Proceedings of the Operational Research Society Simulation Workshop 2025 (SW25)
M. Luis, A. Harper, T. Monks, N. Mustafee, eds.
12th Simulation Workshop, 31 March - 2 April 2025, Exeter, UK

CHIP AWAY EVERYTHING THAT DOESN'T LOOK LIKE AN ELEPHANT

Dr John D Salt

Cranfield Defence and Security
Cranfield University
Defence Academy of the United Kingdom
Shrivenham, SN6 8LA
United Kingdom
j.salt@cranfield.ac.uk

ABSTRACT

This paper addresses the question of how conceptual models are created in a simulation modelling activity. Assuming an entity-based approach to simulation, some techniques for discovering good entity classes are considered, including personation. Also considered are the notations by which a conceptual model can be represented, and the modes of thought required for good conceptual modelling. Specifically excluded from consideration is the idea of applying a cut-and-dried method. The shortcomings of computers for conceptual modelling are remarked upon.

Keywords: Conceptual modelling, Stochastic discrete-event simulation, best practice

1 INTRODUCTION

Making a conceptual model remains one of the most mysterious parts of the art of simulation modelling.

An old joke of indeterminate origin (https://quoteinvestigator.com/2014/06/22/chip-away accessed 20 Oct 2024) has someone asking a sculptor how he produces a lifelike sculpture of an elephant from a block of marble. The answer is to take a hammer and chisel, and chip away everything that doesn't look like an elephant. Such advice may be true, but it is hardly helpful.

The intention of this paper is to offer a few ideas on the means of representation and modes of thought that the author has found to be helpful in developing a conceptual model. It is hoped that these might prove slightly more useful than "chip away everything that doesn't look like an elephant".

2 PREVIOUS RESEARCH

Since being introduced to stochastic discrete-event simulation modelling in 1987, the author has seen little written about the way simulation modellers make conceptual models. This is disappointing; the initial stage of deciding what simulation to write is surely one of the most important parts of any simulation modelling project. The first book on the topic was published by Robinson et al (2010), and Robinson has clearly maintained an interest in the question (Robinson 2011, Robinson 2013, Robinson 2020) and has done a fine job of identifying the existing literature and persuading simulation academics to get together to discuss the matter.

Despite being an academic, the author's interest is more from the perspective of the practitioner than the academic. This paper will therefore offer a few observations on how conceptual models are constructed, based on his own experience over several decades writing simulation models for the defence, oil, transport, and aerospace industries.

3 DISCOVERING THINGS

Practically all the simulation modelling the author has ever done is entity-based. Indeed, the idea of entity-based simulation is so widespread that it might not occur to people that there is any other way of doing it.

3.1 The idea of thingness

Philosophers – specifically, ontologists – have agreed for over two thousand years that the physical universe is made up of things. Typically these things have properties, and may be divided into classes or categories. The idea of thingness extends beyond the physical universe. Abstract objects can exist without physical reality; the idea of a triangle, a Thursday, or a kingdom.

Despite the popularity of this view, it is not the only one possible. In the novel "The Third Policeman", Flann O'Brien (O'Brien 1967) refers to the imaginary substance raw omnium. This is the stuff everything in the universe is made from, but before it is made into things. It seems to me that the task of the conceptual modeller is not unlike sculpting omnium.

Entities have been part of the software representation of simulation models at least since the 1960s. The idea of object-oriented programming using classes (Dahl and Nygaard 1968) was devised specifically for the purpose of creating discrete-event simulation models, a point often forgotten now that it is the dominant approach to software development everywhere. Even earlier, Tocher's idea of activity-cycle diagrams (ACDs) (Tocher, 1963) implicitly assumed the existence of entity classes whose entity life-histories were depicted in the diagrams.

A large part of conceptual modelling, therefore, lies in deciding what classes of things should be present in the model.

3.2 Beings and doings

One of the pioneers of object-oriented techniques in mainstream software development, Peter Coad (Coad and Yourdon 1990) observed a tendency in the analysis phase of projects he had been involved with for the development team to split into two parties, one pursuing initially a functional decomposition, and the other a data-based decomposition. In his experience, the latter was generally a sounder basis for development, but took longer to put together. Usually, therefore, the functional decomposition was pursued, often leading to problems later on.

Not everyone would agree with Coad's observation. There exists a considerable family of purely functional programming languages, such as ML, its offspring, OCaml and Standard ML, and Haskell, Curry, and Clojure. These represent a minority sport in practical computer programming, and as far as I have been able to tell have made little impression on practice in simulation modelling.

Still, simulation modellers, unlike database designers or enterprise architects, cannot get away with just defining what things there are and how they are related. The relations in a simulation model are necessarily dynamic. The data must be made to dance. In the classic object-oriented view, the distinction between "beings" and "doings" is traditionally between instances of classes (objects) and their methods (including multi-methods).

A popular method for discovering candidate classes is known as "grammatical dissection". This is done by taking a short textual description of the system under study, and underlining the nouns. These then become the initial set of candidate classes. Underlining the verbs might be done to give an idea of some potential methods. It seems highly optimistic to rely on a language with such informal grammar as English to make such a distinction. On the one hand, any verb phrase can be replaced by one using a noun ("John gave Mary a kiss" rather than "John kissed Mary"), and, on the other, "there's no noun in the language that can't be verbed". Bertrand Meyer (Meyer, 1997) rightly mocks the naïveté of this approach, and offers more useful advice based on the idea that classes are implementations of abstract data types. What is more, it can be argued that the distinction between an object and a process (which one might think of as representing a noun and a verb respectively) is a false dichotomy. Any entity in your simulation can be thought of as a permanent thing, or as a transient process, depending solely on the timescale under consideration. For example, while it may be unusual to consider a building as a process, rather than an entity, it seems more process-like when

one considers that there was a time when it did not exist, and there will be a time when it does not exist again.

3.3 Discovering classes

Having decided on a set of candidate classes from which one might start to construct a working model, the conceptual modeller(s) will need to decide which to keep, which to discard, and how they interact. A popular way of doing this is by using CRC (Class-Responsibility-Collaborator) cards. CRC cards were originated by Kent Beck and Ward Cunningham (Beck and Cunningham 1989) as a way of helping students to experience an object perspective. They have since proven themselves useful for responsibility-driven design (Wirfs-Brock et al, 1990) and enjoyed a certain amount of popularity with the agile programming movement.

Some of the strengths of the CRC card technique are that it requires minimal equipment (a few file cards and some writing tools), and is intrinsically participative. Since it is an avowedly informal approach, it can be understood by subject-matter experts (SMEs) and operators of the real system who may have no great simulation modelling expertise. This is reminiscent of the HOCUS (Hand Or Computer Universal Simulator) approach devised by Hills (Hills 1968) where ACDs would be developed by hand before being translated into computerized form. The advantage of ACDs is that the execution syntax is more formal than with CRC cards. What matters, it seems to me, is to develop a "cardboard model" in collaboration with SMEs before attempting to implement the thing on a computer.

3.4 The naming of parts

During the process of class discovery, whether using the CRC card technique or not, naming is important. Scott Ambler has described the art of object-oriented programming as extending the programming language into the problem domain. If the people who use the real system do not have a name for something, that thing seems unlikely to be important to the functioning of the system. On the other hand, if they have fine distinctions in naming between things that seem similar, further investigation is merited.

3.5 Personation: the big secret

The idea behind CRC cards, and behind Peter Coad's "Object Game", was to get people to experience an object perspective. This is more like a "first-person shooter" view, from the perspective of a single object, than a "God's-eye view" taking in the whole of the system at once.

I often think that the big secret about simulation modelling is that there is no big secret, but, if there is one, it is this. When acting out the interactions between objects during a CRC card or similar "cardboard modelling" session, the participants pretend to be the objects. This approach was taught to me by Peter Coad in a short course on Object-Oriented Analysis, but I have found no mention of it in the associated book. I believe he called it "personation", and he gave the impression that other analysts used a very similar technique. Whether because it has no agreed name, or because it involves imaginative play that risks appearing childish, I have not seen it referred to in either the simulation modelling or more general software development literature. It would be very interesting to know if other simulation modellers employ such an approach.

4 NOTATIONS

There are a number of diagrammatic notations associated with simulation modelling. ACDs have long been established in the UK, and go back at least to Keith Tocher's foundational work (Tocher 1963). whereas Event graphs (Buss 1996) seem to have been first put forward by Evans, Wallace and Sutherland in 1967 (Evans et al 1967), are now associated mostly with the name of Schruben (Schruben 1983), and have more recently been referred to as Simulation Graphs (Schruben & Yücesan 1993). The author has experimented with Role Activity Diagrams (Holt et al 1983) and Petri nets (Petri 1963), but found neither of them satisfactory.

4.1 OMG standards

The Object Management Group (OMG) has published a standard for the Business Process Model and Notation (BPMN) (https://www.omg.org/spec/BPMN accessed 21 Oct 2024), but I have yet to see it used for simulation modelling.

Better known, for mainstream software development, is OMG's standard for the UML (Unified Modeling Language) (https://www.omg.org/spec/UML/ accessed 21 Oct 2024). This includes a large number of diagram types. Probably the most useful for the simulation modeller are the Statechart, which resembles the traditional State Transition Diagram, and the Activity Diagram, which has execution semantics resembling those of a Petri net, and, given its use of swimlanes rather than overlapping activities, appears to be a rather clumsier version of the Tocher-style ACD. Class diagrams, an extension of the classic Entity Relationship Diagram, are useful for showing the static structure and relationships between classes, but do not help to capture dynamic behaviour.

Before the arrival of the UML, numerous authors had their own diagramming methods for object-oriented software development. Some of these could become quite complex. Grady Booch (Booch 1991) proposed a notation with more than 50 symbol types. There were also various extensions of existing diagramming methods proposed in PhD theses (including the author's). None of these seems to have endured.

4.2 Visual Interactive Modelling: "Visual GPSS"

Probably the most frequently-encountered diagram style in discrete-event simulation is the network consisting of service points, queues, entry and exit points, and resources. Notations of this kind are found in the various VIM (Visual Interactive Modelling) tools such as Simul8 or ExtendSim that offer the user the ability to construct and run simulation models "with no programming" (although I would argue that what is happening is visual programming).

These notations strongly resemble the block diagrams (originally sequence charts) associated with GPSS (Gordon 1969). While this "visual GPSS" approach lends itself admirably to job-shop queueing systems that are suitable for modelling lots of problems in transport, manufacturing, and telecommunications, the forced distinction between temporary entities (transactions) and permanent entities (service points and resources) makes it hard to model multi-party collaborations, something at which ACDs excel. It is a mystery to me why, with so many VIS tools on the market, none seems to implement the ACD approach.

4.3 Others

Still other notations have been suggested for conceptual modelling. Influence diagrams, ideally suited to systems dynamics modelling, seem to me to be ill-adapted to entity-based modelling. Rich Pictures, from the Soft Systems Methodology (SSM), may prove useful because they do not limit what the modeller includes in them, but, lacking any specified execution semantics, they may end up producing undesirably static models.

Some software development notations seem to me to be quite unsuitable. The Input-Processing-Output (IPO) approach embodied in Data Flow Diagrams (DFDs) has never struck me as a natural way of conceptualizing systems. The fact that Ed Yourdon, the person most associated with popularizing DFDs, abandoned them for an object-oriented approach suggests that better things are available. Control flow charts have scarcely been used in software development for many decades now, but remain unaccountably popular with some managers. Teaching on the Discrete and Continuous Simulation course at Shrivenham, I have noticed that a fairly common mistake among students is to try to specify the behaviour of a simulation using a traditional flowchart. This usually works very badly. The emphasis on branching decisions is less likely to be of interest in a simulation than questions of timing and randomness, neither of which a control flowchart can show. Flowcharting does not address the question of what entities exist, cannot deal with several things going on at once, and implies the God's-eye view that is seldom helpful to the modeller.

5 MODES OF THOUGHT

The trick a simulation modeller is trying to pull off is to find and express a useful correspondence (or metaphoresis) between the problem-owner's system of abstractions and some set of precise and executable mathematical constructs. This is not a trivial task of transposition, but, in the same way as translating from one human language to another, it is often perceived as such and seen as trivial by those with no skill or experience in it. This failure to regard simulation modelling as a distinct skill is a failing I have seen have dire effects on a simulation project. Ed Russell listed the skills necessary to a successful simulation project as being software development, project management, subject matter expertise, and modelling skill (Russell 1983). Of course, I am not a disinterested party in the matter, but it seems to me that modelling skill should command greater regard, and of course more lavish emoluments. Certainly nobody would be silly enough to attempt a simulation project with no attempt to provide for any of Russell's other categories of needful skill.

In order to do an adequate job of conceptual modelling, it seems to me that the simulationist should cultivate certain habits of mind.

5.1 Simplifying

The essence of simulation modelling is simplification. The modeller therefore needs to be able to distinguish those parts of the system under study that can be idealized, or left out entirely. The more experienced the modeller, the greater is likely to be the confidence with which they will make sometimes quite brutal simplifications. It is always easier to add complication to a model than to take it out. I have seen simulation models fail because they were too complicated; I cannot recall having seen one do so because it was too simple.

It is a commonplace that simulation modelling is used to address "what if?" questions. I suggest that the conceptual modeller should often ask themself "what if not?" – what would happen if we left something out? This is in accordance with the principle enunciated by Saint-Exupéry (Saint-Exupéry 1939): *"Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher"* (It seems that perfection is attained, not when there is nothing left to add, but when there is nothing left to take away). Economy of expression is much to be desired. Simulationists should be software poets, rather than software novelists.

One great aid to simplifying is a clear idea of the ultimate purpose of the simulation study. Here the old cyberneticist's question, "it is a machine to do what?", can be very helpful. If the modeller is unclear as to what the simulation is supposed to achieve, it becomes impossible to tell whether any particular element is or is not important, so it is included just in case, and the model grows like Topsy. As John Sterman puts it (Sterman 1991), "The art of model building is knowing what to cut out, and the purpose of the model acts as the logical knife".

5.2 Clarity and concretion

It seems to me that simulation modellers, at least for entity-based simulation, must of necessity be fairly concrete thinkers. Not for them the numinous and grandiloquent utterances of managers seeking to impress. For a conceptual model to be useful, it is necessary that all parties understand its elements in the same way. While it might be tempting to let people understand things each in their own different way for the sake of a quiet life, things will go badly when the disagreement is detected.

I try to impress upon my students that there are often times when it is more important to be clear than to be correct. If you are clearly in error, the error can then be corrected; if it is not clear what you mean, such correction might be deferred indefinitely.

5.3 Tolerance of uncertainty

It is a natural human tendency to desire psychological closure. Once things are done and dusted, one can stop worrying about them. This is the sort of thinking the conceptual modeller should avoid.

Here I am not thinking of the aleatory uncertainty of randomness, although of course simulationists should be comfortable with that. Rather, it is the suspension of final judgement on what is the right way to proceed. Even at the conceptual modelling stage, simulation modellers should

expect to use an iterative approach that requires re-visiting elements of the model at frequent intervals. Conceptual modellers should have the flexibility of mind to consider several different representations of the system under study, to think in different categories.

Note that this point is not in conflict with the previous one. It is possible to be clear, concrete, and provisional, just as it is possible to be vague, abstract, and final.

5.4 Dealing with ignorance

Simulation modellers often need to become instant experts in areas that are new and strange to them. Obviously this needs them to be quick learners, and it has already been mentioned that it is a good idea to try to learn and speak the language of the problem domain. It also means that they need to become accustomed to being ignorant.

One aspect of wrestling with one's ignorance is a willingness to ask questions which may, in retrospect, sound stupid. Better to expose your ignorance with the stupid question than to preserve it, and attempt to preserve your dignity, by not asking.

Another natural human tendency is to stick to what one knows, and it can be tempting to concentrate on those parts of the system under study that are well understood, and for which the best data is available. This is likely to prove a mistake: the greatest benefit from a simulation study will come from improving our understanding of poorly-understood areas. Too often have I seen students scope a simulation model so as to exclude matters they were not comfortable with.

5.5 Spirit of play

The technique of personation requires playing games of make-believe in a manner easily seen as childish. Collaborative sessions creating cardboard models, using CRC cards or otherwise, therefore run best if animated by the spirit of play. If everybody is at play, then they will feel comfortable exposing outlandish creative ideas, and have an environment in which it is safe to explore, and safe to fail.

Although it probably does not count as a business benefit, this will also make conceptual modelling a lot more enjoyable.

6 METHODS AND MACHINES

It will not have gone unnoticed that I have not recommended either a method for the construction of conceptual models, nor any computer software to assist with it.

6.1 Methods

I do not believe that it is possible to devise a fixed method to carry out creative, artistic work such as conceptual modelling. The idea that methods can be prescribed for activities of all kinds smacks of a Fordian attempt to de-skill creative workers. This is defined as "creationism" in the Jargon File (http://www.catb.org/jargon/html/C/creationism.html accessed 21 Oct 2024) in the following terms:

> *The (false) belief that large, innovative software designs can be completely specified in advance and then painlessly magicked out of the void by the normal efforts of a team of normally talented programmers. In fact, experience has shown repeatedly that good designs arise only from evolutionary, exploratory interaction between one (or at most a small handful of) exceptionally able designer(s) and an active user population — and that the first try at a big new idea is always wrong. Unfortunately, because these truths don't fit the planning models beloved of management, they are generally ignored.*

As Albert Camus (Camus, 1956) had it, *"Quand on n'a pas de caractère, il faut bien se donner une méthode"* (when one has no character, one is obliged to resort to a method). Conceptual modellers should have sufficient character not to need a method. A more recent aphorism expressing the same thought comes from Nassim Nicholas Taleb (Taleb 2016), who says "Your brain is most intelligent when you don't instruct it on what to do."

6.2 Machines

Given these beliefs, it will come as no surprise that I do not think a computer will help much in the conceptual modelling phase. While Hill's HOCUS was born at a time when extensive paper-based preparation was done for all programming activities before consuming any expensive computing resources, the idea of restricting conceptual modelling to paper or cardboard media remains a good one. For one thing, paper or card models are much easier to throw away and re-do. Also, despite the advances in other aspects of performance, computers remain miserable at allowing more than a couple of people to work on the same task simultaneously. Where a group of people might be able to conduct a conceptual modelling session with rich multi-party interactions between all members of the group, introducing a computer screen into the proceedings will transmute the richness of many-to-many human interactions to a limited number of human-to-computer interactions. This is not what we want.

It must be admitted that simulation modelling has pioneered the application of visual programming in VIM. However, the universality of the "visual GPSS" paradigm means that the conceptual modelling has largely been done beforehand. This is bad luck if the model you need does not fit neatly into the limited "job shop" view.

7 CONCLUSION

Conceptual modelling does not appear to be very well understood, even by acknowledged experts in simulation. It does not seem to be accorded the same respect by management as other skills. As successful conceptual modelling seems to depend on freely admitting to ignorance, and on a child-like spirit of play, this is perhaps unsurprising. Popularity with management is not helped by the fact that conceptual modelling is not amenable to automation or codification. I would, however, contend that it is the most critical skill needed to conduct a useful simulation project. Without a good conceptual model, the rest of the project will be built on a base of sand.

ACKNOWLEDGMENTS

The author would like to acknowledge a vast debt to the profound thinkers who have influenced his thinking about simulation modelling over the last 37 years, including notably Prof Isi Mitrani, Colin Grey, the late Jim Barr, Chris Carrigan, Ron Belanger, Peter Coad, Prof Mike Pidd, Dr Paul Syms, Phil Barker, and most of all my good friend the late Prof Ray Paul.

For the purposes of open access, the author has applied a creative commons attribution (CC BY) licence to any accepted author manuscript version arising from this submission.

REFERENCES

Beck, K and Cunningham, W (1989). A Laboratory For Teaching Object-Oriented Thinking. In: Meyrowitz, N K (ed), *OOPSLA'89 Conference Proceedings,* New Orleans, LA, pp. 1-6.

Booch G (1991). Object Oriented Design with Applications, Benjamin Cummings.

Budd, T (1994). Multi-paradigm programming in Leda, Addison-Wesley.

Buss, A H (1996). Modeling with Event Graphs. In: Charnes, J M, Morrice, D J, Brunner, D T, and Swain J J, *Proceedings of the 1996 Winter Simulation Conference*, San Diego, CA, pp. 153-160.

Camus, A (1956). La Chute, Gallimard.

Coad, P, and Yourdon, E (1990). Object-Oriented Analysis (2nd ed), Yourdon Press.

Dahl, O-J, and Nygaard, K (1968). Class and subclass declarations. In: Buxton, J N (ed), IFIP Working Conference on Simulation Programming Languages, North Holland.

Evans, G; Wallace, G; Sutherland, G (1967). Simulation Using Digital Computers", Prentice-Hall.

Gordon, G (1969). System Simulation, Prentice-Hall.

Hills, P R (1968). HOCUS: A Simple Approach to Simulation. In: *Data Processing*, May 1968.

Holt A W, Ramsey H R, and Grimes J D (1983). Coordination System Technology as the Basis for a Programming Environment. Electrical Communication 57-4, pp. 308-314.

Meyer, B (1997). Object-Oriented Software Construction (2nd ed), Prentice Hall.

O'Brien, F (1967). The Third Policeman, MacGibbon and Kee.

Petri, C A (1963). Fundamentals of a Theory of Asynchronous Information Flow, 1st IFIP World Computer Congress, Munich 1962, North Holland, pp. 386-390.
Saint-Exupéry, A (1939). Terre des Hommes, Gallimard.
Robinson S, Brooks R, Kotiadis K, and van der Zee D-J (eds) (2010). Conceptual Modelling for Discrete-Event Simulation, CRC Press.
Robinson S (2011). Choosing the right model: Conceptual modeling for simulation. In: Jain S, Creasey R R, Himmelspach J, White K P, and Fu M (eds). *Proceedings of the 2011 Winter Simulation Conference.* Piscataway, NJ, pp. 1428–1440.
Robinson S (2013). Conceptual modeling for simulation. In: Pasupathy R, Kim S-H, Tolk A, Hill R, and Kuhl M E (eds). *Proceedings of the 2013 Winter Simulation Conference.* Washington DC, pp. 377-388.
Robinson S (2020). Conceptual modelling for simulation: Progress and grand challenges. Journal of Simulation 14(1): 1-20.
Russell, E C (1983). Building simulation models with SIMSCRIPT II.5. CACI, La Jolla, CA.
Schruben, L (1983). Simulation Modeling with Event Graphs, Communications of the ACM, 26(11), pp. 957–963.
Schruben L, and Yücesan, E (1993) Complexity of simulation models: a graph theoretic approach, In: Evans, G W, Mollaghasemi, M, Russell, E C, and Biles, W E (eds), *Proceedings of the 1993 Winter Simulation Conference.* Los Angeles, pp. 641–649.
Sterman, J D (1991). A Skeptic's Guide to Computer Models. In: Barmey, G O, et al (eds.), Managing a Nation: The Microcomputer Software Catalog, Westview Press, pp. 209-229.
Taleb, N N (2016). The Bed of Procrustes: Philosophical and Practical Aphorisms, Penguin.
Tocher, K (1963). The Art of Simulation, English Universities Press.
Wirfs-Brock, R, Wilkerson, B, and Wiener, L (1990). Designing Object-Oriented Software, Pearson.

AUTHOR BIOGRAPHY

JOHN D SALT received a BA (Comb Hons) in French and Russian from the University of Exeter in 1982, a MSc in Computing Science from the University of Newcastle-upon-Tyne in 1988, and a PhD in Computer Science from Brunel University in 1999. With over 35 years of experience as a simulation practitioner, he has created and maintained simulation models on a great variety of topics for organisations such as Hunting Engineering Limited, Eurotunnel, the European Space Agency, Shell International, Saudi Aramco, The Norwegian Defence Research Establishment, General Dynamics UK, DERA and DSTL. He is a member of the SSAISB, ACM, MCA, UKSS and RUSI, and is a Fellow of the Operational Research Society. He expects to retire next year from his post as a lecturer in simulation, OR and analytics with Cranfield Defence and Security at the Defence Academy of the UK, Shrivenham.
https://www.cranfield.ac.uk/people/dr-john-salt-493715

Chip away everything that doesn't look like an elephant

Salt, John D.

2025-04-02

Salt JD. (2025) Chip away everything that doesn't look like an elephant. In: Proceedings of the Operational Research Society Simulation Workshop 2025 (SW25), 12th Simulation Workshop, 31 March - 2 April 2025, Exeter, UK

https://doi.org/10.36819/sw25.014