelling framework. Some of the issues encountered during this process were system boundary issues (the functional boundary between models and sub-models), mixed programming languages, differences in data semantics, intellectual property and ownership.

This paper looks at these difficulties and how they were overcome. It explores some software development techniques that facilitated the process and discusses some guidelines that can not only make this process simpler but also move models towards framework independence.

Software availability

Name: Agricultural Production Systems Simulator (APSIM)
Developer: Agricultural Production Systems Research Unit (APSRU)
Contact address: P.O. Box 102, Toowoomba 4350, Queensland, Australia
Tel.: +61 7 4688 1596
Email: christopher.murphy@dpi.qld.gov.au
First available: 1992
Hardware required: Microsoft Windows (98 or later) compatible PC
Software required: Microsoft .NET 2
Language: FORTRAN, C++, C#.NET and VB.NET
Program Size: 54 MB (download size)
Availability: www.apsim.info
Cost: Free.

1. Introduction

The software engineering industry has been striving for source code reuse for decades. This has progressed from the sharing of libraries of subroutines in the 1960s and 1970s, to object orientated techniques in the 1980s and 1990s, through to more recent component-based designs. Many implementation technologies have been created in an attempt to facilitate sharing of routines, classes and components. These range from simple exchange of binary libraries through to more complex technologies such as Microsoft COM, CORBA®, Java NetBeans®, and Microsoft .NET™. Some of these technologies have seen wider adoption than others.

Reuse and extension of existing work are preferable to creating something completely new. Utilisation of existing work is much more efficient and potentially more robust when the existing technology is already proven. However, when new functionality is required, a new solution is often favoured over extending existing work. While extending existing work can be time consuming, especially when the existing work isn't well designed, the flow-on benefits of extending are often not considered when deciding between extending and starting anew. Building extra capability around an existing solution aids others and often brings additional capabilities that aren't currently required but may prove beneficial in the future.

The environmental modelling community has, in the last decade, been actively developing simulation models of various environmental and farming systems processes (e.g. Keating et al., 2003; Donatelli et al., 2005; Acutis et al., 2007). Many of these efforts start small with a goal of extending knowledge of a process. As such, the desire to start anew is understandable. Problems arise though when the model is shared and extended by others when extendibility was not built into the original design. Very quickly the model becomes difficult to understand and unworkable. If sufficient demand and resources warrant it, a rewrite follows, building

* Corresponding author. Tel.: +61 7 4688 1279.
E-mail address: dean.holzworth@csiro.au (D.P. Holzworth).

1364-8152/$ – see front matter
doi:10.1016/j.envsoft.2008.10.018

in extra capability and extensibility. All software modelling projects follow this evolutionary process of renewal. Most models never progress beyond the concept stage, some remain as small research projects, some gather a small following within a region or organisation and a few become large simulation frameworks. A framework is defined as a group of interconnected models with infrastructure to support inter-model communication.

The authors are developers of a farming systems simulation framework called the Agricultural Production Systems Simulator (APSIM) (Keating et al., 2003). It is used widely in Australia and increasingly overseas. Other frameworks include the spatial "catchment modelling toolkit" used in Australia called TIME (Rahman et al., 2003), OpenMI (Gijsbers et al., 2005) used in Europe to link time dependent models, AusFarm (Moore, 2001) a grazing systems model used in Australia and MODCOM (Hillyer et al., 2003), the protocol for linking components in the APES (http://www.apesimulator.it/default.aspx) model that is a part of the European SEAMLESS project. The Decision Support System for Agrotechnology Transfer (DSSAT) is a collection of crop, soil and weather models produced by the ICASA (http://www.icasa.net/dssat) consortium of scientists that is used extensively in the United States. CropSyst (Stockle, 2003) is another cropping systems simulation model used widely in the US and elsewhere. Many of these efforts cover different, but overlapping, problem domains. Several of them have different implementations of very similar components, e.g. soil water/nutrient balance, wheat or maize crops. Some of them are linkage protocols (e.g. TIME, OpenMI and MODCOM). In general though, they are mostly incompatible with each other; the exceptions being APSIM/AusFarm (Moore et al., 2007) and the APES models.

In an ideal world, models from different backgrounds and approaches could be easily combined and linked to create new and interesting possibilities. Models of the same type, for example, that of a wheat crop, could be compared and contrasted within a simulation using the same water, nutrient and environment components. Work from one organisation or individual would become easily transferable to other scientists and used in new and unexpected ways. Given the advantages of reuse, and the considerable overlap of functionality in the cropping system domain, it is desirable that some level of compatibility be created between the various models and the frameworks. Attempts have been made in the past to enable this sort of component swapping between frameworks. Moore et al. (2007) describes the creation of a binary protocol that was implemented in both APSIM and AusFarm. This low-level protocol enables models to be interchanged, realising many of the benefits of reuse. While other frameworks could also adopt this same protocol in order to exchange model components with APSIM and AusFarm, this is neither practical nor desirable. A diversity of approaches and technologies brings new ideas and techniques to a project.

While it is not currently possible to have an APSIM model run in other frameworks and vice versa, it is possible to apply several software development techniques that it makes it easier to execute the model in different frameworks. However, many non-engineering issues make this difficult. Intellectual property (IP) and ownership issues, social issues, difficulty in achieving agreement on a 'standard', semantic meaning of data and software related issues all combine to make it difficult to realise this goal. Some of these issues, particularly the software related ones, can be solved while others are more difficult, particularly those with a human or organisational origin.

This paper looks at several experiences of connecting external models into APSIM and examines some software development practices that have aided this process. It concludes by extending the APSIM experience into a set of principles for constructing models in a given problem domain such that they can more easily be incorporated into multiple frameworks. This paper does not subscribe to the notion of standardisation of models or frameworks. Rather, it embraces diversity and looks at the issues involved and examines how some of the work in APSIM is moving towards framework independence.

2. Issues faced when bringing foreign models into APSIM

2.1. Model execution

A farming systems model like APSIM has its own separate crop, environment, soil water and nutrient models. The crop models can be rotated in and out of a simulation, taking up soil resources as required, leaving the soil in an altered state. This soil centric focus is what separates a systems model like APSIM from other single, standalone crop models.

When connecting a foreign crop model into APSIM, it is therefore the crop growth portion of the model that is of interest. Quite often this requires decoupling of this from other components within the foreign modelling environment such as the soil water and nutrient balance or input/output systems. The crop growth portion that remains often needs restructuring to create an entry point that APSIM can call once each timestep. Other frameworks like TIME also have this requirement for a timestep entry point. This disaggregating and restructuring of source code is time consuming, particularly when the original model has its sub-components tightly coupled. When OZCOT (Hearn, 1994), a cotton crop model, ORYZA2000 (Bouman et al., 2001), a rice model and GRASP (McKeon et al., 1990) a native pasture and grazing model were connected into APSIM, the approach taken was to extract just the required science source code and wrap this with code that provided the required soil and weather data from other APSIM model components (Fig. 1). The original models all had sub-models that were tightly coupled, for example, the growth routines referenced variables in the soil (e.g. soil water and nitrate) and environment (e.g. maximum temperature) and these had to be replaced with calls into APSIM. The resulting model then gave different results from the standalone version of the model as it was linked to different water and nitrogen balance models. A process of recalibration then ensued further adding to the time consuming process.

This system boundary issue, defined as where the boundary is drawn between one model or sub-model and the next, can be a very subjective and subtle one. Difficulties can occur even when the two models seemingly have their system boundaries in similar parts of the problem domain. This was the case when SWIM (Verburg et al., 1996), a water balance model based upon a solution of Richards' equation, was provided as an alternative to APSIM's standard "tipping bucket" water balance, SOILWAT (Probert et al., 1998). Both simulate water movement in the soil, albeit in a different manner. It was only upon closer examination that SWIM was revealed to calculate the water uptake by all crops in a simultaneous manner, something that SOILWAT doesn't do. This disparity was solved by modifying the crop models to allow SWIM to provide their water uptake when SWIM is the chosen water balance model, but calculate their own uptake when SOILWAT is the choice of model.

User interface coupling also hinders the ability to connect a foreign model to a framework like APSIM. Models that are intrinsically linked to user interfaces can be very difficult to run in frameworks that have a different user interface or even none at all. Models that have forms and user interface events embedded in the science require major recoding to remove the user interface portions. In APSIM, UI components have had short lifetimes ($\sim$3 years) while models have much greater longevity (5–10 years), thus emphasising the need to separate models from the user interface.

In a few rare situations, models were inserted into APSIM with no exchange of source code. AusFarm (Moore, 2001) and APSIM share a binary protocol called the CSIRO Common Modelling Protocol (CMP) (Moore et al., 2007). This protocol allows a model

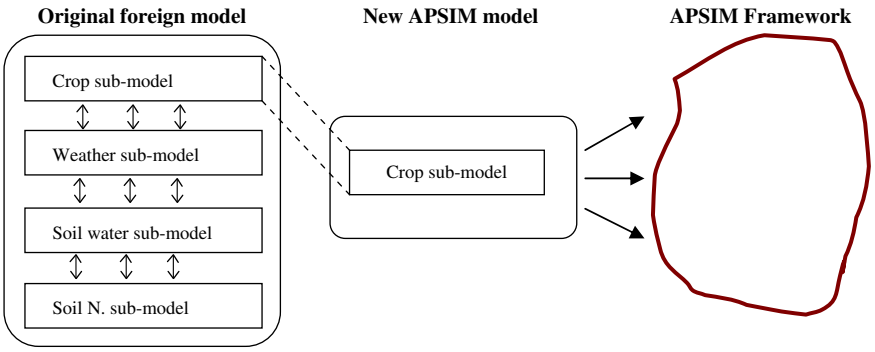

Fig. 1. To connect a foreign model into APSIM, the approach taken was to extract the required sub-model (e.g. crop growth) and modify all communications so that they are routed through the APSIM framework.

from AusFarm to be executed in APSIM, and vice versa, by simply copying an executable file from one system to the other. The CMP defines a binary transport mechanism and describes a message based mechanism for packing and unpacking data, entry points in the executable and a set of defined messages to transfer variables and events from one model to another. This allows easy exchange of models between AusFarm and APSIM and it has been used with great success. However, expecting other modellers to adopt and implement the complex CMP to gain APSIM compatibility is unrealistic, particularly when resources are limited.

Another option exists, in some circumstances, when a foreign crop model is needed in APSIM. Wang et al. (2003) and Robertson et al. (2002) describe generic modelling approaches to parameterising crop models in APSIM. This work has been incorporated into a generic PLANT model that can be parameterised from a configuration file (XML). The PLANT model reads a configuration file at runtime and dynamically instantiates crop growth processes, passing initialisation parameters that describe various growth characteristics. Sometimes it is necessary to create new process classes in PLANT to capture new functionality when adding a new crop, particularly when the new crop model is quite different to existing models. These new processes then become part of the 'library' of processes which can be instantiated for other models. This approach has been used to create many new crop and tree models in APSIM and could be used as an alternative to the alternative source code restructuring approaches. Rather than extensive source code modification, it may be simpler to extract the required equations and parameters, creating a new configuration of PLANT.

2.2. Data semantics

A successful model compiles and link is only one small step towards having a model run correctly. The meaning of data required by a model needs to be fully understood before the model is run. This understanding needs to include the set of data is required by a model, its units and its temporal and spatial context. Once the data requirements of a model are understood, the framework within which it will run needs to provide the correct data. In APSIM, this matching of data requirement to supply is done automatically, based on name alone. If the soil water model supplies a variable called 'sw' and the plant model has a requirement for 'sw', they are matched and connected at runtime. It makes no attempt to match units or semantics. While this works for known components, difficulties can arise when working with foreign components. Variable meaning becomes important, for example, the variable 'sw' may refer to the volumetric or gravimetric water content; it may be a layered array variable as in APSIM or a summed variable over the soil profile. In APSIM, even though variables are matched and linked at runtime, this doesn't guarantee a valid connection. If there is a semantic mismatch then the simulation is invalid.

Good documentation is often cited as a cure for understanding the data requirements of a model. While this is true to some extent, the time needed to create this documentation can be large and unless it is well maintained over time, can quickly become out of date. Documentation that does not accurately reflect the source code can be worse than no documentation at all. One reason for this is the separation between documentation and source code. It therefore follows that if the implementation of documentation and source code was more closely coupled, compatibility is more likely to be maintained. APSIM strives to keep the semantics of the data with the source code wherever possible.

APSIM doesn't precisely define all inputs and outputs, instead preferring a short description, units and its shape (scalar or array). This simple approach has been sufficient for the many APSIM collaborative projects over the past 15 years. As APSIM becomes open-source, perhaps some improvement to these metadata may become necessary. The philosophy behind APSIM's development in recent years has been to solve problems with the simplest solution possible. It is always much easier to step up the complexity ladder later than to move down it. This is also one of the fundamental principles of the agile software development community (Jeffries et al., 2001) that have been adopted by the APSIM development team.

A mismatch between the data semantics of two models is common when bringing a foreign model into a framework. While modifying the foreign model, the framework, or both, will overcome the problem, sometimes this standardisation isn't desirable. It is often better to leave both untouched and work around the data differences. One approach, that APSIM has adopted, is to create 'helper components'. These small components translate the names, and in some cases the meaning, of variables between two models such that they coexist. For example, when the AusFarm STOCK model runs in APSIM, an instance of a helper component is created during the simulation. This converter translates individual dry matter output variables from APSIM PLANT into a structure containing herbage masses and other variables used by the STOCK model. The advantage of this approach is that neither PLANT nor STOCK required any changes. In a sense, this is a form of wrapping, the process where a model is given a different external view so that it appears and operates differently. Often, some form of wrapping is an expedient way of bridging the gap between foreign models without needing to make changes to the models.

2.3. Other issues

Some of the more difficult issues to solve are related to ownership and intellectual property (IP). In some cases, licencing

for source code access has placed severe restrictions on the use and extent of code modifications that are allowed. Ultimately, a limited connection capability has been achieved but a much more flexible solution would have been possible had it not been for the IP issues. Restrictions placed on source code and the resulting modifications can be detrimental to collaboration and the openness of a process. Indeed, until very recently, the APSIM framework had been a closed system, almost discouraging others from extending its components. The lack of availability of source code and restrictions on its modification led others to perceive APSIM as being non-transparent, a black box too difficult to work with. APSIM is becoming more open and attractive to other modellers with a planned move to open-source. Most restrictions are created by a perceived lack of trust or an unwillingness to relinquish some control to others. Allowing others to modify a model to run in another framework does not mean losing ownership of that model.

3. Software development techniques that can help

The problems outlined in the previous section can be largely circumvented if models are better designed using several, well established, software development techniques.

The 'science' of the various parts of a model (equations and such like) should be loosely coupled to each other and to other models in the simulation. Abstraction is a key technique to enable this separation of logic. When the science of the model is mixed with the framework dependant communication code or other technology related issues, then bringing this science into APSIM can be difficult.

3.1. Design-by-interface

One approach to decoupling models is to use a design-by-interface pattern (Lowy, 2006). This pattern describes an approach where source code calls an abstract series of functions to perform a service for the source code. For example, when the science of a model needs to obtain data from elsewhere (e.g. a crop model needs maximum temperature) it calls an interface to get the data. The model doesn't need to know how the data are obtained or even where the data have come from, only that a valid value is returned when requested. The interface provides a service to the science source code. To connect such a model to APSIM would require an implementation of the interface, something potentially much easier to do than substantial source code modification. The degree to which an implementation of an interface can easily be created is dependent on how simple and technology free the interface is and whether the target framework supports the required services. If the interface contains complex, solution space, and types (e.g. structures, variants, file handles, ID's) then the interface will be harder to implement and call than one with simpler strings, integers, and floating point types.

In early versions of APSIM, the interface that a model used to communicate with other models was very closely tied to the CMP. The CMP requires all models to perform communications in a particular, well defined way. This translated to models in APSIM also performing inter-model communications in this manner. The result was a lot of model source code coupled to the CMP, which meant that it was not easy to investigate alternative protocols or approaches. In recent times this CMP style, low-level interface has been redesigned, removing the need for CMP technologies and techniques to be inserted into the model science code. In redesigning the interface, a minimal set of services was created to support the model's science source code. Fig. 2 shows this interface for C++ models. There are methods to 'get' a variable value from the simulation, 'set' the value of a variable in another component, read an initialisation parameter and expose a variable to the simulation so that its value can be returned automatically in response to a 'get' request. APSIM is an event based simulation framework so there are also methods for publishing and subscribing to notification events. The arguments to these methods are simple types, for example, names and units expressed as strings, bounds as simple floating point numbers, etc. This simplicity significantly reduces the plumbing of a model and invites model builders to express their science cleanly and without clutter. It also allows the APSIM developers to explore alternative forms of inter-model communication, something that wasn't previously possible. This is one example where a technology laden interface has been simplified.

a

```
class ScienceAPI
{
 public:
  virtual bool read(string name, string units, float& data, float lower, float upper);
  virtual bool get(string name, string units, float& data, float lower, float upper);
  virtual void set(string name, string units, float& data);
  virtual void expose(string name, string units, string description, float& data);
  virtual void publish(string name, float& data);
  virtual void subscribe(string name, FloatHandler handler);
   . . .
```

b

```
scienceAPI.expose("plants", "plants/m^2", "Plant density", Density);
scienceAPI.get("latitude", "Degrees Celcius", Latitude, -90.0, 90.0);
scienceAPI.get("co2", "mg/kg",co2, 0.0, 1500.0)
```

Fig. 2. The interface class that models in APSIM use to communicate with other models showing (a) an extract from the C++ version of the interface with method names in bold and (b) three examples of using this interface, showing the metadata passed as arguments.

The interface in Fig. 2 shows the methods typed on *float*. Given that model science will typically need to work with different types and structures, an identical set of methods exist that are typed on the other APSIM types, e.g. *int*, *double* and more complex structures. Some languages like C++, C#.NET and VB.NET have the concept of templates or generics which allow overloading on type. The decision was made not to use these features as they aren't easily callable from FORTRAN and aren't as simple to understand. Likewise a variant type, one that is capable of representing different types, could have been used, but like generics, these are more complex to use. Instead, the APSIM typed interface is explicit with a 'get', 'set', and 'expose' for each of the several dozen APSIM types. The downside to such an approach is that the interface becomes very long. To combat the many issues of maintaining such a lengthy interface, we automatically generate it from an APSIM types specification (XML), alleviating the problem of hand coding for different types. Fig. 3 demonstrates this process. To incorporate a new type into APSIM, a developer creates a small XML description of it in the data types XML file and regenerates the interface. Such an interface provides a lot of flexibility to the model developer. A model can choose to get values of variables dependent on the state it is in. For example, a crop model may not get variables from the simulation at all when it hasn't been sown. Likewise some models will expose certain variables in some simulations and not others. For example, the APSIM PLANT component is capable of simulating multiple cereal and legume crops. When configured as a chickpea model, it will expose a different set of variables compared to when it is configured as wheat.

Simplicity is a very subjective term when applied to software interfaces. Fig. 4 shows an OpenMI (Moore and Tindall, 2005) interface for communications between models. It uses a 'pull' style of interface where a model requests data when needed, for example, a crop model would request maximum temperature each timestep. The OpenMI interface has a GetValues method for performing this variable request. The first argument to this method is

a

```
<type name="NewMet">
  <field name="today" kind="double"/>
  <field name="radn" kind="single" lower="0.0" upper="50.0" units="MJ/m2/d"/>
  <field name="maxt" kind="single" lower="-10.0" upper="70.0"  units="oC"/>
  <field name="mint" kind="single" lower="-20.0" upper="50.0"  units="oC"/>
  <field name="rain" kind="single" lower="0.0" upper="1000.0"  units="mm/d"/>
  </type>
```

b

```
//------ NewMet ------
struct NewMetType
   {
   double today;
   float radn;
   float maxt;
   float mint;
   float rain;
   };
```

c

```
class ScienceAPI
  {
  public:
    . . .
    virtual bool get(string name, NewMetType& data);
    virtual void set(string name, NewMetType& data);
    virtual void expose(string name, string description, NewMetType& data);
    virtual void publish(string name, NewMetType& data);
    virtual void subscribe(string name, NewMetTypeHandler handler);
    . . .
```

Fig. 3. To add a new data type to APSIM, the developer creates (a) an XML description of it and then auto-generates the source code for the interface. The auto-generator produces (b) a C++ structure for the type and (c) a series of interface methods for the type. The auto-generator also does this all supported APSIM languages; FORTRAN, C++ and the .NET languages.

a time parameter specifying a time span for the request and the second argument is a linkage object defining the location and quantity of the required value. The interface also defines methods for adding and removing linkage objects. This interface, while quite short, uses more complex types than the APSIM one presented in the previous paragraph. For example, the OpenMI interface

```
«interface» ILinkableComponent
+ «property» ComponentID() : string
+ «property» ComponentDescription() : string
+ «property» ModelID() : string
+ «property» ModelDescription() : string
+ «property» InputExchangeItemCount() : int
+ «property» OutputExchangeItemCount() : int
+ «property» TimeHorizon() : ITimeSpan
+ «property» EarliestInputTime() : ITimeStamp
+ Initialize(properties :IArgument[]) : void
+ GetInputExchangeItem(inputExchangeItemIndex :int) : IInputExchangeItem
+ GetOutputExchangeItem(outputExchangeItemIndex :int) : IOutputExchangeItem
+ AddLink(link :ILink) : void
+ RemoveLink(linkID :string) : void
+ Validate() : string
+ Prepare() : void
+ GetValues(time :ITime, linkID :string) : IValueSet
+ Finish() : void
+ Dispose() : void
```

Fig. 4. The OpenMI interface that models use to communicate with other models and the framework (Moore and Tindall, 2005).

requires the model developer to work with 'inputExchangeItemIndex' and 'outputExchangeItemIndex' and 'ILink' types. These are very low-level, implementation concepts. It would be better to devise a mechanism to hide these data types so that the model code deals with problem domain entities.

3.2. Component-based design and reflection

Component-based design is a technique that many of the newer programming languages support. Models written in .NET or JAVA are constructed of classes that contain not just methods but also properties and optionally event handlers. Frameworks that execute these types of components need to create an instance of the class, set the values of some properties of that instance and then call methods to perform the timestep logic. This approach is sometimes called the Create-Set-Call pattern (Stylos, 2007). However, in a model in which every component will have different properties and methods the framework cannot automatically know what properties need setting and what methods need to be called.

Reflection is one way to alleviate this problem. Reflection allows the framework to discover properties and methods automatically. The model developer inserts metadata into the source code to tag various properties and methods. Rahman et al. (2003) discuss this technique at length and suggest that "metadata frameworks reduce the amount of code required to develop a model and make use of framework features."

The .NET models in APSIM currently use some of the same TIME metadata tags to mark up outputs and event handlers, alleviating the need to call the *expose* and *subscribe* methods of the interface outlined in the previous section. Fig. 5 shows an excerpt from the APSIM SLURP model written in VB.NET. The class variables "int_radn" and "rlv" have metadata added to their declarations. The framework running this model can automatically extract this metadata and expose the variables to other models. Likewise a method (e.g. 'OnNewMet') can be tagged as an event handler. The system can extract this information and automatically call this method whenever a 'NewMet' event is published by the environment model. Table 1 shows a complete list of metadata tags that can be inserted into a .NET model in APSIM. Many of these tags are identical to the TIME framework making it easier to transfer models between the two frameworks.

3.3. Other software development processes

There are many other good software development practices that can aid in the design of a model to assist in exchanging components between frameworks. Jeffries et al. (2001) outline many popular processes including test first development, unit testing, and continuous integration. Design-by-contract, the process where properties and arguments to methods are explicitly tested before

```
<Model()>
Public Class Slurp

  <Output(), Description("Daily Intercepted Radiation"), Units("MJ/m2")>
  Public IntRadn As Single

  <Output(), Description("Root Length Density"), Units("mm/mm3")>
  Public rlv() As Single

    <EventHandler()> Public Sub OnNewMet(ByVal Data As NewMetType)

  . . .
```

Fig. 5. An extract from the APSIM Slurp model (written in VB.NET) showing examples of various APSIM metadata tags (in bold).

Table 1
A complete list of all metadata tags used in APSIM.

Name	Purpose
Model	Specifies that the following class contains the model code. This avoids the need to use inheritance and thus coupling the component to a framework.
Output	Specifies that the following declaration should be made available to output components.
Minimum	Specifies a lower bound for the following variable.
Maximum	Specifies an upper bound for the following variable.
Description	A description of the following variable.
Units	Units of the following variable.
EventHandler	Specifies that the following declaration is an event handler.

and after any calculations that use them, has been used extensively by some modelling groups (e.g. Donatelli and Rizolli, 2007). While these processes are not absolutely necessary, they are recommended as good software practices and will help significantly.

4. Discussion

The previous two sections have illustrated some of the issues faced when bringing models into APSIM and a few software development techniques that have helped this process. Combining these generates a set of principles for model builders.

A model's system boundaries should be carefully considered before and during the construction phase. While this is a very subjective concept, consideration of other models and how they may be interconnected will help with the decision. For example, if the desire is to produce a crop model, decouple the crop growth from the water and nutrient movement. The fact that there are many models in existence for movement of water and nutrients already exist, suggests that a design pattern where the soil processes should be considered external to the crop growth is advantageous. If different soil evaporation models are to be compared, then this would suggest the boundary be placed around the soil evaporation process. The result would be separate soil evaporation, runoff and water movement models or at least sufficiently decoupled sub-models. Quite often the boundaries change over time. While at the outset, the decision to couple crop and water balances may make perfect sense, over time, when interests change, the boundaries inevitably need to be moved to accommodate the new required functionality. This is a normal part of the evolutionary process of every model. It can be circumvented somewhat if a careful analysis of the problem domain is conducted at the outset.

Once the system boundary decision has been made, some form of abstraction is necessary to decouple the model from other models and the environment within which it operates. Even if the model is a standalone executable, abstraction still plays an important role. The sub-models within the executable can still be separated from each other via some interface. Rather than a crop model directly accessing the soil water, it should call a method of an interface to perform the variable access. The primary drawback of such an interface is that the length of the interface and the number of calls to the interface can obfuscate the code, but this is still preferable to tight coupling.

Reflection is another abstraction technique that is available to newer computer languages, namely Java and the .NET languages. It provides a high level of abstraction but has the added benefit of reducing the amount of 'plumbing' that needs to be written. The code is generally easier to read as there are less calls into interfaces, for example, variables are retrieved automatically by the framework. Currently, APSIM uses a hybrid approach of using a 'pull' interface with a 'push' metadata system giving the .NET APSIM models the needed flexibility while removing many of the unwanted calls to the interface. An additional advantage is that various tools and user interfaces can discover this metadata and produce custom documentation or user interfaces. The only minor downside to a reflection approach is the minimal delay, once at initialisation, while the framework discovers this metadata.

No matter which abstraction technique is employed, interfaces or reflection, both should contain metadata describing the model data requirements. Describing the data in the source code has many benefits including increasing the chances of keeping the description and the source code synchronised, the ability to create automatic documentation and the ability for user interfaces to automatically extract the information and use it in different ways.

The APSIM project has taken a simplistic approach to metadata and data matching. Further up the complexity ladder, ontologies have become a popular way to accurately attribute meaning to data. The ontology is used to match inputs with outputs and in some instances is used to convert semantics to enable a match. For some situations this may be necessary. Indeed Argent (2003) suggests that when modelling across disciplines (e.g. environmental management), problems of data meaning are extensive. In these situations the complexity of an ontology may be warranted. For the types of problems that APSIM has been used in, the simpler approach has been satisfactory.

User interfaces should be kept separate from the models. In APSIM there are several user interfaces that can execute the same model. These user interfaces range from a simple command line for doing batch runs, a flexible one for researchers and another much simpler user interface for end-users like growers and agricultural consultants. Coupling a model to a user interface would make this impossible.

The final recommendation is borrowed from the Agile Software Development Community. The Agile Manifesto (www.agilemanifesto.org) emphasises simplicity with a "build for today" philosophy. Jeffries et al. (2001) controversially introduces the acronym "YAGNI: You Aren't Gonna Need It" which describes this tendency to over-engineer a solution. Parts of APSIM exhibit over-engineering where complexity was added to support some perceived future function. That future never materialised and the increased complexity is only now being gradually removed. Hindsight makes it easy to identify unwanted complexity. It is much harder to look forward and envisage all possible uses of a model or piece of source code. Developing for extensibility and reusability often leads to over-engineered models and yet that is something to be avoided. How do we resolve this conflict? How do we develop something that is reusable in different contexts and yet not over-engineer the solution? The solution is to keep it simple. If a model developer finds themselves writing significant "plumbing" or infrastructure code to achieve a more reusable model, then that is a sign of complexity and a simpler solution should be sought. If a model developer finds themselves developing a model that is large and difficult to reuse then perhaps some simple reengineering would be worthwhile, for example, by splitting a large model into smaller models or into simple libraries.

Somewhere on the continuum between complexity and simplicity is a place where model reuse and extensibility can be achieved. We are trying to move APSIM from the complex end of the scale back towards the simple end. We are doing this through substantial and ongoing refactoring (Fowler, 1999) of source code. Our philosophy: if a simpler approach can be found, it should be adopted.

5. Conclusion

For those model builders starting out in a particular problem domain, prototyping is a good starting point. Inevitably, a decision point is reached on how to progress the prototype to the next stage. Models are often more useful when connected to other models, so to enable this, an existing framework should be adopted rather

than creating a new one. Even when the model to be created is deemed to be a research project looking at some small aspect of the system, once the model is published or it is given to other modellers, the decision to use a framework to build the model will be beneficial. The choice of frameworks is often dictated by collaborators or personal preference. Beyond that, comparing the features and support of the many frameworks can help with the decision. A framework that is 'lightweight', meaning one that has minimal overheads and baggage, and one that has a simple model development pathway will be beneficial.

For modellers who are already working in a framework, we recommend a continual process of review and refactoring, looking for ways to simplify. Where there are framework specific entities or technologies, these should be abstracted in some way, using either reflection or an interface. With a mindset of simplification and framework independence, a model builder can produce a model that can more easily be executed in other frameworks.

Framework builders, likewise, need to look to simplify programming interfaces and libraries to aid the model builders. When making significant changes, the preference should be an evolutionary approach.

Sometimes when resources are plentiful, an organisation, or individual, will be tempted to build from scratch rather than reuse and extend an existing solution. Certainly, there are costs associated with collaboration and reuse. In the long-term, not choosing reuse leads to a proliferation of incompatible, but often very similar models. Often a future need emerges for integration of these models into other frameworks, with other models, from other organisations. Indeed, a trend is emerging in the literature of model integration across disciplinary boundaries thus making it even more important for some convergence of models and frameworks.

The way forward is not for all modellers to adopt a single framework or technology although evidence of convergence is already visible between several frameworks (van Evert et al., 2005). This convergence is something that should be encouraged but not at the expense of diversity. Diversity of models and approaches is something to embrace and support. The recommendations outlined in this paper support diversity but at the same time encourage model builders to think about the design and implementation of their models in order to better facilitate model exchange and reuse.

References

Acutis, M., Trevisiol, P., Gentile, A., Ditto, D., Bechini, L., 2007. Software components to simulate surface runoff, water, carbon, and nitrogen dynamics in the soil. In: Proceedings of the Farming Systems Design, Sicily.

Argent, R.M., 2003. An overview of model integration for environmental applications – components, frameworks and semantics. Environmental Modelling & Software 19, 219–234.

Bouman, B.A.M., Kropff, M.J., Tuong, T.P., Wopereis, M.C.S., ten Berge, H.F.M., Van Laar, H.H., 2001. ORYZA2000: Modeling Lowland Rice. International Rice Research Institute, Wageningen University and Research Centre, Los Baños, Philippines, Wageningen, The Netherlands.

Donatelli, M., Carlini, L., Bellocchi, G., 2005. A software component for estimating solar radiation. Environmental Modelling & Software 21, 411–416.

Donatelli, M., Rizolli, A., 2007. A design for framework-independent model components of biophysical systems. In: Proceedings of the Farming Systems Design, Sicily.

van Evert, F., Holzworth, D., Muetzelfeldt, R., Rizzoli, A.E., Villa, F., 2005. Convergence in integrated modeling frameworks. In: Zerger, A., Argent, R.M. (Eds.), MODSIM 2005 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand, ISBN 0-9758400-2-9, pp. 745–750.

Fowler, M., 1999. Refactoring. Improving the Design of Existing Code. Addison-Wesley, ISBN 0-201-48567-2.

Gijsbers, P., Gregersen, J., Westen, S., Dirksen, F., Gavardinas, C., Blind, M., 2005. OpenMI Document Series: Part B Guidelines for the OpenMI (Version 1.0). Edited by Isabella Tindall. Available from: http://www.OpenMI.org.

Hearn, A.B., 1994. OZCOT: a simulation model for cotton crop management. Agricultural Systems 44, 257–299.

Hillyer, C., Bolte, J., van Evert, F., Lamaker, A., 2003. The ModCom modular simulation system. European Journal of Agronomy 18, 333–343.

Jeffries, R., Anderson, A., Hendrickson, C., 2001. Extreme Programming Installed. Addison-Wesley, ISBN 0201708426.

Keating, B.A., Carberry, P.S., Hammer, G.L., Probert, M.E., Robertson, M.J., Holzworth, D., Huth, N.I., Hargreaves, J.N.G., Meinke, H., Hochman, Z., McLean, G., Verburg, K., Snow, V., Dimes, J.P., Silburn, M., Wang, E., Brown, S., Bristow, K.L., Asseng, S., Chapman, S., McCown, R.L., Freebairn, D.M., Smith, C.J., 2003. An overview of APSIM, a model designed for farming systems simulation. European Journal of Agronomy 18 (3–4), 267–288.

Lowy, J., 2006. Programming.NET Components, second ed. O'Reilly Media Inc., ISBN 0596007620

McKeon, G.M., Day, K.A., Howden, S.M., Mott, J.J., Orr, D.M., Scattini, W.J., Weston, E.J., 1990. Northern Australian savannas: management for pastoral production. Journal Biogeography 17, 355–372.

Moore, A.D., 2001. FarmWi$e: a flexible decision support tool for grazing systems management. In: Proceedings of the XIX International Grassland Congress.

Moore, A.D., Holzworth, D.P., Herrmann, N.I., Huth, N.I., Robertson, M.J., 2007. The common modelling protocol: a hierarchical framework for simulation of agricultural and environmental systems. Agricultural Systems 95, 37–48.

Moore, R.V., Tindall, C.I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environmental Science and Policy, vol. 8, Issue: 3. Research & Technology Integration in Support of the European Union Water Framework Directive. In: Quevauviller, P. (Ed.), Proceedings of a Workshop held in Ghent (Belgium) on 4–5 October 2004, pp. 279–286.

Probert, M.E., Dimes, J.P., Keating, B.A., Dalal, R.C., Strong, W.M., 1998. APSIM's water and nitrogen modules and simulation of the dynamics of water and nitrogen in fallow systems. Agricultural Systems 56, 1–28.

Rahman, J.M., Seaton, S.P., Perraud, J.-M., Hotham, H., Verrelli, D.I., Coleman, J.R., 2003. It's TIME for a new environmental modelling framework. In: Proceedings of MODSIM 2004 International Congress on Modelling and Simulation, vol. 4. Modelling and Simulation Society of Australia and New Zealand Inc., Townsville, Australia, pp. 1727–1732.

Robertson, M.J., Carberry, P.S., Huth, N.I., Turpin, J.E., Probert, M.E., Poulton, P.L., Bell, M., Wright, G.C., Yeates, S.J., Brinsmead, R.B., 2002. Simulation of growth and development of diverse legume species in APSIM. Australian Journal of Agricultural Research 53 (4), 429–446.

Stöckle, C.O., Donatelli, M., Nelson, R., 2003. CropSyst, a cropping systems simulation model. European Journal of Agronomy 18, 289–307.

Stylos, J., 2007. Usability implications of requiring parameters in objects' constructors. In: Proceedings of ICSC 2007, Minneapolis, MN, pp. 529–539.

Verburg, K., Ross, P.J., Bristow, K.L., 1996. SWIM v2.1 User Manual. Divisional Report No 130. CSIRO Division of Soils, Canberra, Australia.

Wang, E., Robertson, M.R., Hammer, G.L., Carberry, P., Holzworth, D., Hargreaves, J., Huth, N., Chapman, S., Meinke, H., McLean, G., 2003. Design and implementation of a generic crop module template in the cropping system model APSIM. European Journal of Agronomy 18, 121–140.