Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

SIMULATION SOFTWARE AND MODEL REUSE: A POLEMIC

Michael Pidd

Department of Management Science
The Management School
Lancaster University
Lancaster, LA1 4YX, U.K.

ABSTRACT

Is it really true that simulation models and simulation software should always be regarded as candidates for reuse, or is it better to be selective? What are the obstacles to simulation software and model reuse? Can these be surmounted and, if so, at what cost? There is a range of levels at which simulation software may be reused, a range of costs to be borne and range of benefits that may be achieved. It is crucial to consider the issue of validity when considering model reuse and this needs to be a fundamental part of any reuse strategy. There may be circumstances in which reuse is economic, especially when a small, low-fidelity model will suffice.

1 SOFTWARE AND MODEL REUSE

Software reuse is the isolation, selection, maintenance and utilisation of existing software artifacts in the development of new systems (Reese and Wyatt, 1987). Any survey of current literature will reveal that there are many different approaches to achieving this end. Although much attention has focused on the reuse of source code level artifacts, reuse can be productively applied to all stages of development. This may involve any element of a system, including entities from requirements specification, design, implementation and testing (Reese and Wyatt, op cit). This paper concentrates on code and model reuse, rather than the other elements.

1.1 A Reuse Spectrum

Figure 1 shows a spectrum of different types of software reuse, cast in terms that are recognisable to the simulation community. It shows 4 positions on a very non-linear scale with two different horizontal axes. The first, frequency, indicates that reuse is much more frequent at the right-hand end of the spectrum, where all of us engage in code scavenging. The second axis, complexity, runs in the opposite direction, making the point that code scavenging is relatively easy, whereas successful reuse of entire simulation models can be very difficult indeed.

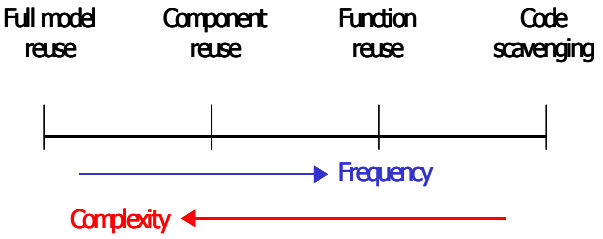

Figure 1: A Reuse Spectrum

- **Code scavenging**: this is the polite description of what all of us do with our computer programs. If we know there is some code that we can use again, probably with some slight modification, then we will do so as long as we trust the person who wrote the code. Since most such scavenged code is reused by the person who wrote it in the first place, then the grounds for trust are either very high, or extremely low, depending on whether you are an optimist or pessimist. Such reuse is fine grained, with relatively small segments of code being employed and modified. Code scavenging is surely uncontroversial and is probably the way that most of us learned how to program and how we go on learning. We take something that appears to work and we use this to do something new.
- **Function reuse**: this is the next step along our spectrum of reuse and, again, this is relatively uncontroversial since we do not think hard about using built-in functions from particular languages or systems (though Pierre L'Ecuyer (2001) has longed warned us about the dangers of doing this with random number generators!). The functions that we reuse in this way are usually very specific in their functionality and are fine grained, which enables us to check that the function is performing as required.

- **Component reuse**: there are many definitions of component, but for present purposes, it is an encapsulated module with a defined interface, providing limited functionality and able to be used within a defined architecture. In a way, a component can be regarded as a function++, since its reuse should offer all the apparent benefits of function reuse, but more so. The is the point on the reuse spectrum where things get a little more tricky, especially as components get bigger and offer broader functionality.
- **Full model reuse**: this has long been the holy grail in some parts of the simulation world, especially when models have been expensively and time-consumingly written and developed. It is, of course, one of the justifications for the HLA, with its desire to support universal and uniform interoperability. The problem to be faced, though, is fundamental and is to do with the nature of simulation modelling.

1.2 Reuse Strategies

No matter which approach is selected, or what size the reusable software artifacts it deals with, it is vital to have a properly developed reuse strategy if any benefits are to be gained from reuse. It has been suggested that a reuse strategy must include features to support the following technical aspects.

- Abstraction – The ability to provide succinct, high level descriptions of reusable artifacts is essential in assisting the developers in understanding their purpose, nature and behaviour.
- **Selection** - To aid a developer in performing reuse, mechanisms that allow the location, comparison and selection of artifacts are also important. In effect, these require directory and search services.
- **Specialisation** - Any technique that maintains collections of reusable abstract artifacts requires features to support the specialisation of those artifacts into useable, concrete entities, since it is unlikely that large grained components can be reused without modification.
- **Integration** - An integration 'framework' is required to provide a mechanism for the combination, connection and communication between reuse artifacts. This must include an agreed architecture within which reuse is achieved.

2 WHAT ARE SIMULATION MODELS AND WHY DO WE BUILD THEM?

A shown in figure 2, a simulation model is a device on which dynamic experiments can be conducted – this definition excludes the use of simulation models for entertainment and for game playing. Thus, the user of the model conducts experiments on the model with a view to understanding what will happen in the 'real' system; this being the one that the model is intended to represent, whether or not it actually exists.

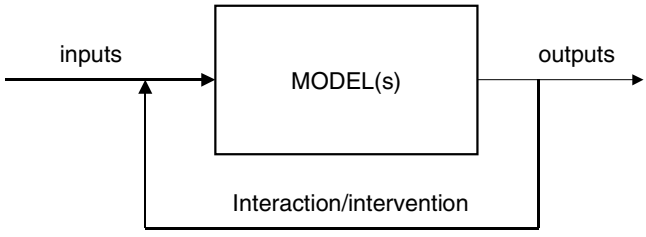

Figure 2: The Essence of Simulation

For the purposes of this paper a computer simulation is the use of a computer-based dynamic model of some referent system. The referent system may or may not exist in full or in part. Such a model, as explained in Pidd (1996) can be defined as follows, "an external and explicit representation of part of a referent system as seen by the people who wish to use that model to understand, to change, to manage and to control that part of the referent system." To become part of a computer simulation, the model must be expressed in a computable form and then run on a suitable computer system. As discussed here, such simulations are dynamic, which means that they are concerned with representing behaviour through simulated time. Simulations are used because they should be cheaper, faster and safer than real experimentation and may be more accurate than mathematical models. The simulation model and its computer-based representations are not ends in themselves but are only means to whatever end is in mind.

2.1 The Problem of Validity and Credibility

With this in mind, the question of model validity looms large and simply cannot be ignored. It seems widely accepted in the simulation community that though models cannot possibly be fully validated, it makes sense to have some form of quality assurance so as to ensure that the model is fit for its intended purpose (Pidd, 1998). This is the most important issue in model validation, not model fidelity. A low fidelity model can be just as useful, often more so, than a high fidelity, very detailed model built at enormous expense.

As a simulation model passes an increasing number of well-designed tests, then confidence in that model should increase, though full validation will not be achieved. Law and Kelton (2000) suggest six classes of techniques that should be considered when attempting to verify a computer simulation program. These tests, and others suggested by Sargent (1988) and Balci (1994) do not demonstrate that a model is valid, instead they are part of a process in which the

credibility of a model is demonstrated. The harder the tests passed by the model, the more credible it becomes. However, just as a Popperian view of science (Popper 1959, 1964) holds scientific progress occurs as theories are refuted and new conjectures emerge, so to there will always be tests that a model might fail. It is important to recognise, though, that a simulation model may still be regarded as useful even though its complete validation is out of reach.

If complete validation of a model of any size is an ideal at which to aim but is, strictly speaking impossible to attain, this surely serves to emphasise that a simulation model should only ever be (re)used for the same purpose for which it was originally constructed. This is possible when a model is used on a routine basis to support tactical decision making within known and defined limits. It is not possible to be sure that reuse is valid when a model is used for a purpose different form that for which it is built or is used in combination with other models that might be based on different sets of assumptions. If a model is to be reused for a purpose other than that for which is its constructed, then it is vital to devise a new credibility assessment process against which the model's validity may be assessed in terms of its new use. Assuming that its credibility will transfer from one application to another is simply not justified.

Proper credibility assessment does not come free and its cost must be built into any estimates of the value of model reuse.

3 WHAT ABOUT THE PEOPLE INVOLVED?

If software is to be reused, there are likely to be at least two groups of people involved, possibly many more. The first group is the initial software developers, the people who write and tested the model the first time that it was used. They may have intended that the model be reused and may have operated accordingly, or they may not have given much thought to this possibility.

The second group is the application integrators, the people who will take an existing piece of software, whether a component or a full model, and will use it in a specific application. This reuse will not be cost-free, for some credibility assessment is essential and, in many cases, the software must be modified in order to reuse it – the exceptions being the routine reuse of tactical models.

The two groups have quite different interests and may be employed by different people and different organisations. A cost for one may be a benefit to the other and this needs to be reflected in the decision to reuse. For example, if a model or component is being developed with potential reuse in mind, then it will be developed with a specific architecture in mind and this will require adherence to the following.

- A set of rules.
- An interface specification.
- Documentation standards.
- A precise definition of the services any implementation must provide.

These costs will be borne by the group that develops the software in the first place, but any benefits will accrue to the group that sets out to reuse the software; whether it be a component or a full model.

4 COSTS AND BENEFITS OF REUSE

4.1 A Simple Financial Model

A simple financial model of the costs and benefits of software reuse can easily be constructed as follows. Suppose that the following can be known.

C = Cost to develop the software for its first use.
A = Cost to adapt for reuse each time its is reused.
N = number of times that the software is reused.
K_N = Average cost/use
Thus, $K_N = (C + A(N-1))/N$

In these terms, reuse N is worthwhile if $K_N < C$.

4.2 Problems with the Simple Model

Obviously this model is too simple, in that the cost of adaptation is unlikely to be constant for each instance of reuse and the cost of reuse includes many elements, including credibility re-assessment and software adaptation. It would also be sensible to apply a discount rate to future costs so as to allow for the time value of money. Nevertheless, the basic model illustrates the usual economic argument made in favour of software reuse.

However, as discussed earlier, the main problem with cost models of this type is that they assume that all the costs are borne by the same group, whereas this often not the case. The costs of adhering to the architecture are part of the cost of initial model and software development and these are borne by the initial developer. The benefits of this, and the correspondingly decreased costs of each instance of reuse, accrue to the reusing group.

On the other hand, the re-users must bear another set of costs, for few large grained components and full models are reused without modification. Some of this modification may be needed if the artifact is to be reused within an architecture for which it was not originally defined. To do this may require the development of wrappers as containers for a piece of code, to make it compliant with a particular architecture, of adapters to bridge between non-compliant components and the rest of the application and of mediators to provide semantic agreement between components that are otherwise compliant.

5 SO, WHAT DO WE CONCLUDE?

The first conclusion is obvious; software reuse has been with us for a long time and shows no sign of going away. The second is equally obvious; to achieve the benefits claimed for reuse requires a properly developed strategy. When we come to model reuse, the picture gets much cloudier, for the question of validity and fitness for purpose loom very large. There is no magic wand that can be waived over a reused model to ensure that it is fit for purpose and proper validation costs time and money.

It is also important to realize that the costs and benefits of reuse are not necessarily fairly distributed. Preparing software for reuse is not free and the costs are borne by the developer. The benefits, though, accrue to the re-user, the system integrator who takes exiting software and models and reuses them.

As a postscript, it is crucial to bear in mind that a high fidelity model is not always better than a simple one. If the simple model is built to answer a specific and important question, then it may be much more useful than one that attempts to represent everything that is going on in a system. Therefore, it may be better not to reuse models on at least some occasions, since it may be cheaper and at least as useful to write a quick and dirty model using a VIMS.

ACKNOWLEDGEMENTS

This paper has benefited greatly from discussions with colleagues in the Lancaster University Computing Department, notably Ian Somerville and Gerald Kotonya and with Noela Oses, a PhD student whom I supervise. Thanks are also due to Dick Nance and Ray Paul for entertaining and stimulating debate on this topic.

REFERENCES

Balci O. (1994) Validation, verification and testing techniques throughout the life cycle of a simulation study. In Balci O.(1994) (Ed) *Annals of operations research, Vol 23: Simulation and modeling.* J.C. Balzer, Basel.

L'Ecuyer P. (2001) Software for uniform random number generation: distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*, eds Brett A. Peters, Jeffrey S. Smith, D. J. Medeiros and Matt W. Rohrer, Arlington VA, Dec 9-12, 2001. pp 95-105

Law AM and Kelton WD (2000) *Simulation modeling and analysis* (3rd edition). McGraw-Hill, Boston, Mass

Pidd M. (1996) *Tools for thinking: modelling in management science.* John Wiley & Sons Ltd, Chichester.

Pidd M. (1998) *Computer simulation in management science* (4th edition). John Wiley & Sons Ltd, Chichester.

Popper K.R. (1959) *The logic of scientific discoveries.* Hutchinson, London.

Popper K.R. (1964) *Conjectures and refutations.* Routledge, London.

Reese, R. and Wyatt, D. L. (1987) Software reuse and simulation, *Proceedings of the 1987 Winter Simulation Conference*, eds A. Thesen, W. Grant and W.D. Kelton,185-192.

Sargent R.W. (1988) A tutorial on validation and verification of simulation models. *Proceedings of the 1988 Winter Simulation Conference*, San Diego, Calif, pp33-39.

AUTHOR BIOGRAPHY

MICHAEL PIDD is Professor of Management Science in the Department of Management Science at Lancaster University. He was President of the Operational Research Society for the years 2000 and 2001. He is best known for two books: Computer simulation in management science (now in its 4th edition) and Tools for thinking (2nd edition in press) both published by John Wiley. He has been an active teacher, researcher and consultant for over 25 years and is particularly interesting in modelling. He holds several EPSRC research grants related to modelling. He is a member of INFORMS and his email and web addresses are `m.pidd@lancaster.ac.uk` and `<http://www.lancs.ac.uk/staff/smamp>`.