Do Not Reinvent The Wheel: Extending The Life Span Of Agricultural Models

Christopher Teh Boon Sung[a,*], Ian Henson[b], Mohd. Haniff Harun[b], Goh Kah Joo[c] & Ahmad Husni[a]

[a] *Dept. Land Management, Uni. Putra Malaysia, Serdang, Selangor, Malaysia*

[b] *Malaysian Palm Oil Board, Bangi, Selangor, Malaysia*

[c] *Applied Agric. Res. Sdn. Bhd., Sg. Buloh, Selangor, Malaysia*

Abstract

The objective of this paper was to discuss several guidelines in model design so that agriculture models are reusable and extendible. These two properties promote further development of a model; thus, increasing the model's life span and its adaptability with changes. An oil palm growth model called SAWIT was used an example. SAWIT was designed specifically to be reusable and extendible by following three principles: 1) decouple the interactions among the model classes, 2) separate the model engine and user interface, and 3) design the model engine to be independent of any hardware and software. SAWIT comprises of several main classes, and these classes were designed to be as self-contained as possible to minimise any dependency or interaction with other classes. To further remove any interaction among classes, the Courier design pattern was used, whereby a mediator (microclimate class) formed a sort of communication centre to keep the other classes from referring to each other directly to obtain any external information. This design greatly reduced the intricate network of interaction among classes and promotes code reuse and extension. SAWIT also had two clear separate sections of code: 1) the model engine (the core of the model that deals with modelling and calculations), and 2) the user interface (a cosmetic, unimportant front for users to interact with the model engine such as data entry). This facilitated model sharing and allows for a situation where the model engine can be easily separated and studied, reused or extended by other modellers.

Keywords: design pattern, model, reusability, extendibility, object oriented, oil palm

Introduction

The model development process is often guided by two goals: to develop a model that is accurate and that remains relevant with time. Of the two, model accuracy is understandably of greater importance. A model is created primarily to solve an immediate problem, and a model is only useful if it solves the given problem with the desired precision. In contrast, however, developing a model that remains relevant with time, or model perseverance, is seldom achieved. Agriculture models often have short life spans. Models are created, used in limited conditions then discarded for newer ones. Moreover, there is a great deal of overlap among agriculture models, where different models often share the same or similar approaches or equations. Consequently, this has led Loomis (1985) and Seligman (1990) to remark the development of agriculture models today are akin to "reinventing the wheel". It would be better instead if a few useful existing models are identified for a given problem, then developed further by various researchers. The scientific community working on a few existing models instead of creating new ones from scratch has several advantages. Fewer models provide focus, avoid duplication of work, and help to speed

[*] *Corresponding author. Tel: +60-3-89466976; fax: +60-3-89434419; Email address: chris@agri.upm.edu.my*

progression of knowledge through the intense scrutiny and sharing of knowledge by workers from various background and experience.

Achieving model perseverance however requires much thought in the model design. Merely producing a working model is not a pre-requisite. Model perseverance is achieved if the model is designed to support two key properties: *reusability* and *extendibility*. Reusability is a property that allows a model to be used to construct another model (Meyer, 1998). This reuse can be in verbatim where the whole or one part of the model is incorporated into another model without change of its original code, or it is modified slightly before being used to develop another model. On the other hand, extendibility is a property that allows a model to adapt to changes (Meyer, 1998). Extendibility means completed models should still be "open for development". Models must designed to be modifiable so that they could be more accurate, efficient or applicable in more diverse conditions. As new knowledge is acquired, models will have to extended to incorporate this new knowledge, or even altered for another approach in solving the given problem. In short, model perseverance can be achieved if models are designed in such a way that allows them to be studied easily and be re-used in other conditions, and models that are adaptable to change (modifications) and further development by other workers.

It is thus the objective of this paper to discuss the principles of designing a stable and organised modelling framework that allows for model reusability and extendibility; subsequently, increasing the model's worth and life span. An oil palm (*Elaeis guineensis* Jacq.) growth model called SAWIT which we developed will be used an example to discuss these principles.

A reusable and extendible modelling framework: The SAWIT design

SAWIT is a model that simulates the oil palm (*Elaeis guineensis* Jacq.) growth and yield, taking into account water and flowering cycle stresses (Teh *et al.*, 2005). This model was designed to overcome several common flaws found in most agriculture models. SAWIT was designed to be reusable and extendible primarily by: 1) reducing the coupling or interaction among the model classes, 2) separating the user interface from the model engine, and 3) creating a model engine that is independent of any hardware and software. SAWIT was written in the C++ programming language, conforming completely to the C++ standards set by ISO (International Organization for Standardization).

Interaction among model classes

It is good programming practice to solve a large problem by dividing it into several smaller and more specific tasks. Consequently, a model is often an assemblage of smaller submodels or components, where each component performs one or more highly specific tasks. Together, these components interchange information, perform their respective calculations, and ultimately solve the given problem.

This interchanging of information means that there is a great deal of interaction or coupling among the components. And it is this interaction among the model components that hinders a model's reusability and adaptability to change. This is because closely linked components create a very intricate network of information flow, where one component not only requires information from other components, but it also supplies information to other components (Figure 1). Consequently, this intricacy creates a rigid model structure, which makes it difficult to modify, evolve and reuse the model for new knowledge or approach. A change in one model component, for example, often breaks the whole model structure because a change produces a "domino effect" where other components are undesirably affected and have to be modified as well. This "domino effect" results from components that are locked together in a pre-determined and inflexible relationship.

The SAWIT model reduces this tight coupling or interaction among components by: 1) developing model components that are self-contained as much as possible, and 2) using the *Courier* design pattern (Gamma and Helm, 1996) which is a variant of the *Mediator* pattern (Gamma *et al.*, 1995).

Figure 2 shows that the SAWIT model comprises of seven components but in contrast to Figure 1, the intricate coupling among the components has been very much reduced. The tortuous many-to-many relationships among the components have now been simplified to a one-to-many relationship in SAWIT. Note: following C++ terminology, the terms “submodels” or “components” will now be referred in this paper as *classes*. Classes that have been instantiated are called *objects*.

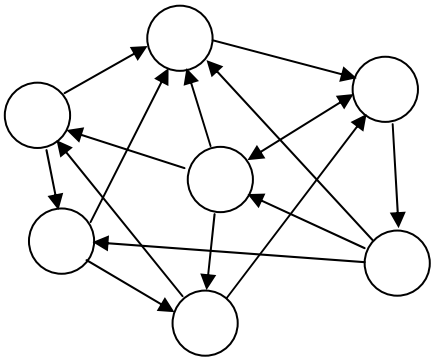

Figure 1. Model components (classes) typically interact with each other, creating an intricate network of relationships. Note: arrows denote flow of information.

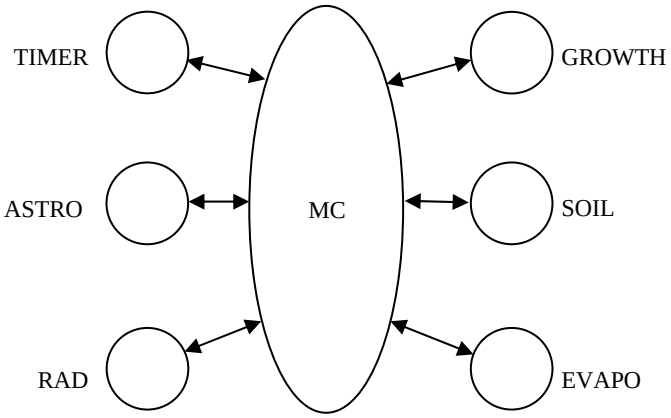

Figure 2. The SAWIT model consists of six classes: timer (TIMER), meteorology (ASTRO), solar radiation (RAD), oil palm growth (GROWTH), soil water balance (SOIL) and evapotranspiration (EVAPO). The seventh class, microclimate (MC), is the mediator for the six classes.

SAWIT comprises of seven main classes and their roles are described in Table 1. These classes have been identified and designed to be as self-contained as possible; that is, each of these classes perform its own unique, specific role and calculations with little help as possible from other classes. Designing each class that is completely self-contained is highly preferable because this will eliminate any exchange of information or interaction among the classes (*e.g.*, Figure 2); thus, making the model highly reusable and extendible. But agriculture objects often interact with each other. Plant transpiration, for example, depends, among others, on the interception of solar radiation and soil water content, where the latter also depends on plant water uptake and amount of solar radiation reaching the soil surface. In short, there is a naturally a great deal of interaction or dependency among agriculture objects which makes it difficult, if not impossible, to create completely self-contained classes. The solution here is then to create and design classes which have clearly-defined, unique and specific roles and they each perform their own roles and calculations as much as possible with minimum dependency on other classes.

Table 1. SAWIT main model classes and the external information they each require

Object	Purpose / description	External information required	Source of information
TIMER	Simulation timer	none (independent class)	none
ASTRO	Meteorological properties such as simulating hourly weather from daily weather data, and calculating day length, solar position, and times of sun rise, sun set and solar noon.	Current simulation time	TIMER
EVAPO	Simulates the daily and hourly latent and sensible heat fluxes in the microclimate system using the modified Penman-Monteith evapotranspiration equation by Shuttleworth and Wallace (1985). Also determines the potential evapotranspiration.	Current simulation time step number, and current simulation time	TIMER
		Daily and hourly weather properties (*e.g.*, solar irradiance, air temperature, wind speed, vapour pressure and saturated vapour pressure), solar position, and reference height	ASTRO
		Leaf area index, and total tree height	GROWTH
		Hourly soil water content, soil water content at saturation, permanent wilting point and critical water point for C3 crops, and soil pore size distribution	SOIL
		Daily and hourly solar interception by the tree	RAD
GROWTH	Simulates the growth and yield of oil palm, accounting for flowering and water stresses. Calculations adapted from models from van Kraalingen *et al.* (1989) and Jones (1997)	Current simulation time step number, current simulation time, total number of time steps, and timer step size	TIMER
		Hourly soil water content, and soil water content at saturation, permanent wilting point, field capacity and critical water point for C3 crops	SOIL
		Daily solar irradiance, day length, site latitude	ASTRO
		Daily solar interception by the tree, and inter-row spacing	RAD
SOIL	Soil water balance	Current simulation time step number, current simulation time, and total number of time steps	TIMER
		Rainfall amount, and daily maximum air temperature	ASTRO
		Daily evapotranspiration	EVAPO

continues

continued

RAD	3-D simulation of the daily and hourly solar interception by the oil palm tree, accounting for direct and diffuse solar components	Current simulation time step number, current simulation time, and total number of time steps	TIMER
		Hourly solar irradiance, solar position, and time of sun rise and sun set	ASTRO
		Leaf area index, and canopy dimensions	GROWTH
MC	Mediator; it prevents classes from calling each other explicitly (*see text*).	none	none

Nonetheless, having near self-contained classes alone is insufficient because interaction among classes still exist. To overcome this problem, SAWIT implements the Courier design pattern. As shown in Table 1, the microclimate (MC) class, unlike the other six classes, does not actually perform any calculations, but its role is vital. It exists to support the Courier design pattern. The MC class acts as a mediator, forming a hub or communication centre which promotes loose coupling by keeping the other objects from referring to each other explicitly. Except for the MC class, the other six classes do not communicate with each other directly, and they are "unaware" of each others' existence. For the six classes, the MC class is their only source for external information.

For example, the solar radiation class (RAD) calculates the daily solar interception by the oil palm (Table 1). To do this, RAD requires the external information regarding the current leaf area index (LAI). The RAD class "submits" a request for this information to the MC class. Being the mediator, MC calls the LAI function found in the growth class (GROWTH). After obtaining this information from GROWTH, MC then relays the information back to RAD, completing the request cycle.

The highlights an important point: that from RAD's perspective, the MC class behaves like a black box, where RAD's requests for external information are provided by the mediator (MC) by means unknown (and unimportant) to RAD. This black box perspective helps the modeller to develop a class to handle a specific task without being distracted or troubled about the implementation details or protocols of other classes.

Moreover, having this simple one-to-many relationship and black box design are crucial because they allow for one class to be modified for new knowledge or approach without breaking the whole model structure. This design additionally allows for one class to be replaced completely by another class. In the current SAWIT model, for example, the hourly solar irradiance is simulated by the meteorology class (ASTRO). In the future, a simpler calculation may be sufficient and the current ASTRO class could be replaced entirely by a new class ASTRO2 (Figure 3). Entire replacements of one or more classes can be done without breaking the whole model structure because, as stated earlier, no two classes (except for MC) communicate directly with each other.

One problem that could arise when a class is replaced completely is that a certain request cycle could become incomplete. The RAD class, for example, also requires information about the solar position to calculate the canopy extinction coefficient. This request is forwarded by MC to a routine in ASTRO. But if the current ASTRO class is to be replaced by ASTRO2 and if this new class does not calculate the solar position, then this particular request by RAD will now fail. Nonetheless, this problem is solved easily by creating a new class (SOLARPOS) which calculates the solar position. The SOLARPOS is then "attached" to the MC class (Figure 3) so that the mediator now calls the function found in SOLARPOS instead of ASTRO as was done before. Moreover, because the RAD

class does not “know” the existence of other classes, the attachment of the new class SOLARPOS and the entire replacement of ASTRO by ASTRO2 do not affect the RAD class in any way. This concept is akin to the “plug-and-play” concept to describe the connection between a computer and its peripherals.

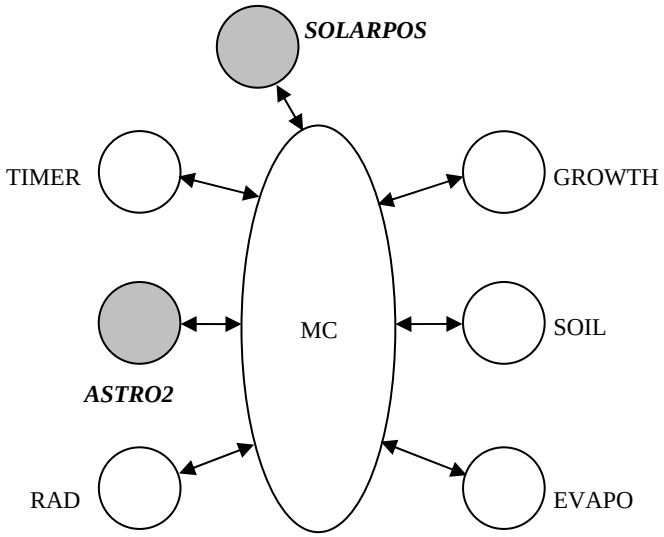

Figure 3. A scenario for SAWIT when the ASTRO class is replaced by ASTRO2. A new class, SOLARPOS, is “attached” to support the request for information on the solar position.

Separation between the user interface and model engine

Another problem often found in agriculture models is the lack of separation between the *user interface* and model *engine*. The user interface in the model is the section of code that deals with user interaction such as producing a graphical-based menu system, complete with buttons and dialog boxes, to receive user selection or input for model parameters, as well as to display or print the model output. The model engine, in contrast, is the section of code that deals with calculations and modelling that form the core and purpose of the model.

When both model sections are kept apart, the model engine can be developed further or reused without the distraction of the user interface. The model user interface is merely a front for the model engine. The importance of separating both sections can be demonstrated with an analogous circumstance of a car. A car comprises of an engine (model engine) and various accessories such as car stereo, lights, leather seats, *etc*. (user interface). The car engine and the various car accessories are cleanly separated; if they were not, replacing the car carburettor, for example, could cause the car stereo to be replaced as well! This scenario is of course bizarre in real life. Unfortunately, this scenario is analogously common for agriculture models. There is commonly an indistinguishable mix between both sections of code dealing with the user interface and model engine. This creates code that is both difficult to understand and modify. But more importantly, a change in the user interface code undesirably affects the model engine code. To achieve model perseverance, it is the model core, or its engine, that would have to be scrutinised and developed further by workers. The user interface is unimportant and merely a cosmetic front for users to interact with the model engine, and keeping this section apart form the model engine greatly simplifies model development.

Recognising this importance, the SAWIT model separates both sections (Figure 4). And because of this separation, it is easy to identify parts of code that has to be modified, and modification does not break the whole model structure. As shown in Figure 4, users will interact with the SAWIT

engine via a user-interface. The user interface of SAWIT was written using the MFC (Microsoft Foundation C++ Classes).

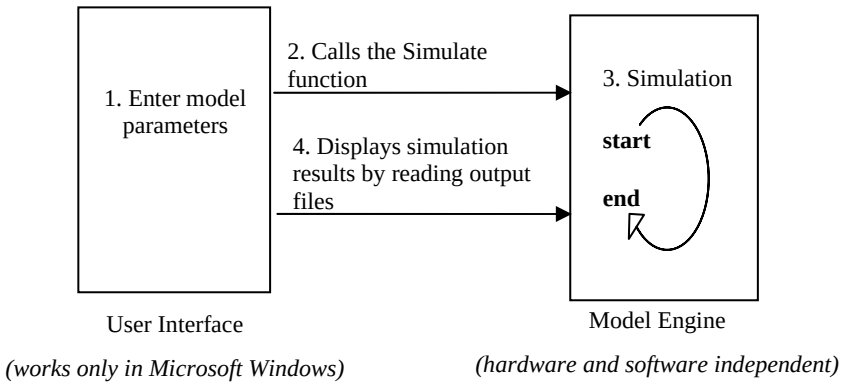

Figure 4. The separation between the model engine and user interface in SAWIT.

Once simulation has ended, the results of the simulation are saved into text files by the model engine. These output files are in turn read by the user interface of SAWIT so that the various values at every time step are tabulated and plotted. Again, this is a clear separation of concerns between the user interface and model engine.

Hardware and software independence

One obstacle to model sharing is when a model is developed unnecessarily to work only on a specific hardware and software platform. A model that runs only on Intel-based PC systems and on Microsoft Windows, for example, hinders collaborative modelling research. Furthermore, some models can only be compiled on a compiler from a specific company only.

These problems happen because modellers often fail to distinguish between the standard and non-standard features of the computer programming language they are using. Popular programming languages used by the agriculture modelling community such as C++, Pascal and FORTRAN are guided by the international standards set by ISO. Conforming models to these standards are essential because programs written following the standards are guaranteed to compile and run on any system or platform.

These standards are followed by all large software companies, but complications arise when these companies often add extensions to the language that are unique only to the company's compiler. For example, "_gcvt" is a C++ function to convert a floating-point value into a string. This function, however, is specific only to Microsoft C++ compilers. Thus, a model that uses any company-specific commands will fail to compile on a compiler from a different company. Likewise, a model that calls or uses any hardware- or software-specific instructions will fail to compile and run on a different platform.

Use of the Java programming language to develop agricultural models is appealing because Java programs can run on any computer and platform. Nonetheless, Java has a serious flaw: programs written in Java are often significantly slower and more memory-intensive than programs written in other languages. Because of this, we do not recommend the use of Java unless it is used for simple and small models (such as without intensive calculations) or when concern for speed is unimportant.

One major advantage of making a model hardware and software independent is the ease of designing the model's user interface. As discussed in the previous section, a model consists of two sections: the model engine and the user interface. By keeping both sections separate as advised

earlier, the model engine can be written using standard language so that it remains hardware and software independent and thus the model engine can be used regardless of the platform (Figure 4). Also recall that it is the model engine section that is of greater interest because this is the section where calculations and modelling are done and the part that has to be modified or reused. The user interface, in contrast, can be written using non standard language because developing a user interface often involves using specific hardware and software instructions (such as accessing the mouse or printer).

In the SAWIT model, the model engine was written in standard C++; consequently, it is independent of any hardware and software. Its user interface, as stated earlier, was written using the MFC to design its graphical user interface. Code that uses the MFC library can only be compiled on a Microsoft C++ compiler, and the user interface can only be used on the Windows operating system.

In short, separating the user interface from the model engine and creating a platform-independent model engine facilitates model sharing and allows for a situation where the model engine can be easily separated and studied, reused or extended by other modellers. A new user-interface for the model engine can be finally developed by these modellers so that their new or modified version of SAWIT can now be run in a different operating system such as Linux, or run over a network.

Conclusions

SAWIT is an example of an agriculture model that has been designed specifically for prolonged life span by supporting model reusability and extendibility by other workers. This is achieved by following three important principles: 1) decouple the interactions among the model classes, 2) separate the model engine and user interface, and 3) design the model engine to be independent of any hardware and software.

The source code for SAWIT is available upon request from the corresponding author.

Acknowledgments

This study was funded by the Ministry of Science, Technology and Innovation, Malaysia, IRPA RMK8 EAR Grant No. 01-02-04-0686-EA001.

References

Gamma, E. and Helm, R. 1996. The Courier pattern. Dr. Dobb's Sourcebook, 3: 55-59.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995. Elements of reusable object-oriented software. Addison-Wesley, Reading, MA, 395 pp.

Jones, L.H. 1997. The effects of leaf pruning and other stresses on sex determination in the oil palm and their representation by a computer simulation. Journal of Theoretical Biology, 187: 241-260.

Loomis, R.S. 1985. Systems approach for crop and pasture research. In: Yates, J. (Editor), Proceedings of the 3rd Australian Society of Agronomy Conference, Hobart, Tasmania, pp. 1-8.

Meyer, B., 1988. Object-Oriented Software Construction. Prentice- Hall, Cambridge, 1254 pp.

Seligman, N.G. 1990. The crop model record: promise or poor show? In: Rabbinge, R., Goudriaan, J., van Keulen, H., Penning de Vries, F.W.T. and van Laar, H.H. (Editors), Theoretical production ecology: reflections and prospects. Pudoc, Wageningen, pp. 249-258.

Shuttleworth, W.J., Wallace, J.S., 1985. Evaporation from sparse crops – an energy combination theory. Quarterly Journal of the Royal Meteorological Society, 111: 839-855.

Teh, C.B.S., Henson, I.E., Harun, H., Goh, K.J. and Husni, M.H.A. 2005. Modelling oil palm growth and yield. In: Teh, C.B.S., Ahmed, O.H., Fauziah, C.I., Izham, A., Wan Noordin, W.D. and Zakaria, Z.Z. (Editors), SOILS 2005: Advances in soil science for sustainable food production. Proceedings of the Malaysian Society of Soil Science 2005. pp. 204-206.

van Kraalingen, Breure, C.J. and Spitters, C.J.T. 1989. Simulation of oil palm growth and yield. Agriculture and Forest Meteorology, 46: 227-244.