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

SIMULATION PROTOTYPING

Ingolf Ståhl

Department of Managerial Economics
Stockholm School of Economics
Box 6501
SE-11383 Stockholm, SWEDEN

ABSTRACT

A simulation model is successful if it leads to policy action, i.e., if it is implemented. Studies show that for a model to be implemented, it must have good correspondence with the mental model of the system held by the user of the model. The user must feel confident that the simulation model corresponds to this mental model. An understanding of how the model works is required. Simulation models for implementation must be developed step by step, starting with a simple model, the **simulation prototype**. After this has been explained to the user, a more detailed model can be developed on the basis of feedback from the user. Software for simulation prototyping is discussed, e.g., with regard to the ease with which models and output can be explained and the speed with which small models can be written.

1 THE IMPLEMENTATION ISSUE

Every project, be it a simulation project or some other kind of management science - decision support project, is made with the aim leading to success. The important question is what constitutes success, when it comes to the use of a scientifically based system applied to practical situations. The criteria of success are then usually different from those of academia, e.g., as used in the evaluation of doctoral theses, articles in scientific journals or when deciding on tenure.

The criterion for success when it comes to evaluating a simulation project as applied to a practical problem in business is primarily whether or not the results of the simulation project are really used for supporting decisions. For a simulation model, an additional, but closely linked, criterion is whether the company not only uses the simulation model once, but also rather continues to use it on a more permanent basis, because it really contributed to better decision-making. The traditional academic criteria when it comes to evaluating a computer program, of e.g., originality, elegance, use of the latest available methods, etc., are of less direct importance. In fact, these criteria matter only to the extent they in some way influence the success in the form of actual **implementation** of the simulation model.

Our next question is hence what factors improve the chances of success in the form of implementation. There has been much research on the factors that have affected the implementation of different types of management science models, which in turn should be relevant for the more particular field of simulation models. Much of this literature was written in the 1970's, when implementation of models was an issue of focus (see e.g., Bennet and Felton 1974, Churchman 1970, Doktor *et al.* 1979, Lönnstedt 1971, Mintzberg 1979, Radnor *et al.* 1970, Schultz and Slevin 1975).

On the basis of a review of these books and several others (e.g., Keller *et al.* 1991), coupled with my own experience from trying to implement simulation models in the Swedish construction industry (Ståhl 1981), I have come to regard certain aspects as fundamental with regard to the implementation of simulation models.

We can have a situation as shown by Figure 1 below. We here have a real system, a model builder and a user of the simulation model.

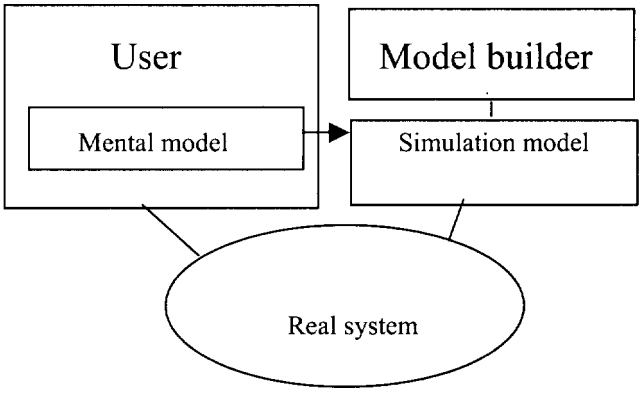

Figure 1: View of Implementation

With the user we mean a person, or a team of several persons, that make some decision on the basis of the simu-

lation model. For the sake of simplicity, we shall below use the word user, even if it is quite common that one on the user side has a whole team of persons. Important aspects of this user role is that the user will be **responsible** for the outcome of possible decisions made on basis of the simulation, that the user can decide not only whether or not the simulation model shall be used, but also how far it shall be developed. The user is the buyer of the simulation model. The user, who can be regarded as the "problem owner", has special "domain" knowledge about the actual situation.

The model builder, again either a single person or a team of persons, produces a simulation model, i.e., a computer based simplified representation of this real system. The role called model builder hence refers to one or several persons who have special knowledge about simulation technique, but less knowledge about the problem situation than the user. The model builder can be an outside consultant or a staff officer, working as an inside consultant.

The implementation of the model must focus on the perception of the user of the model. In order that the user shall want to accept this model, the user must think that this computer model has a good correspondence with the mental model of the system that the user herself has of the system. Every person responsible for decisions regarding a system will in her head have a model or picture of the system, where certain simplifications have been made, implying that some factors in the real system are stressed and others disregarded. Thus, in business situations certain customers or products might be regarded as critical, while others are regarded as unimportant. The computer model is likewise a simplification of the real system, where certain factors are included, while others are disregarded.

Two important issues arise here. First, a correspondence must exist between the simplifications in the user's mental model and the simplifying assumptions of the simulation model. Secondly, the user must be confident that this correspondence really is at hand. Good correspondence between the two models is insufficient for implementation, if the user does not perceive this correspondence, but fears that the simulation model might be at odds with her own mental model.

A prerequisite for action is hence that the user understands the main assumptions of the model and feels comfortable with them. A situation of false confidence, i.e., that a user believes the simulation model to be in line with her own mental model, when this is not at all the case, is an unlikely situation, since most users of a model for important problems will insist on finding the assumptions of the model. The question of user confidence or model credibility is a very important one. When the term user implies a team of persons, different criteria of what constitutes credibility can apply. Various formal methods of credibility assessment are then often suitable (Balci 1998).

For implementation we hence need a simulation model in which the user can have faith in the sense that it is compatible with her own mental model of the studied system. In order to establish faith in the model, the model builder should ensure that the user has a basic understanding of how the model functions and secondly, if the user is not satisfied with the simulation model, due to discrepancy with the user's mental model, adjust the simulation model so that the simulation model comes closer to the mental model of the user.

It is in most cases not efficient with regard to implementation to try to change the user's mental model in case there is a discrepancy. Since, as defined above, the user of the model is a person with a long experience with the system at hand, while the simulation model builder's expertise lies in the field of simulation technique, the model will probably not be used, if the developer fails to adjust it to the mental model of the decision maker. This does not rule out that the model builder can ask the user detailed questions to resolve discrepancies between the developer's mental model, formalized into the simulation model, and the user's mental model. An open discussion and exchange of ideas is generally productive, but the final "vote" must be given to the user, since she is the one who bears the responsibility for the decisions to be made on basis of the model.

2 CHARACTERISTICS OF THE PROTOTYPE

The question is then first how one can ensure that the user understands the simulation model and secondly how one can best adjust the model to correspond to the mental model of the user. It seems that both tasks are best handled by a step-by-step approach, implying that one starts with a very simple model with an easily understood structure. After this simple model, the **simulation prototype** has been presented and explained to the user, a more detailed model can be developed on the basis of the feedback obtained from the user. One can then continue to make the model more complex, if the user finds this suitable.

The advantage of starting simple is not only that user's understanding of the model is improved, but also that one can stop the development process at an early stage, if this is found suitable by the user. The stopping of the development process can be done for two reasons:

1. The user might find that the simulation approach is unsuitable and it is proper to terminate the whole development process before too much money has been wasted. Starting in simulation with a simple prototype is hence quite a safe strategy compared to committing oneself already from the beginning to producing a very complex simulation. It is a strategy that can be recommended especially to corporations with little experience with simulation and where one might soon find out that simulation in some way is incompatible with the normal decision process in the corporation.

2. The user might find that the prototype model is already useful enough and that additional complexity and refinement infers such low marginal utility that it does not compensate for the marginal costs of continuing one step further. An important reason for stopping at an early stage might also be time pressure. A decision has to be made before a certain date. Even if a more complicated model might give more precise answers to the critical questions, it might be better to act on the basis of less precise answers than postponing the decision. A **quick and dirty** approach to decision making (see e.g., Woolsey 1993), using a very simple simulation prototype model, is in many situations the only practical alternative to not doing any simulation at all.

As regards the question of when to stop, one might often stop first after the last step of development has proved to be "a step too far", i.e., the marginal benefit of the last step of development was too low. If one had known the outcome of this last step in advance, one would have stopped even earlier. The smaller the steps in the development process, the less costly the unnecessary last step.

In many cases the user might, however, not want to stop the development of the model, having found that simulation is a very suitable method for investigating the problem. The user might then want to make the model considerably more complex and "realistic", containing a great amount of specific details. We here use the term *prototype* for the early smaller model, which contains the essence of the larger more complex simulation model, and the term *operational* simulation model for the larger more complex model, which serves as a basis for actual decisions, probably during a longer period of time.

The dividing line between these two types of model is not a clear one, since the complexity of the model might increase gradually. Furthermore, the degree of complexity does not constitute the only difference between the prototype and the operational model. Several other features distinguish the simulation prototype models from the operational models; six are listed below.

1. The **efforts** spent on **making the model efficient** from an execution point of view. Since a prototype is not likely to be run over a long time with a great many users, we do not spend efforts on making a prototype model efficient, while we might spend considerable efforts on making an operational model very efficient to execute. This execution efficiency might be important when we run the model many times, to reach statistically significant conclusions and/or to find a strategy that is "optimal" or at least better than many other investigated alternatives.
2. While it is often important to give the operational model, to be used by many unsophisticated users in particular situations, a specific **user interface**, the prototype model will generally rely on a standard type of user interface.
3. The more detailed operational model is in many cases most properly done in a completely **different simulation system** than that of the simple prototype model. This will be discussed in greater detail below.
4. The developer of the prototype and of the operational model might be completely **different persons**. Point 3 above implies different requirements on the model builder, both as regards knowledge of simulation techniques and time availability. It should in this connection be stressed that, since different people might be working with the prototype and with the operational model and since the time between the finish of the prototype and of the full operational model might be extensive, it is of great importance that the prototype model is clearly documented, in particular with regard to the assumptions on which the model rests and the basic logic of the model.
5. While, as discussed above, it is very important that the user understands the prototype model, it is not so important that the user understands the operational model. It might be enough for the user to know that the operational model is a more complex version of the prototype. The prototype can be used as a check of the **validity** of the operational model. If the overall performance of the operational model in some critical aspects produces results deviating by only a few percent from those of the prototype model, then the user's confidence in the prototype can be transferred to the operational model, since it basically relies on the same type of assumptions. It is in this context of importance to note that it will generally be impossible to have the user understand the details of a very complex model, perhaps consisting of thousands of lines of code in a difficult-to-understand computer language.
6. The development of the prototype model might be mainly "**self-service**", in the sense that a significant part of the initial development of the prototype is done by the user herself. The task of the "model builder" might then be more of a teacher, showing how simple models can be built. Such self-service has the advantage of greatly facilitating the understanding of the model by the user and her confidence in this. Self-service is, however, generally unthinkable for a large operational model. A complex model will have to be constructed by a simulation expert.

Summing up, the prototype simulation model will, or should, have the following characteristics:

a. It is written in a short time.
b. It is fairly small.
c. It is easy to explain the model to the user.
d. It is easy to judge the validity of the model.
e. The model is used for short period of time, in a "quick and dirty" way, to solve an immediate problem, or as the base for a more complicated operational model.
f. There is no stress on efficiency of execution.
g. There is no use of a specific type of user interface.
h. The prototype model can be produced by the user in a self-service mode, with the model builder as a teacher.
i. The assumptions and the basic logic of the model are documented to enable different programmers to build a more complex model on the basis of the prototype.

3 SOFTWARE FOR THE PROTOTYPE

We shall against the background of these main characteristics of the simulation prototype discuss what kind of software is most suitable for this type of model. It appears suitable to start with an overview of the main types of simulation systems available. Simulation refers here to "discrete events simulation", i.e., we leave out the area of continuous simulation. We find it in this context suitable to distinguish between the following four main groups of systems:

1. **G**eneral **P**urpose **L**anguages (GPL). A large amount of simulation is still done in a GPL (like C, C++, Pascal, Java and FORTRAN), usually without any major use of standard libraries. (For simulation in a GPL, see Law and Kelton 1991.)
2. **GPL Based S**imulation **L**anguages (GPLBSL). These are languages that in their core have a GPL, possibly slightly modified. Some examples are Simula (based on Algol 60), SIMSCRIPT (based on FORTRAN), MODSIM (based on Modula), CSIM19 (based on C++; see Schwetman 2001) and Silk (based on Java; see Kilgore 2000). Since they have a GPL, often object oriented, in the core, they can be used for writing in principle any kind of program, doing this in a very structured fashion.
3. **B**lock **B**ased **S**imulation **L**anguages (BBSL). These are **general**-purpose **simulation** languages, in the sense that they are applicable to a great variety of simulation problems. The simulation model can be represented not only as code, but also as a block diagram. Examples of such languages are GPSS, SLAM and SIMAN. They constitute today often the core of a more extensive system; e.g., ARENA is built on SIMAN and Awesim on SLAM.
4. **A**nimation **O**riented **S**imulators (AOS), like WITNESS, ProModel and TAYLOR, most often used for the simulation of production system. The model is usually started by drawing a layout of the system to be modeled, fitting closely to the animation that will later be done.

Besides these four kind of systems, simulation can be done also in other types of systems, such as e.g., spreadsheets like Excel, packages based on group 2 type of systems (like DEMOS, based on Simula), simulators based on program generators with a menu (like CAPS and Draft), corporation specific simulators (see e.g., Savén 1995), etc. Many surveys of usage of simulation software, like the investigation made by McHaney (1996), represented in Table 1 below, provide, however, strong evidence that the bulk of simulation is done in the four systems presented below. Hence we shall concentrate on these four systems.

It should here be stressed that animation can be done by many BBSL. The distinction between a BBSL and an AOS does not lie in whether animation is possible or not, but whether the system is transaction or server oriented. The main implication of this difference, to be discussed further below, is that in a BBSL a specific server can be represented in several places, while it in an AOS can be represented in only one place.

Table 1: Software Used in Simulation Projects

1. BBSL: GPSS (12.2%), SIMAN, SLAM	30.9 %
2. AOS: AutoMod, ProModel, WITNESS, etc.	22.0 %
3. GPLBSL: SIMSCIPT, MODSIM, Simula	10.5 %
4. GPL: C/C++, FORTRAN, ADA, BASIC	21.3 %
5. Remaining systems, incl. Excel	15.3 %

4 ARE GPL AND GPLBSL SUITABLE FOR THE SIMULATION PROTOTYPE?

We first argue that in the context of simulating a project in a corporation, the GPL is almost always an inferior alternative, irrespective of whether it is used in the introductory prototyping phase or the more advanced operational phase. The GPL seems to be almost completely dominated by the GPLBSL. The only reasons for using a GPL seems to be that the GPL that the programmer knows does not have a corresponding GPLSBL and/or that one wants to avoid the extra cost of buying the GPLSBL. These factors appear to be of minor importance. The time of learning the syntax of a new language is generally outweighed by the great time reductions in programming gained when having access to routines for queue handling, statistics collection and analysis, etc. The additional costs of a GPLSBL are generally also small.

The reported fairly high frequency of e.g., FORTRAN usage for discrete events simulation must rather reflect great conservatism and inertia or a preference for spending many hours programming in a language one is accustomed to. This choice might be in line with "individual rationality", but is rarely efficient from a corporate point of view.

Next comparing the three remaining types of systems, we argue that, while the GPLBSLs have an important role to play in the operational phase of the simulation project, they are of less interest in the prototyping phase of the project. The reason for this difference is as follows: The main strength of GPLBSLs is that they allow for any kind of programming constructs and for very well structured programs, thanks to many possible hierarchical level (subroutines within subroutines) and object oriented programming (with inheritance and "detail hiding"). This makes them very powerful when it comes to very long programs and to programs, in which many details are modeled at a high level of precision.

These advantages are, however, not so interesting when it comes to prototype models that are short and for which the precision in the modeling is not so important and when the operational model is not likely to be a direct extension of the simulation prototype. If one from the beginning knows that one shall produce the operational model by such a direct extension of the prototype, GPLBLS features, such as inheritance, would be of greater interest than in the case when the operational model is rebuilt from scratch in another system than the prototype.

On the other hand, the GPLBSLs are inferior to the two other types of system (BBSL and AOS), when it comes to the factors that we in the discussion above found to be important for the prototype models. The main factor is the ability of the system to facilitate the demonstration of the basic assumptions of the prototype in a simple manner to the user. GPLBSLs have no standardized way of representing the model structure. Possible graphical representation methods like flow charts seem to be seldom used, since they do not convey information much better than the actual code. A big problem is that the code is usually much longer than the code in e.g., a BBSL, often at least three times longer, due, among other things, to the need for many declarations of different types. The code is often also less self-documenting than that of a BBSL.

For these reasons we rule out also the GPLBSLs when it comes to the prototype phase of simulation. However, when it comes to switching from the prototype phase to a detailed modeling in an operational phase, the GPLBSLs have a very important role to play. One can start with a prototype in a BBSL and, after "selling" the ideas to the user, continue in a GPLBSL, like in the Swedish icebreaking case (Jennergren *et. al* 1996).

5 COMPARISONS BETWEEN BBSL AND AOS

The remaining comparison is between the BBSL and the AOS as regards to their suitability in the prototype phase. It here appears that one must clearly take into account to what type of situation the prototype model refers. There are certain situations in which the AOS are very strong with regard to conveying the "gist" of the simulation. This refers mainly to manufacturing situations, where the animation of all processes can be carried out in proportional time.

This concept of **proportional time** must be more clearly explained. Proportional time implies that the time compression ratio (time-in-reality/time-on-the-screen) is constant during at least a substantial part of the simulation. If, for example, one process takes 5 minutes in reality and 5 seconds on the screen, then a process taking 30 seconds in reality should in the case of proportional time take 0.5 seconds on the screen.

For many animations this is not true, since one must "fake" the times in order to make a "pretty" simulation. Time faking is frequent in animation of service processes. Take, for example, the classic barbershop, where a haircut in reality takes 20 minutes, while it takes 10 seconds to move from the waiting chair to the barber's chair. If a haircut in the animation takes 6 seconds, i.e., we have a time compression ratio of 1200/6=200, then moving between the chairs should take only 10/200=0.05 seconds on the screen. This is too short a time for the human eye to recognize as anything else but a sudden jump. Hence in order to have a "neat" animation, one might allow, for example, 2 seconds for this movement, implying a time compression ratio of 5, i.e., 40 times smaller than that used for the haircut.

In situations where one can make a nice animation of physical processes with the same time compression ratio for all processes, the AOS have a great value in conveying important information about the prototype model. However, in situations where one is forced to choose between the three evils of either 1. running a boring animation for a long time without much happening, 2. distorting reality by having different time compression ratios or 3. having the animation very jumpy, animation systems are not helpful in conveying the main idea of the model to the user. In these situations one might often do just as well without animation.

Animation might furthermore not be very informative, when the simulation mainly deals with "invisible processes" such as message flows, cash flows and other accounting features, e.g., costs and profits. One must then choose symbols that often do not lead to much better model understanding than one could obtain by presenting graphs and text. This is often just as easily done in a BBSL.

Even if the animation-based system can provide significant benefits, like for manufacturing system with constant time compression, one might in some cases still prefer the BBSL. One reason for this is that the model logic of the AOS would be much more complicated than that of the BBSL.

As mentioned briefly above, in an AOS each permanent server is in principle only represented **once**, since it in the animation workspace, representing e.g., the factory floor, must be in only **one** place. In a BBSL a permanent server can be represented in **many** different places, since we here follow the temporary entities and, if different entities use the same server, this usage of the server can take place in different parts of the program.

This difference is important when it comes to establishing what kind of models a user will be able to write on their own after a short amount of learning time. When using an AOS, only certain simple systems are very easy to model, namely when each server only serves one type of temporary entity and only does so once. Hence, a system where each product has its own machines, each visited once, is easy to model. You just place the machines in the work area and draw the paths from the entry source through the machines to the exit. For each machine you just input the processing times. This is exemplified in Figure 2.

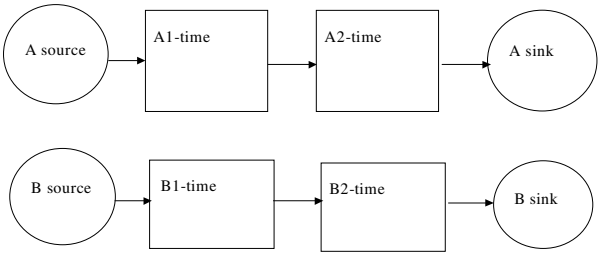

Figure 2: Each Product has its Own Machines

If one machine, however, processes more than one product, modeling becomes considerably more complicated. You must then have rules for determining which processing time applies to which product, as exemplified in Figure 3, and which path, leading from the machine, each product should take, as exemplified in Figure 4.

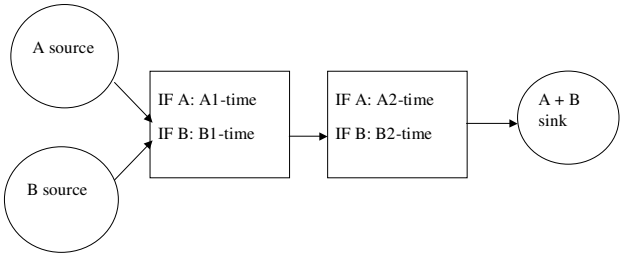

Figure 3: Two Products Use the Same Machines

While the user has to write code that is complicated for the beginner, inside the dialog of the servers in the AOS, the logic is just as simple is the two second examples as in the first one in the BBSL. The simple logic structure of figure 2 applies in all three cases for the BBSL, since e.g., the same machine can in one place have one time and in another place another time.

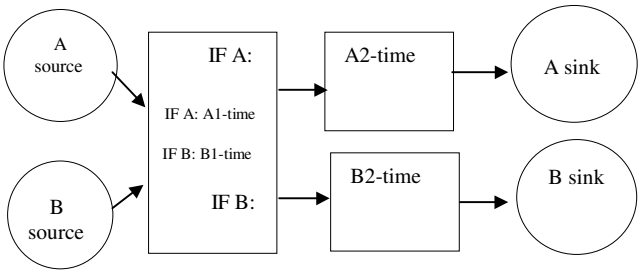

Figure 4: The Products Also Choose Paths

The logic in the AOS becomes even more complicated if a transaction comes to the same server several times. This is illustrated by "the Boris vodka shop problem", where we instead of machines have humans and instead of products have customers: "At a store, run by Boris and Naina, customers arrive at rate of 7 $\pm$ 3 minutes. A customer first goes to Boris and chooses his bottle. This takes between 3 and 7 minutes. Next he goes to Naina to pay for the bottle. This also takes 3 to 7 minutes. Finally, he returns to Boris to pick up his bottle. This takes between 1 and 3 minutes. He then leaves the store. There is one waiting line in front of Boris and one in front of Naina. A customer returning to Boris to pick up his bottle has to start at the end of this line again. The store is closed after eight hours".

This example has been used in two "pseudo-experiments" for comparisons between AOS and BBSL. The first experiment was carried out in September 1996 with a class of Latvian students with no prior experience of simulation at the Riga Technical University. Half of them had four hours of an AOS (WITNESS), the other half four hours of a BBSL (GPSS). At the end of each of these sessions, the students were asked to model the Boris problem. While none of the AOS students could do this, all the BBSL students could do so. The second "pseudo-experiment" was carried out with vendors at the Winter Simulation Conferences in the late 90s. The vendors of different systems were asked to solve the Boris problem using their own system. While the BBSL vendors could solve this problem in less than five minutes, all of the AOS vendors required more than 30 minutes. Figure 5 shows the complicated logic in an AOS.

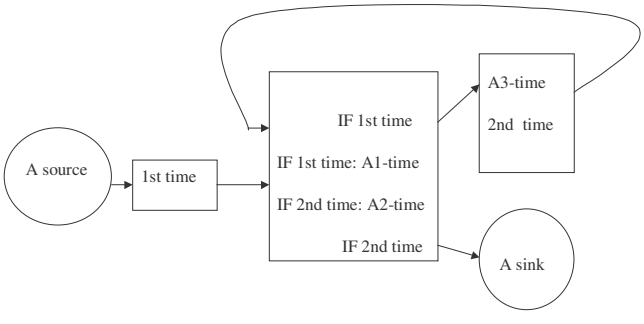

Figure 5: The Boris Model in an AOS

The logic in the BBSL is much simpler. This can be exemplified by Figure 6, showing the block diagram of the whole program in one GPSS dialect (micro-GPSS, Stahl, 1990).

The reason for the difference in complexity is that an AOS requires Boris to be located at **one** spot and the program must then for each customer keep track of whether he comes to Boris for the first or the second time. In the BBSL, Boris is "seized" in two different places.

Another factor speaking for using a BBSL rather than an AOS for the prototype is the possibility of good **documentation.** This is very important for the user of the prototype when trying to understand the logic of the prototype program. As regards documentation, some BBSLs have the advantage of providing both a compact and readable text version of the program as well as an easy-to-read, but detailed, block diagram presenting the logic of the model (see e.g., Figure 6). One can start by looking at the main structure of the block diagram, before looking at the details of the program syntax. In some AOS there is no unified, easy-to-read, listing of all the code involved, but the understandable code parts are scattered over a set of input dialogs. In other types of AOS, the documentation provided automatically is not clearly coupled to the way in which the model was originally constructed in dialogs and hence more difficult to understand.

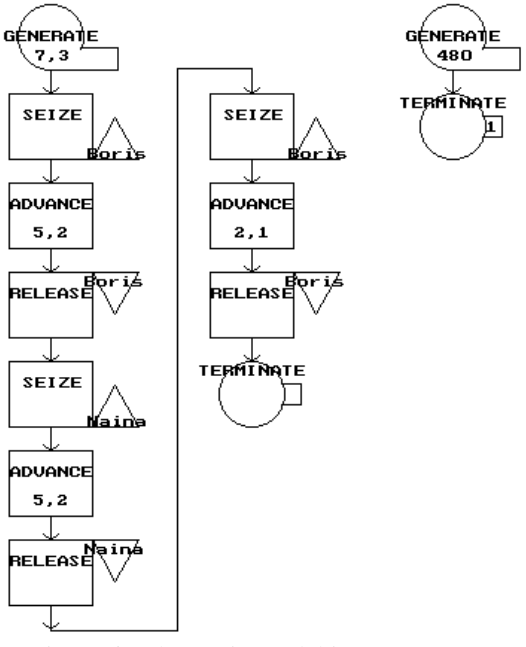

Figure 6: The Boris Model in one BBSL

It should, however, be noted that for many manufacturing situations with proportional time representation, these two disadvantages of AOS are small compared to the great benefits that the visual representation has in conveying the main modeling ideas of the simulation to the user.

The conclusion is hence that the relative merits of AOS and BBSL are really quite dependent on the actual system to be simulated. There is clearly no type of system that is better for all types of prototype models. The choice of software to be used for the prototype, an AOS or a BBSL, must be determined on the basis of what type of situation is to be modeled.

REFERENCES

Balci, O. (1998) Verification, Validation, and Testing. In J. Banks (ed.) *Handbook of Simulation*. Wiley, N.Y.

Bennett, J. and E.L. Felton, Jr. (1974) *Managerial Decision Making: Case Problems in Formulation and Implementation.* Grid Inc., Columbus.

Churchman, C.W. (1970) Managerial Acceptance of Scientific Recommendations. In A. Rappaport (ed.) *Information for Decision Making*. Prentice Hall, Englewood Cliffs.

Doktor, R., R. L. Schultz and D. P. Slevin (eds.) (1979) The Implementation of Management Science. *Studies in the Management Sciences*, Vol. 13. North-Holland, Amsterdam.

Jennergren, L.P. , L. Lundh, U. Törnqvist and S. Wandel. (1995) Icebreaking Operations in the Northern Baltic. In H. J. Miser (ed.) *Handbook of Systems Analysis: Cases*. Wiley, Chichester, U.K.

Keller, L., C. Harell and J. Leavy (1991) The Three Reasons Why Simulation Fails. *Industrial Engineering,* April.

Kilgore, R. (2000) Silk, Java and Object-Oriented Simulation. In J. Jones, R. Barton, K. Kang and P. Fishwick (eds.) *Proceedings of the 2000 Winter Simulation Conference,* 246-252. SCS, Orlando.

Law, A. M. and D. Kelton (1991) *Simulation Modeling and Analysis.* McGraw-Hill, NY.

Lönnstedt, L. (1971) *OR in corporations listed on the stock exchange - A problem of innovation and adjustment for the individual and the organization.* (In Swedish) Bonniers, Stockholm.

McHaney, R. 1996. *Simulation project success and failure: Some survey findings.* Working paper, Deparment of Management, Kansas State University, Manhattan.

Mintzberg, H. (1979) Beyond Implementation - An Analysis of the Resistance to Policy Analysis. In Haley (ed) *Operations Research '78*. North-Holland., Amsterdam.

Radnor, M., A. Rubinstein and D. Transik (1970) Implementation in Operations Research and R&D in Government and Business Organizations. *Operations Research.* Vol 18, 967-991.

Schwetman, H. (2001) CSIM19: A Powerful Tool for Building System Models. In B. Peters, J. Smith, D. Medeiros and M. Rohrer (eds.) *Proceedings of the 2001 Winter Simulation Conference*, 250-255. SCS, Arlington, VA.

Schultz, R.L. and D.P. Slevin.(eds.) (1975) *Implementing Operations Research/Management Science*. American Elsevier, New York.

Savén, B. *Business modeling for decision support and learning: A study of discrete production simulation at ASEA/ABB 1968-1993.* (In Swedish) Linköping Studies in Science and Technology. Dissertation No. 371.

Ståhl, I. (1981). *Budget simulation within the construction industry* EFI Working Paper, Stockholm.

Ståhl, I. (1990) *Introduction to Simulation with GPSS - on the PC, Macintosh and VAX.* Prentice Hall, Hemel Hempstead, UK.

Woolsey, G. (1993) Where we were, where we are, where we are going, and who cares? *Interfaces TIMS*. Vol. 23, 5: 40-46.

AUTHOR BIOGRAPHY

INGOLF STÅHL is a Professor at the Stockholm School of Economics, Stockholm, and has a chair in Computer Based Applications of Economic Theory. He has taught GPSS for 25 years at universities and colleges in Sweden and the USA. Based on this experience, he has led the development of the micro-GPSS and WebGPSS systems. His email address is <ingolf.stahl@hhs.se>. His simulation web-address is <www.webgpss.com>.