BEYOND DYNAMO : FUTURE SYSTEM DYNAMICS SIMULATION PROGRAMMING

Raj Kumar Bapna
International Software India Limited
SDF Block 1, Units 5 & 6, M.E.P.Z.
Madras-600 045 India

Sujoy Ghose
Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur-721 302 India

Sushil K. Sharma
Department of Mechanical Engineering
Institute of Technology, B.H.U.
Varanasi-221 005 India

ABSTRACT

Advances in computer technology, since DYNAMO first appeared, promise significant developments for System Dynamics modeling and simulation. This paper discusses the future System Dynamics simulation languages as a part of human-engineered, integrated simulation programming environment. Many features of such languages and environments, that may become available in the next five years, are identified and discussed. The relevant advances in Simulation and Computer Science are presented with appropriate modifications in the context of System Dynamics methodology. Some aspects of implementation of such languages and environments are also discussed.

1 INTRODUCTION

DYNAMO was designed over 25 years ago for simulation of the System Dynamics (SD) models and it continues to be the most widely used simulation language for SD studies (Forrester 1961, Richardson and Pugh 1981). It is a compliment to the creators of DYNAMO that it has remained largely unchanged over the years, and that other languages like, DYSMAP (Coyle 1977), DYMOSIM (Mohapatra and Bora 1983), NDTRAN (Uhran and Davisson 1984) are similar to DYNAMO.

DYNAMO was initially developed for batch processing and program entry was done with computer cards. It was later adapted for interactive use with computer terminals. However, it did not exploit the advantages of the interactive mode (and graphical/visual communication). Computer technology has greatly developed since DYNAMO first appeared. Hardware has become cheaper and more powerful and software has become more versatile and convenient to use. These developments promise new possibilities of rapid developments for SD modeling and

simulation. The future SD simulation will be fundamentally similar to what it is with DYNAMO. However, significant developments in SD modeling and simulation in the near future will emerge primarily in the following two ways:

(1) Application of currently well established computing techniques to provide a number of tools in an integrated SD simulation programming environment. Typical in this category are dimensional analysis facility and spreadsheet-like program execution monitoring facility

(2) Development of fundamental concepts and techniques in the fields of SD modeling and simulation. Typical in this category is a facility to qualitatively/quantitatively analyze the SD model performance.

Such developments are likely to happen at a faster pace for SDSLs(System Dynamics Simulation Languages) than for other simulation languages (because implementing SDSLs is easier due to their simple syntax), and make SD and SDSLs more popular and wide spread in use. Other developments will come from applications of other fields, e.g., Artificial Intelligence.

Some work has been done on advanced and futuristic concepts in SD, simulation and software (Aus and Korn 1971, Baltzer 1983, Coyle and Sharp 1976, Ghose, Chakrabarti and Bapna 1987a, Hooper 1984, Oren and Ziegler 1979, Randers 1980). This paper primarily addresses those methods, techniques, features, or facilities which are likely to become available during the next five years in the SDSLs.

1.1 Brief History of System Dynamics and DYNAMO

Prof. Forrester pioneered the philosophy, principles, techniques and applications of System Dynamics (Forrester 1961, 1969, 1973). DYNAMO was developed at MIT over 25 years ago as the first simulation language for SD studies. Though DYNAMO is still the standard language for SD studies, there are other languages like DYSMAP (Coyle 1977) and DYMOSIM (Mohapatra and Bora 1983) which also were specifically developed for SD studies. In addition, general purpose continuous system simulation languages are also being extensively used for the same purpose (Gordon 1982).

Various types of studies on SD methodology have been conducted. Such studies include eigenvalue analysis (Forrester 1983), critical parameter identification (Starr 1981), sensitivity analysis (Vermeulen and De Jongh 1977), parameter estimation (Thillainathan and Price 1981), objective function specification (Rohrbaugh and Andersen 1983) and parameter optimization (Birta 1977). There have been a large number of successful applications of the SD approach. SD is also considered as a useful philosophy for modeling and simulation of any system in general (Roberts 1983). Use of both SD methodology and DYNAMO has spread to many

areas over the years - SD and DYNAMO have even been found useful as tools for teaching simulation to children (Roberts 1983).

1.2 Relevant Trends in Simulation and Computer Science

Computer Science community seems to be directing its efforts for designing computer systems keeping the user in mind (Raeder 1985). The main reason for this is the wide use of powerful yet inexpensive micro computers. Better user interfaces are being designed by using graphics and using results of Artificial Intelligence research. Typical examples of the current trend are Macintosh (Benzon 1985) computer, UNIX (Kochan and Wood 1984) operating system, Ada (Buxton 1980) and Small Talk (Goldberg and Robson 1983) programming languages. Also there is a trend to provide a set of tools in an integrated environment, e.g., UNIX.

Unlike the traditional simulation languages, like, GPSS, SIMULA, SIMSCRIPT etc, the current trend in simulation languages attempts to graphically draw a model on the screen, and then study its behavior through animation and dynamically evolving statistics. IDSS (IDSS 1983) and SIMAN (Pegden 1982) allow a user to store a simulation trace and use it for post simulation run animation. CINEMA (an extension of SIMAN) allows the user to draw the simulation scene in color and icons. There has been a trend to include popular features from other simulation languages, e.g., both discrete and continuous system simulation features are provided in modern simulation languages (Cellier 1979, Hooper 1984). Another area of current interest in simulation is the application of model bases and knowledge bases.

1.3 Overview of Future System Dynamics Simulation

Many of the fundamental ideas and techniques for the future SDSLs will come from DYNAMO itself. Such features include the DYNAMO style of writing equations, automatic sorting of equations, pre defined functions, reporting of graphical and tabular outputs etc. However, there will be development of integrated SD simulation programming environments. In addition to providing SDSLs, such environments will also provide supporting software tools. The primary design consideration will be ease of use as well as power and flexibility.

Many features can be provided by such SD environments in future. User-friendly interfaces can ease the process of programming. Syntax-directed editors can help enter programs correctly. Graphical languages can allow writing programs as diagrams. Run-time editing, debugging can be important features. Assertion-directed break points and spreadsheet-like facility can provide easy debugging and monitoring of program execution. Modular programming can facilitate modeling and simulation of large and complex systems by teams. Dimensional analysis, unit conversion and model validation can enforce reliable modeling and programming. Program generators can permit flexibility and

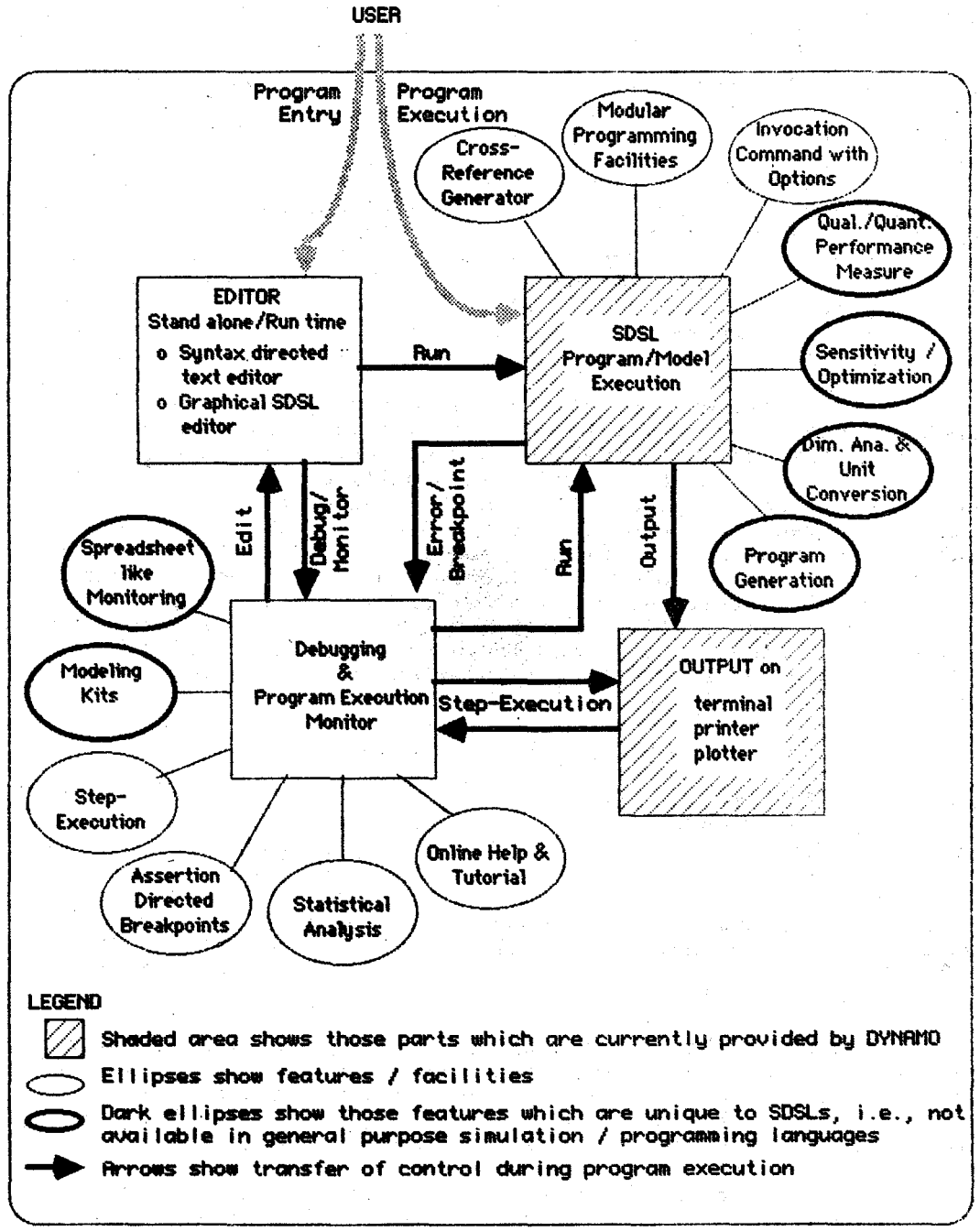

Fig. 1 SCHEMATIC OF FUTURE SYSTEM DYNAMICS SIMULATION ENVIRONMENT

portability. The user can analyze performance of a model qualitatively as well as quantitatively.

Figure 1 shows the schematic of such an environment. The features and facilities provided by DYNAMO are shown shaded in the figure. Ellipses represent features or facilities. Dark ellipses represent those additional features which are unique to the SDSLs and not available in general purpose simulation or programming languages. The user initiates interaction with the environment via editor or via program execution. And, as the program executes, the control passes to various parts through the paths of the environment indicated by the arrows.

2 HUMAN-ENGINEERED INTERACTIVE SIMULATION ENVIRONMENT

Human-engineering attempts to develop effective user-interfaces (Be 1982). The design of such interfaces considers various factors such as psychology of programming (Weinberg 1971), operator skill, user reinforcement, feedback, consistency, demands on human memory etc. Errors are dealt with in a comprehensive and user-friendly way. Human-engineered designs make extensive use of interactive style of user-computer communication, computer graphics, visual programming.

Future SDSLs will be human-engineered throughout the entire life cycle of the SD simulation project/programming - model design, program development, program entry, program execution, debugging, testing and report generation. Ease of use is likely to be the primary design criteria for the future SD simulation systems. The language can be so designed that a first time user can learn and use it easily, but at the same time, an experienced user can have flexibility and power of programming.

A future SDSL will not be just a language, but it will rather be an environment. Because simulation (including SD simulation) is an experimental technique, a human-engineered interactive simulation environment is of great importance in making simulation easier, more powerful and in increasing user productivity.

2.1 Interactive User-friendly Interfaces

Interactive user-friendly interfaces for SD can use several techniques which are already popular in the computer industry. Some of them are discussed below, and others are discussed elsewhere in this paper.

Syntax: The syntax of commands as well as the SDSLs can be uniform and concise. Most commands can also have abbreviated forms.

Variety and choice: A rich set of features may be provided so that most of the operations can be done in more than one way.

This extra choice enables a user to work in a way which is natural to him.

Invocation command with options: An invocation command, that accepts options, can allow a user to select options without making changes to the program.

On-line help and tutorial: On-line help allows a user to get help-information on the screen while he is using the computer system interactively. On-line tutorial with practice sessions makes it easy to learn and use the system.

Report generation: This allows the user to print the SDSL source program as well as the tabular and graphical outputs in a style he like.

Defaults: To provide power and flexibility of use, a rich set of features are needed. However, most of the time, the user needs to know little about language constructs, commands and arguments. When unspecified by the user, a suitable option is assumed as default. The defaults are set to normally used values.

Mouse and menus: A menu is a list of various options displayed on the screen from which a choice can be made. The selection is made by typing a code or using a special selection device called mouse. A primary benefit of menus is that they do not require the user to memorize the commands and their syntax. Further, menu based systems encourage system exploration - reminding the user of various options available.

Windows: Graphics-based displays allow the screen to be divided into many rectangular areas called windows. Each window is independent of other windows. All windows on the screen can contain different information simultaneously. Windows can be created, deleted, overlapped, moved and changed in sizes.

Integrated environment: Most computing environments (including the future SD simulation environment) consist of several software tools, each designed for a sub task within the overall system. An integrated environment presents a uniform and systematic view of all the tools to the user. This makes rapid changes of modes easy and encourages use of parts of the system that would otherwise remain unused.

2.2 Run-Time Editing, Debugging and Monitoring

The future SDSLs will execute programs under the user's control and provide facilities like run time editing, debugging and monitoring. Tracing and assertion-directed break points are important for debugging. Spreadsheet-like monitoring facility would be very useful in the context of SD

Run-time error handling and correction: As soon as an error is detected during program execution, the program will be suspended.

Then the user is shown appropriate statements with suitable error messages and suggestions for error correction. The user may correct the errors, and then may continue the program execution, stop the program execution, make more changes, or restart the program.

Tracing: As the program executes, the output, as soon as it is calculated, may be produced in a tabular or graphical form. This way the user can trace the execution.

Assertion-directed break points: This is a facility to halt a program when some assertion or condition becomes true during program execution. The control is transferred to the user terminal. The user can set new break points, change values of variables, edit the program, resume execution, or view values of various variables using a spread-sheet like facility.

Spreadsheet-like monitoring: The concept of spreadsheet was popularized by a package called Lotus 1-2-3 (Kelley 1983). A spreadsheet allows viewing two dimensional arrays on the screen as cells arranged in rows and columns. The spreadsheet can contain a large number of rows and columns and a portion of that is shown on the screen at a time. In SD context, each cell can keep a numeric value. Each column of the spreadsheet can keep values of one particular variable corresponding to different times. And each row can keep values of different variables corresponding to some particular time. Spreadsheet can offer another important facility, i.e., the values of one or more variables may be changed and simulation may be automatically repeated for the necessary iterations to update all the cell values. This facilitates answering "what-if" questions. It is possible to insert or remove rows or columns; to instantly create a graph for certain variables and view the graph; and to perform statistical calculations such as max, min, mean etc.

2.3 Syntax-Directed Editing

Text-editors are used for entering and modifying data and program. Syntax-directed editors are text-editors but they also have knowledge of the particular language syntax, i.e., the rules defining the correct sequences of language elements. Thus they allow only syntactically correct statements, equations or language elements to be entered.

Syntax-directed editing has several advantages:

(1) It reduces typing effort and minimizes typing errors. This reduces program development time.

(2) It ensures that a program is syntactically correct.

(3) It provides an interface for debugging tools.

Syntax-directed editors for SDSLs can be designed to be particularly more effective (than those for general simulation or programming languages) because of the simple syntax of SDSLs.

2.4 Graphical Programming and Visual Programming

All the currently used SDSLs require the user to convert the model specifications from some kind of a causal loop diagram to a set of equations. The future SDSLs may allow programs to be entered graphically. Suitably modified causal loop diagrams may be used for this purpose. Such a graphical programming language offers a more natural way of SD modeling and simulation. This should result in reduced program development time, increased reliability and increased understanding of the program by the user.

Visual programming (Melamed and Morris 1985) attempts to use visual (and graphical) means to show on the screen a program and its execution. This results in increased understanding of the program and its behavior.

3 RELIABLE MODELING AND PROGRAMMING

The designs of programming languages have evolved over the past three decades. The design changes reflect our changing understanding of good methods of writing large and complex programs that are reliable and maintainable (Pratt 1984). And it also reflects the changing programming and execution environment.

3.1 Modular Programming

Features like subroutines, procedures and macros in programming languages facilitate modular programming. Modular programming (Pressman 1982) means that a large software problem is divided into a set of smaller software units or elements called modules. Modularization utilizes the principle of divide and conquer. Modularization also simplifies program debugging and testing.

In SD context, modular programming offers several benefits. Modules developed in current SDSLs require all variable names to be different and unique. Modular programming will require that variable names within each module need be unique, but the same variable name may appear as local-variable in more than one module (Some global or inter-module variables need to be known to more than one module). This simplifies model development and makes it possible to use some previously developed modules without major alterations. Modular programming will facilitate development of "algorithmic control modules" (Wolstenholme 1985a) and make it possible to easily incorporate them in the SD programs. The algorithmic control modules are standard submodules or submodels which fit into many SD models.

Modular programming facilitates top-down design of software (Mills 1971), i.e., the software system is initially designed from a global or a total view, and then details for each module are worked out at a later time during implementation. Modularization also facilitates development of large programs by a team (Pressman 1982).

Further, with the availability of modular programming features, execution-time profiler may also be provided. An execution-time profiler is used to determine how much time is taken for execution by each module. If some module is found to take too much time, the model of that module may be simplified. It is obvious that such changes will make the program run faster, but may also reduce the accuracy of the model.

3.2 Dimensional Analysis and Unit Conversion

Dimensional analysis, a facility currently provided by DYSMAP (Coyle 1977), is used to find out if all the equations in an SD model are dimensionally correct. For this purpose, the user has to specify the dimensions of each variable. A natural extension of dimensional analysis concept is to allow the users to specify different units for the same type of quantity and also specify the conversion formulae. This facility called unit conversion was implemented by us in a DYNAMO-like language (Bapna 1985, Bapna et.al 1987a).

3.3 Model Validation and Error Diagnosis

A syntax-directed editor (See Section. 2.3) will identify and help correct most syntax errors. The future SDSLs will perform extensive error diagnosis and generate user-friendly error messages. Run-time error correction (See Section 2.2) would also form an important part of error diagnosis and correction.

Model-validation will be performed to a limited extent since comprehensive model validation is not yet possible in SD or any other method of simulation. Expert systems are expected to solve some of the problems in this area.

3.4 Software Maintenance Tools

Computer programs are always changing. There are bugs to fix, enhancements and optimizations to make, versions to change etc. Certain tools help maintenance of software.

Cross-reference generator: In the context of SD, cross-reference generator can provide cross-reference listings showing the line numbers in which a variable gets a value assigned and all the line numbers where it appears in an equation on the right hand side. Such lists can be separately generated for each type of equation, e.g., A-type, L-type, R-type etc. In case of modular programming, cross-reference listing for inter module variables may also be generated.

Source code complexity measure: Software matrices (Gibb 1977) can be used to find the complexity of a program in terms of its modularity, readability and maintainability. A suitable source code complexity measure may be provided by the future SDSLs.

Source code version control systems: Source code version control systems can be used to keep track of history of each module and this facilitates development and maintenance of large programs by teams. This facility is available in good operating systems like UNIX.

4 ANALYTICAL TECHNIQUES AND TOOLS

Availability of analytical techniques and tools for studying SD models will be another important aspect of SDSLs in future. Such techniques and tools can help the user to understand the SD modeling and simulation better, guide the user to design better models, and give the user more flexibility and control.

4.1 Program Generators

Some languages are implemented as program generators. For example, a CSMP program is first translated into FORTRAN program which is then compiled.

A program generator for SDSL will accept as input a program in SDSL and produce as output an equivalent program in Fortran, Pascal, Basic, C or some other high level programming language.

Availability of a program generator will offer many advantages:

(1) Rather than writing a complete program in a high level language for a special application, the user can write the initial program in SDSL. Then the user can get an equivalent program generated in a high level language and make necessary changes to allow flexibility which SDSL may not allow.

(2) The generated program in high level language is correct because many logical checks are made by the SDSL processor.

(3) The generated program in high level language can be compiled with the best available optimizing compiler for high run time efficiency.

(4) Using the generated program, the SD model can be simulated on a computer even if SDSL is not available on that computer.

4.2 Qualitative/Quantitative Performance Analysis

At present, using DYNAMO or other SDSL, the graphical outputs (or tabular outputs) have to be studied to know how good the system performance of an SD model is. The future SDSLs will allow a user to determine the performance of an SD model qualitatively

(Ghose et al 1987b, Wolstenholme 1985b) as well as quantitatively (Bapna and Sharma 1987b). We strongly feel that availability of such general purpose QPM (Qualitative/Quantitative Performance Measure) will be a major development in the SD methodology.

4.3 Sensitivity Analysis

Availability of QPM will make it possible to provide practical sensitivity analysis tools. Sensitivity analysis can be used to study how the system performance of an SD model is affected by variations in certain variables or policies in the model.

4.4 Optimization

With the availability of QPM, it will be possible to provide features to perform optimization. Three types of optimization will be particularly feasible for SD (Bapna and Sharma 1987b).

(1) The user may specify the range of variables. Then non-linear programming (Operations Research) can be used by the package to find a sub-optimal solution with associated values of the variables.

(2) The user may specify various alternate policies for an SD model. The SDSL can find QPM for each of the options and choose the best solution.

(3) A model base may contain several models for the same system. These models may differ by one or more modules. The SDSL can find QPM for each model and choose the best solution.

Qualitative optimization (Ghose et al. 1987b, Wolstenholme 1985b) is another possibility. In all optimization, some of the best solutions can be given to the user, so the an expert user can then make the final choice.

4.5 Applications from Other Fields

Some new ideas and techniques in SD methodology may come from other fields, like Simulation, Control Systems, Optimization, Decision Support Systems, Model-base technologies, Artificial Intelligence, Expert Systems etc. (Ghose, Chakrabarti and Bapna 1987a, Mohapatra and Sharma 1985, Nilsson 1972, Weiss and Kulikowski 1984). The availability of a suitable qualitative/quantitative performance analysis methods will help develop applications from these fields. Such applications will lead to development of new SD techniques and tools.

4.6 System Dynamics Modeling Kits

The future SDSLs may also provide SD modeling kits. These kits may contain several well written programs for common use for SD applications. Such kits may serve two purposes. First, it helps a student or new user to quickly acquaint himself with SD modeling

and simulation programming. Second, it serves as a reference for the experienced user when he needs to build SD models. A drawback of SD modeling kits is that the users may tend to use a model or program from the kit even without ensuring its suitability for a specific application.

4.7 Statistical Summary Facility

Summary statistics, such as extreme values, mean, standard deviation, min, max etc. may also be computed; these may be provided in a easy to use fashion with a spreadsheet type of facility (See Section 2.2).

5 IMPLEMENTATION OF FUTURE SDSLs

Programming language design and implementation methods have evolved rapidly since the earliest languages (including DYNAMO) appeared in the late 1950s (Aho and Ullman 1977). The implementation of SDSLs is particularly easy because of its inherent simplicity. Absence of control structures is the primary cause of simplicity.

The implementations of SDSLs in future will have to consider various factors like implementation techniques, portability considerations, efficiency considerations and the available hardware systems. An attempt to provide a user-friendly interactive environment will also be an important design consideration.

5.1 Portability Considerations

Portable implementation means that the implementation can be moved to different machine or computing environment with much less effort than that required to rewrite it (Wallis 1982). The main advantage of portability is economy - if a software product is portable, it is cheaper and simpler to implement on other computers. Portability may have the effect of making the software better documented, better designed and more thoroughly tested than it might otherwise have been - these aspects make it more reliable and easily maintainable. MS-DOS and UNIX operating systems as well as the C programming language are likely to be a popular choice for achieving portability.

5.2 Efficiency Considerations and Implementation Methods

There are two techniques for translation of programming languages - compilation and interpretation. Compilation is usually used for achieving fast program execution and interpretation is used for providing good interactive program development and debugging facilities. Many of the current languages (including DYNAMO) have been designed to maximize the execution-time efficiency. However, execution-time efficiency is only one side of the problem. The other side is that a program also has to be created, modified,

debugged, tested and maintained by the users. The precious user time and effort can be saved by providing good interactive facilities (See Section 2).

The future SDSLs will be implemented to contain both compiled and interpreted modules to take advantages of both the approaches. Some implementations may use microprogramming (Hayes 1978) for achieving higher run-time efficiency.

5.3 Computer Hardware and Technology

The computer hardware has become relatively cheap and powerful. Inexpensive personal computers may be used in various simple SD applications. Bigger computers are of course needed for complex applications. SDSLs will need to be implemented on various types of computers.

Future SDSLs may also exploit the advanced technology like parallel processing, distributed processing and computer networking. The very nature of SD simulation methodology permits exploitation of parallel computers and array processors. Such hardware can result in significantly increased computing power.

5.4 Estimate of Time Needed for Implementation

A rough estimate of the time needed for experimental implementation of each feature described in this paper is 1 to 10 man months. The effort required may be less if many features are to be implemented by the same team.

6 AN EXPERIMENTAL IMPLEMENTATION

We implemented a DYNAMO-like language (Bapna 1985, Bapna et.al 1987a). The implementation included usual DYNAMO-like features like sorting of equations, pre defined functions, detecting level-less loops and generating associated cross-reference listing etc. and some additional features like dimensional analysis, unit conversion, local sensitivity analysis (over DT period).

7 CONCLUSIONS

This paper argued that the advances in the fields of Simulation and Computer Science offer opportunity for improvements in the way SD is created, the way user interacts with the program and in general the way SD studies are carried out. These advances were presented in a suitably modified way in the context of System Dynamics. Various features discussed include human-engineered interactive simulation environment, reliable modeling and programming and analytical techniques and tools. Many of the features discussed are likely to become available in next five years. Some aspects of implementation were also discussed.

REFERENCES

Aho, A. V., and Ullman, J. D. (1977) Principles of Compiler Design, Addison-Wesley.

Aus, H. M., and Korn, G. A. (1971) "The Future of On-Line Continuous System Simulation", Proc. AFIPS/JSCC.

Balzer, R. R. et al.(1983) "Software Technology in 1990s : Using a New Paradigm", Computer November.

Bapna, R. K. (1985) Design and Implementation of a Simulation Language for System Dynamics, Master's Thesis, Indian Institute of Technology, Kharagpur, India.

Bapna, R. K., Ghose, S., Sharma, S. K. (1987a) "Development of a System Dynamics Simulation Language", Proc. of the Second National Conference on System Dynamics, India, Jan 1987.

Bapna, R. K., and Sharma, S. K. (1987b) "A Quantitative Performance Measure and its Applications to System Dynamics", Proc. of the 1987 International Conference of the System Dynamics Society, China, June 1987.

Be, K. (1982) "Human-Computer Interaction", Computer, November.

Benzon, B. (1985) "The Visual Mind and the Macintosh", Byte, January.

Birta, L. G. (1977) "A Parameter Optimization Module for CSSL-Based Simulation Software", Simulation, Vol 28 No 4.

Brooks, F. P. (1975) The Mythical Man-Month: Essays in Software Engineering, Addison-Wesley.

Buxton, J. (1980) Stoneman: Requirement for Ada Programming Environments, US Dept of Defense.

Cellier, F. E. (1979) Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools, Ph.D. Dissertation, ETH, Zurich.

Coyle, R. G., and Sharp, J. A. (1976) System Dynamics - Problems, Cases and Research, John Wiley and Sons.

Coyle, R. G. (1977) Management System Dynamics, John Wiley and Sons.

Davis, R., and Lenat, D. B. (1982) Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill.

Forrester, J. W. (1961) Industrial Dynamics, MIT Press.

Forrester, J. W. (1969) Urban Dynamics, MIT Press.

Forrester, J. W. (1973) World Dynamics, Wright-Allen.

Forrester, N. B. (1983) "Eigenvalue Analysis of Dominant Feedback Loops", The 1983 International System Dynamics Conference Plenary Papers.

Ghose, S., Chakrabarti P. P., Bapna, R. K. (1987a) "Intelligent Decision Support Systems Using System Dynamics Models", Proc. of the Second National Conference on System Dynamics, India, Jan 1987.

Ghose, S. et. al (1987b) "Qualitative Optimization-An AI Approach to Policy Design in System Dynamics", Proc. of the Second National Conference on System Dynamics, India, Jan 1987.

Gibb, T. (1977) Software Matrices, Cambridge, Mass.

Goldberg, A and Robson, D. (1983) Smalltalk 80, The Language and Its Implementation, Addison-Wesley.

Gordon, G. (1982) System Simulation, Prentice-Hall.

Hayes, J. P. (1978) Computer Architecture and Organization, McGraw-Hill.

Hooper, J. W. (1984) "Simulation Technology : Advances, Trends, Challenges", Simulation, January.

IDSS (1983) Prototype(2.0), IDSS User's Reference Manual, Pritsker and Associates, Inc.

Kelley, J. E. (1983) The IBM PC and 1-2-3, Banbury Books.

Keloharju, R. (1987) "Use of Optimization Techniques in SD", Orientation Course Lecture Notes of the Second National Conference on System Dynamics, India, Jan 1987.

Kochan, S. G., and Wood, P. H. (1984) , Exploring the UNIX System, Hayden Book Company.

Lucas, J. J., and Wait, J. V. (1975) "DAREP - A Portable CSSL-type Simulation Language", Simulation, Jan 1975.

Melamed, B., and Morris, R. J. T. (1985) "Visual Simulation: The Performance Analysis Workstation", Computer, August.

Mills, H. D. (1971) "Top_Down Programming in Large Systems", In R. Rustin(ed.), Debugging Techniques in Large Systems, Prentice-Hall, Englewoods Cliffs.

Mohapatra, P. K. J., and Bora, M. C. (1983) DYMOSIM User's Manual. Indian Institute of Technology, Kharagpur, India.

Mohapatra, P. K. J., and Sharma, S. K. (1985) "Synthetic Design of Policy Decisions in System Dynamics Models - A Modal Control Theoretic Approach", System Dynamics Review, Vol 1 No 1.

Newman, W. M., and Sproull, R. F. (1979) Principles of Computer Graphics, McGraw-Hill.

Nilsson, N. J. (1980) Principles of Artificial Intelligence, Tioga.

Oren, T. I., and Zeigler, B. P. (1979) "Concepts for Advanced Simulation Methodologies", Simulation, Vol 32 No 3.

Pegden, C. D. (1982) Introduction to SIMAN, System Modeling Corporation.

Plattner, B., and Nievergelt, J. (1981) "Monitoring Program Execution : A Survey", Computer, Vol 14 No 11.

Pratt, T. W. (1984) Programming Languages: Design and Implementation, Prentice-Hall.

Pressman, R. (1982) Software Engineering: A Practitioner's Approach, McGraw-Hill.

Pugh, A. L. (1976) DYNAMO III User's Manual, MIT Press.

Raeder, G. (1985) "A Survey of Current Graphical Programming Techniques", Computer, August.

Randers, J., ed. (1980) Elements of System Dynamics Method, MIT Press.

Roberts, N. et al. (1983) Introduction to Computer Simulation: The System Dynamics Approach, Addison-Wesley.

Richardson G. P. and Pugh, A. L. (1981) Introduction to System Dynamics with DYNAMO, MIT Press.

Rohrbaugh, J., and Andersen, D. F. (1983) "Specifying Dynamic Objective Functions: Problems and Possibilities", Dynamica 9

Starr, P. J. (1981) "Identifying Critical Parameters in System Dynamics Models" in Ed. Paulre, E. System Dynamics and Analysis of Change, North Holland.

Thillainathan, V. and Price, D. H. R. (1981) "Estimating Parameters Values in Continuous Simulation Model of an Industry", European Journal of Operational Research 8.

Uhran, J. J., and Davisson, W. I. (1984) "The Structure of NDTRAN: A System Simulation Language", IEEE Trans. on Systems, Man and Cybernatics, Vol 14.

Uyttenhove, H. J. J. (1978) Computer-Aided Systems Methodologies: An Assemblage of Methodological Tools for Systems Problem Solving, Ph.D. Dissertation, State Univ. of New York, USA.

Vermeulen, P. J., and De Jongh, C. L. (1977) "Dynamics of Growth in Limited World : A Comprehensive Sensitivity Analysis", Automatica 13.

Wallis, P. S. L. (1982) Portable Programming, Macmillan.

Weinberg, G. (1971) The Psychology of Computer Programming, Van Nostrand Reinhold.

Weiss, S. M., and Kulikowski, C. A. (1984) A Practical Guide to Designing Expert Systems, Chapman and Hall.

Wolstenholme, E. F. (1985a) "Algorithmic Control Modules for System Dynamics Models", Proc. of the 1985 International Conference of the System Dynamics Society.

Wolstenholme, E. F. (1985b) "A Methodology for Qualitative System Dynamics", Proc. of the 1985 International Conference of the System Dynamics Society.

Zeigler et al. (1979) Methodology in Systems Modeling and Simulation, North-Holland.