Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

XCELL: A CELLULAR, GRAPHICAL FACTORY MODELLING SYSTEM

Richard Conway
William L. Maxwell
Cornell University
Ithaca, NY 14853, U.S.A.

ABSTRACT

XCELL is one of a growing number of new simulation tools that capitalize on the graphics capability of personal computers. It is a complete, self-contained, interactive system intended for use by non-programmers. Graphic representation is the fundamental medium in XCELL, used in the construction of the model as well as the display of results.

1. INTRODUCTION

The simulation tools available to model manufacturing systems have been improving rapidly in recent years, after a long period of relatively slow progress. Realistically, a large fraction of this improvement is directly attributable to progress in computer hardware, which has inspired new approaches, as well as made old methods more cost effective. This help is coming none too soon, since the pace of facilities design and rennovation is steadily quickening. We no longer have the luxury of time to tune and debug new manufacturing systems on the floor, since the expected economic life of a new system, before major revision will be required, has become frighteningly short.

Simulation tools are progressing on two distinctly different fronts, but failure to recognize the distinction between the two is causing some confusion. The first front is the important task of increasing the productivity of the simulation professional. The second is the campaign to make some reasonable form of simulation available to the non-professional -- presumably to the person who is directly responsible for the solution to the problem. The confusion arises in the fact that progress on both fronts is sought by making tools that are "easier to use". The difficulty is that this phrase has quite different meanings in the two different contexts.

To make a tool "easier to use" for a professional, one seeks to increase the power of the available constructs. This is often achieved by increasing the variety of constructs available, as well as increasing the flexibility of particular constructs. In effect, it constitutes increasing the richness of the "language", in both breadth and depth, and if this increases the complexity of the tool, and the difficulty of achieving competence, that is an acceptable trade-off.

On the other hand, making a tool "easier to use" for the non-professional respresents almost a diametrically opposed task. The dominant criteria must be ease of understanding and ease which one can acquire a useful degree of competence. Richness in this context is generally counter-productive, and complexity must be carefully hidden. In particular, efforts to increase the modelling productivity of the non-professional, apart from the learning time required to get started, are faced with special difficulty. It may well be that with the simple "hand tools" that are safe and productive for the amateur, it takes longer to accomplish a given modelling task than with the "power tools" available to a professional.

Improving simulation tools for either audience alone is a challenging task. For a particular contribution to be valuable to both is unusual. If one is unclear which audience is the intended beneficiary of a new development, then prospects are unnecessarily jeopardized.

The one path that offers some hope of progress on both fronts is the development of special-purpose systems for particular industries or particular problems. In effect, by "building-in" some knowledge of the problem, the amount that the amateur must learn is reduced, at the same time that the amount that the professional must do is reduced. However, the price is obviously loss of generality, and limitation of the potential market. Again, if one does not clearly understand the benefits and hazards of this path, the prospects of success are limited.

One deceptively attractive path toward serving both audiences is a "multi-layered" toolkit: the amateur uses a subset of the full professional toolkit. Its primary virtue is the relatively smooth transition as the amateur is seduced into becoming a professional -- that is, as the technique becomes more interesting than the problem. As reasonable as it may seem, an effective layering is extraordinarily difficult to achieve, and it is arguable that the result usually serves neither audience as well as a system designed for one or the other. The same worthy objective has entranced the general-programming-language community for years, but success there too has been limited. It is simply very difficult to devise a language where unseen features do not intrude upon an introductory subset, and/or the constraints of a usable subset do not distort the structure of the superset. For example, in the computer science community, the consensus seems to be that a subset of a rich production language (e.g. PL/I) is a less desirable instructional vehicle than a frugal language designed expressly for that purpose (e.g. Pascal). Similarly, a superset of Pascal is less effective for sophisticated contemporary programming tasks than a language conceived with a much broader objective (e.g. Ada).

The tantalizing parallel for those who would develop simulation tools for the amateur is, of course, the financial spreadsheet. A comparably manageable simulation tool, that would drastically expand the potential class of users, would be both of great service and great profit. The catch may turn out to be that simulation is inherently a much more complex task than financial projection, and there may be a lower bound on the complexity that can be achieved in a simulation system of useful flexibility that is substantially above that of spreadsheet systems. It will be some time before the truth of that proposition can be known, and in the meantime the enthusiastic pursuit of the spreadsheet counterpart will leave the simulation field much the richer -- if not the development entrepreneurs.

2. GRAPHICAL, INTERACTIVE, MICROPROCESSOR-BASED SYSTEMS

A consensus has emerged regarding at least four characteristics of potential simulation tools for amateurs. (The same consensus may apply to tools for professionals, but they are more tolerant, and that is another story.)

1. They should be oriented to graphics, rather than text.
2. They should be more or less interactive in nature.
3. They should be "menu-driven", rather than involve conventional "programming".
4. The host system should be a single-user microprocessor-based system -- a "personal computer" or a desktop work-station.

To some extent, the fourth item is simply a consequence of the demands of the first three, and perhaps also just testimony to the significant power that is available in such systems today. However, there is probably also some psychological aspect to the attractiveness of microprocessor-based systems -- a do-it-yourself simulation system just seems more appropriate on a run-it-yourself computer system. Be that as it may, this poses little real limitation today, for the power of an M68020-based or an I80286-based system compares very favorably in both memory size and processing power with the partition of a time-shared mainframe that was the standard simulation engine of the 1970s.

The position of graphics in the consensus is also probably part logic and part emotion -- the "picture is worth a thousand words" argument is persuasive, but graphical systems are also just a whole lot more fun than studying columns of numbers. Graphics in simulation has pretty much come to imply animation, rather than the bar charts and pie charts of business-graphics packages. Animation is an eminently reasonable way to observe the behavior of a dynamic system, but after the first rush of entrancement wears off, it slowly becomes obvious that animation is a less than effective way of summarizing behavior, so that something more is required.

There is also the issue of whether the graphics in a simulation system is pervasive and fundamental, or is essentially an optional way of presenting a file of event-oriented data. A truly graphical system will employ this medium in the construction of the model, and not just in the presentation of results. The issue is one of degree, but there is a subtly different feeling between a textual system supplemented with graphics, and a graphical system supplemented by text.

"Interactive" means different things to different people. As a minimum, today, an interactive system will check input immediately upon entry. For example, a detectable structural error in building a model will be protested immediately on commission, and not sometime later in a distinct "checking" phase. This is directly comparable to the immediate syntax-checkers of modern program development environments (Henderson, et al., 1984), in contrast to the complete-program-check-during-compilation paradigm of classical batch-processing compilers.

"Interactive" can also refer to the ease and speed with which the user can pass between different phases of the system. (Strictly speaking, in the programming environment world, systems are said to be "well integrated", or "tightly coupled" if the transition between editing and executing a program is immediate and painless.) From the user's point of view, it is certainly preferable to be able to move freely between the construction and execution of a simulation model, without any concern for or awareness of the structure of the implementation of the tool. For example, it is desirable to be able to change the structure of a model in the middle of a run -- that is, during a pause in the run, but without the necessity of restarting the run from the beginning.

"Menu-driven" is also a somewhat ambiguous term, involving both the nature of the actions that the user takes, and the manner in which the choices are presented. The former is the "programming issue" -- is the user really dealing with the objects of a programming language, or the components of a manufacturing system. (The distinction can actually be very subtle -- but if there is anything at all that smells like a loop, or a conditional, or a subroutine, then you are programming, no matter how well disguised.) The presentation issue is quite separate. In one form or another, a simulation system offers the user a repertoire of different "commands". In a "command oriented" system the user must learn the list of commands (or keep a "User's Guide" close at hand) and enter the commands textually. Alternatively, in a menu-driven system, commands are typically grouped in a hierarchical structure and presented to the user a few at a time. The user may then select with a pointing device (a mouse, lightpen or touchscreen), a special-function key, or a letter or number key corresponding to position in a list on a menu-screen. Although the repertoire of commands may be exactly the same in both cases, there is growing preference for the menu-driven alternative as being more appropriate to the amateur user. (It can also be painfully tedious for an experienced user.)

3. THE XCELL FACTORY MODELLING SYSTEM

Against this background it is easy to characterize XCELL. It is an entirely graphical system, well-integrated and fully interactive, that is intended unequivocably for amateur modellers of manufacturing systems. XCELL is function-key menu-driven without any special "menu screens". (The softkey labelling of the function-keys is always present as the current menu.) XCELL was designed and implemented by a group with a background in manufacturing, who had spent a long sabbatic building program development environments, and who suddenly realized that this same approach could be applied directly to simulation.

The design of XCELL began with a clean slate, with no obligation to be compatible with another system, or to extend the life of an existing batch-oriented, mainframe system. XCELL has been three years in development, and the current version is the third major generation of the system. XCELL has evolved in a continuous dialog with the staff of the Manufacturing Research Center of Hewlett Packard Laboratories, and although the developers are associated with a university, XCELL was designed from the outset to be a practical tool for real-world manufacturing, rather than a teaching tool.

Although the third generation XCELL is enormously more powerful and flexible than we would have thought possible at the outset, it is nevertheless far from a general-purpose simulation tool, and it cannot model every manufacturing system. XCELL's objective remains the modelling of a usefully large class of manufacturing problem, with a user-interface that can be mastered in a few minutes. As one can imagine, as the use of XCELL has spread we have been deluged with impassioned requests for additional features. We are perhaps proudest of the manner in which we have filtered these requests to separate the broadly useful from the idiosyncracies of particular problems, and have steadfastly defended XCELL from being smothered in special features of limited utility. As new features have been added to XCELL, the old ones have been refined and simplified, and the current version is actually easier to use than the first. As must be readily apparent from general observation, it is all too easy to double the complexity of a computer system in extending its applicability by only a small amount. Repetition of this folly is inevitably destructive.

In retrospect, XCELL undoubtedly has benefitted from the initial decision to stay within the limits of relatively low-performance, low-cost graphics systems. Although the appearance of high-resolution, three-dimensional, color animation systems is most impressive, the ability to run on low-cost, widely-available computers that do not have to be specifically acquired for this one purpose is not unimportant. If XCELL appears to be at a visual disadvantage in showing off demonstration models carefully prepared in advance, the real payoff is in the speed and ease with which the viewer can construct and modify his own models -- and XCELL is without equal in this respect.

Another critical design decision that contributes significantly to the conceptual simplicity and consistency of XCELL is the representation of the factory floor as a rectangular grid of uniformly sized cells -- hence the designation as a "cellular" approach to simulation. Every component of an XCELL model logically (and graphically) occupies one cell, and each cell can hold at most one component. The result is geometrically distorted, since distances in the model display are not proportional to real distance (and are not significant in the operation of the model). Similarly, the component symbols represent their functional identity as XCELL components, with no attempt to graphically depict the size or shape of the real system counterpart. This is certainly a major advantage in the implementation of XCELL, and we believe that the user benefits as well. We have observed no difficulty whatever in users' dealing with the somewhat distorted image of the XCELL display, and have often been pleasantly surprised when a fresh observer recognizes the real system being modelled on the XCELL screen.

The key to the implementation of XCELL is the fact that there is really only one model, and that is already built-into the system. While the user is carefully lead to believe that he is creating new components and positioning them on the factory floor, in reality he is simply moving components that already exist to a visible part of the display. Each component is already completely populated with default attributes -- the user is only selectively changing these values. The user is literally just changing values in various attribute tables, under the control of an editor that limits entries to valid values. This makes it relatively easy to ensure that the model is always logically consistent. It also means that a model is always executable, since although the user may consider the model incomplete, XCELL really always has a complete model to run. Similarly, this makes it relatively easy to allow the model to be changed during a pause in a run without requiring the run to be restarted.

From a computer science point of view, XCELL is an interpretive rather than a compiled system, and it enjoys the diagnostic and interactive advantages typically associated with that form of implementation. Presumably it also suffers the relative performance penalty of interpretive execution, but there is little opportunity for direct comparison in this regard, and execution speed has not been a significant problem.

4. THE BASIC XCELL COMPONENTS

Probably the most important decision in the design of a simulation system like XCELL is the choice of the basic component types, and we did not get this right at first. The set that has evolved after a good deal of trial and error is the following:

The "physical" component types that occupy a cell are the following:

Workcenter -- the primary active element; the limiting resource

Buffer -- a finite-capacity storage element interposed between Workcenters

Receiving Area -- a source of input material from outside the model

Shipping Area -- a sink from which finished material leaves the model

Maintenance Facility -- a source of a type of "repair" resource for Workcenters

The "logical" component types that are superimposed upon the physical types are the following:

Process -- an activity that takes place at a Workcenter. A Workcenter can perform a variety of activities, but only one at a time

Stock -- the quantity of a particular "part" in a Buffer

Link -- a "path" over which material flows from one cell to another.

There are many ways in which this collection of component "types" could be expanded or contracted, and we have tried many variations. We certainly cannot prove that this is the best set of choices, but is has stood the test of use well, and it has been some time since we were seriously tempted to add or eliminate a type. In principle, any manufacturing system can be modelled as a network of queues, and in fact, there are simulation systems that take exactly that view, and have an elegantly frugal set of types. But in practice, we believe that XCELL's choice provides a more natural modelling kit for manufacturing problems, and we attribute much of the conceptual simplicity of the system to this choice.

On the other hand, certain types of asynchronous materials handling systems -- automatic guided vehicles, for example -- are not conveniently modelled by these components. We are currently trying to figure out what additional types of components will most effectively extend XCELL's scope to cover such systems.

5. CONCLUSIONS

XCELL clearly seems to represent significant progress in the campaign to provide simulation tools for amateur modellers -- in particular, for non-programmers. There is no question that it is readily mastered and easily used. The real issue is just how rich a class of manufacturing problems it encompasses. With as much objectivity as we can muster, we must conclude that opinion is still divided on the general utility of such systems.

If the current version of XCELL itself does not yet constitute a "spreadsheet breakthrough" in simulation tools, we no longer believe that there is a practical limit to the power of this approach, and each new generation represents a major expansion in capability. Moreover, many other investigators are pursuing similar approaches and rapid progress in the next few years is certain.

Unfortunately, the industrial use of XCELL has revealed a somewhat sobering fact that has heretofore been obscured by the difficulty of constructing simulation models. That fact is that the modelling process itself, quite apart from the task of implementing the model, is a conceptually difficult undertaking. The ability to effectively abstract the essential logical core of a large "noisy" system is not widely or uniformly distributed over the population of engineers and managers. The insight required to identify critical issues and formulate viable solutions is not commonplace. Professors, of course, are inclined to believe that these characteristics can at least be improved, if not created, in students. Nevertheless, this suggests that even if the effort to implement a model were driven to zero by some magic tool, the modelling of manufacturing systems will never be trivially easy or obvious, and simulation tools will never rival spreadsheets in frequency of use. It also suggests that the need for expert consulting and instruction in the art of simulation will be increased rather than diminished by the growth of tools like XCELL.

The implications for instruction are much clearer. Even in its present form, XCELL is capable of revolutionizing a wide variety of engineering and management courses. Where previously, such courses could only afford the time to describe the use of simulation, it can now be a practical, hands-on, laboratory experience. In several courses at Cornell, we have repeatedly demonstrated the ability to use XCELL in laboratory assignments with students who have had no previous experience in simulation, and other professors at Cornell and elsewhere have confirmed these observations. Using XCELL, one can now make routine, daily assignments that would previously have constituted a major term project.

REFERENCES

XCELL is distributed by Express Software Products, Inc., 115 Warwick Place, Ithaca, NY 14850. 607-257-6614

Conway, R., Maxwell, W., and Worona, S. (1986) User's Guide to the XCELL Factory Modelling System. Scientific Press, Palo Alto, CA. 415-322-5221 (Scientific Press is also the distributor of the educational version of XCELL.)

Henderson, P.; editor (1984). Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments. SIGPLAN Notices, Vol. 19, No. 5.

AUTHORS' BIOGRAPHIES

RICHARD CONWAY is a professor in the Johnson Graduate School of Management, and WILLIAM MAXWELL is the Andrew Schultz Professor in the School of Operations Research and Industrial Engineering, both at Cornell University. Both received their PhD in operations research from Cornell.

Their collaboration in simulation and manufacturing spans a thirty year period. Both were at the RAND Corporation in the early 1960's, working with Dr. Harry Markowitz in the development of the SIMSCRIPT language. Their work in industrial scheduling ("Theory of Scheduling", Addison Wesley 1967) has long been the standard reference for the topic.

Richard Conway
Johnson Graduate School of Management
Cornell University
Ithaca, NY 14853
607-255-7207

William Maxwell
School of Operations Research
and Industrial Engineering
Cornell University
Ithaca, NY 14853
607-255-9134