DEVELOPMENT OF "APPLES" -- A GENERALIZED DISCRETE EVENT MODEL

Gordon E. Mead and Candice Howard Prince

Union Carbide Corporation

ABSTRACT

This paper describes the development of APPLES (All Purpose PLant Event Simulator), a general discrete event model used for simulating certain classes of chemical plants. The model, which was developed to eliminate the need to build additional single plant models and to replace a series of successful models [3], has demonstrated better results and lower costs.

Special features of APPLES are its open-endedness and its rear-end driven flow. Open-endedness means that the authors have constructed a basic model which is complete in itself, but which can easily be changed and appended. APPLES is rear-end driven, i.e., the user specifies what is to be removed from the system and the program generates what is to enter the system.

This paper presents a synopsis of the model followed by a simplified example. The heart of the paper is, however, the discussion of the stages of the APPLES development. The techniques described can be applied to the development of any large computerized model.

The APPLES project is on-going. At the end of the current phase, another project will start, bringing additions and modifications to extend the usefulness and applicability of APPLES.

INTRODUCTION

For more than a decade, Union Carbide Corporation Chemicals and Plastics Division has used discrete event models of chemical plants as a source of useful information about the operation of the plants. With few exceptions, each model represented a single manufacturing facility [3]. These efforts had to be specific to a single plant because of the depth of detail modeled. Discrete event simulati.. as an information gathering tool was a new field; the details in the model gave a semblance of reality which made it possible to convince users that these new techniques could produce reliable results. Modeling in detail also helped disguise the ignorance of what was and what was not important. For, as Robert E. Shannon says in his book, System Simulation The Art and Science [4]:

> Most systems operate according to the Pareto principle, that in terms of performance and effectiveness there are few significant factors and many insignificant ones. In fact, the rule of thumb is that in most systems 20% of the factors will account for 80% of the performance, whereas the other 80% of the factors contribute to the remaining 20% of the performance. Our problem is to decide which are the significant few.

The development of each new model allowed previously used techniques to be polished and new ones to crystallize. Each simulation was more sophisticated than the previous one.

In 1976, a time was reached when:

1. similarities between models and facilities had become apparent, (to the users of models as well as the developers),

2. cosmetic features (i.e., those which make a model appear realistic but have no effect on results) and irrelevant parameters were recognized, and

3. the development of most major components of a general model had occurred.

By eliminating the cosmetic, simplifying the unnecessarily complex, and concentrating on the important, a general purpose model of most chemical plants could be produced.

APPLES models a wide range of chemical plants that are characterized by most of the following attributes:

1. The plant produces many different materials in discrete amounts.

2. Utilization of multipurpose storage is an important consideration in plant operation.

3. There are stochastic elements such as random equipment outages, random receipt of customer orders, random fluctuations in quality of products, and random results from laboratory analyses of material.

This paper begins with some necessary definitions, then briefly describes an APPLES model of a simple chemical plant. This is followed by an overview of the stages of APPLES system development, highlighting some of the more important model design and building considerations, many of which could be applied to the development of any large computer model.

DEFINITIONS

The following definitions and operating rules are a synopsis of the APPLES model.

The entities of the APPLES model are of the following types [2]:

1. A material is a chemical substance -- a mixture or a compound.

2. An operating unit changes the substance or status of material and is of the three following types:

   (a) An initiating unit (e.g., a reactor) creates from sources outside the system, as if from "nothing", one or more materials at a constant rate and ratio.

   (b) A processing unit converts one or more materials into one or more other materials at a constant rate.

   (c) A terminating unit (e.g., a loading station) transfers material out of the system, either directly or by loading into containers which are sent out of the system.

3. Aim is on-specification material produced during a run of an initiating unit or a processing unit.

4. A shipping order represents the shipment of a single material to a customer, i.e., to a point outside the system.

5. A storage unit (bin, tank, etc.) is a vessel of fixed volume which receives, stores, and dispenses a single material.

6. A storage unit area is a collection of storage units. Each unit belongs to one and only one storage unit area.

7. A storage unit group is a collection of storage unit areas. Each storage unit area belongs to one or more storage unit groups.

8. A resource is an entity which must be available when certain operating units run, and is one of the following two types:

   (a) A crew represents the people who must be present for certain operating units to run.

   (b) A transfer unit is needed when material is moved from one operating or storage unit to another or to the same operating or storage unit.

9. A resource unit group is a collection of resources such as transfer units or crews. Each resource (crew or transfer unit) belongs to one or more resource groups.

10. A container is a vessel of fixed volume which is filled with material and sent out of the system.

RULES GOVERNING OPERATION OF THE MODEL

1. The user is able to (1) connect any operating unit to any storage unit or any storage unit to any other storage unit and (2) to specify the order in which the storage units are to be considered for possible use. To accomplish this, each storage unit is assigned to one and only one storage unit area. Each storage unit area is assigned to one or more storage unit groups. The user specifies the group to which material can be transferred from an operating or storage unit.

2. A transfer unit cannot feed directly to another transfer unit.

3. Transfer units belong to resource groups. The user indicates which resource group an operating unit can use when making a transfer if a transfer unit is required.

4. Orders for shipment of aim material are usually generated from product mix information supplied by the user.

FLOW IN THE MODEL

Most discrete event models are front-end driven, i.e., entities or quantities flowing through the system are generated either deterministically or stochastically, according to a distribution function (specified by the user) and placed into the system at an entry point. These entities or quantities leave the system when their processing is completed.

APPLES is rear-end driven, i.e., the user specifies the quantity of each material to be removed from the system. The product mix of material is then used to generate shipping orders which are tagged with information such as due date, material, and amount. These orders are then used for scheduling the operating units. The initiating units respond to the removal of material from the system by scheduling production of the material needed to fill the first order. The material has no memory. Once a material is produced, it can be used anywhere in the system regardless of where or how it entered the system.

The processing units are scheduled using the knowledge of material needed first to fill orders and the availability of feed materials.

Processing units do not relay information directly to initiating units, but are slaves to them in that all feed materials are supplied by the initiating units. In a similar manner, the terminating units are slaves to initiating and processing units. The terminating units load containers or move materials to outside the system in the sequence needed to fill orders.

Rear-end driven means flow is pulled, not pushed through the system. The resulting system is more easily perturbed by changes in input data. However, "rear-end driven" is better than "front-end driven" in approximating the real-world situations of supply and demand in a manufacturing facility.

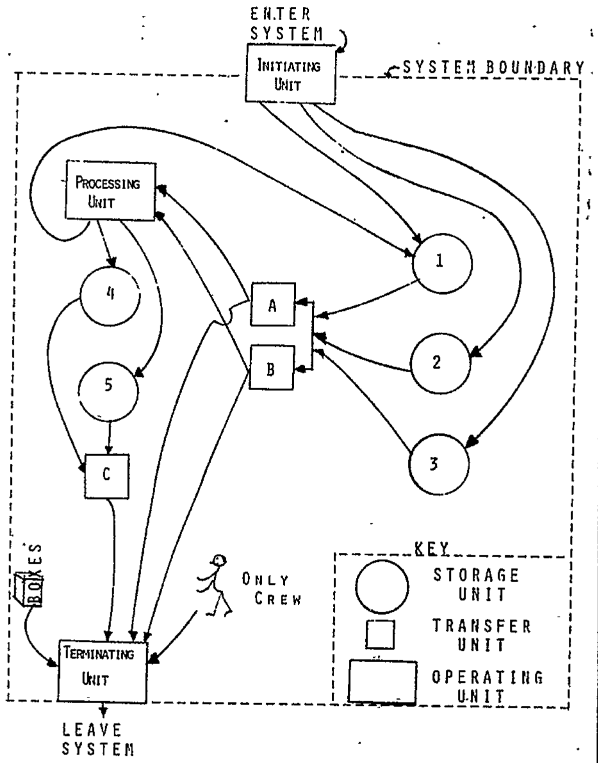
EXAMPLE

Figure 1 shows a very simple chemical plant with one initiating unit, one processing unit, and one terminating unit. The dotted line represents the boundary of the plant, i.e., system being modeled. Material enters the plant via the initiating unit, and leaves the plant by a shipment of containers.

The initiating unit produces two aim or "good grade" materials, X and Y. The processing unit uses materials X and Y to make product PX. Materials PX and Y are shipped out of the system to customers. Material X has corresponding inferior grade materials, XNA and OFF while material Y has YNA and OFF.

FIGURE 1

A SIMPLE CHEMICAL PLANT



This plant has five storage units designed as STU1 through STU5. These are aggregated into storage unit groups and areas shown in Tables 1 and 2. Note that since the order in which we consider the storage units is important, group STU-GRP3 consists of storage units STU4, STU5, and STU1, not STU1, STU4, and STU5. Materials are stored in the storage unit groups indicated in Table 3.

The initiating unit produces materials X, Y, XNA, YNA, and OFF. Usually, inferior grade materials are produced between the production of two aim materials. The sequence in which materials are produced by the initiating unit is determined by either the automatic scheduling algorithm, which locates the material needed first per customer orders, or by a user input schedule. Materials are fed from the initiating unit to the respective storage areas as indicated in Table 3. From there, materials go to the processing unit (to be used when producing PX) or to the terminating unit (to be loaded into containers). The formula for making product PX indicates that feed materials X and Y are needed. When material PX comes from the processing unit, it is sent to storage units STU4, STU5, or STU1. Shipping orders and their due dates determine when materials are loaded and shipped, and in which containers. Materials cannot leave the system unless a shipping order for materials and the proper containers are available. In this example, the containers are called boxes. The information about materials is summarized in Table 3.

Resources are, in effect, auxiliary entities needed when activities take place. In this plant, the resources are transfer units A, B, and C, and crew ONLY CREW. The transfer units are needed when material is fed to the processing unit or terminating unit, and the crew is needed when the terminating unit is loading out material.

An outage is really any operating unit run calling for the unit not to make or ship any material for a period of time. In effect, an outage preempts the current run of material from the unit. At the end of the outage, the operation of the unit on the current run is resumed.

Please understand that this is a very simple plant. An actual plant would include many more materials, operating units, etc.

TABLE 1

Storage Unit Areas

Area #	Area Name	Storage Units
1	Area 1	STU1
2	Area 2	STU2
3	Area 3	STU3
4	Area 4	STU4, STU5

TABLE 2

Storage Unit Areas

Group #	Group Name	Area #'s	Storage Units
1	ST-GRP1	1,2,3	STU1, STU2, STU3
2	ST-GRP2	3	STU3
3	ST-GRP3	4,1	STU4, STU5, STU1
4	ST-GRP4	4	STU4, STU5

STAGES OF MODEL DEVELOPMENT

This paper is not only about the APPLES model, but also about its development. This model is an offspring of many parent models of specific chemical plants. These old models were tedious and costly to build and had difficult to use input and output. In effect, these were "throw-away", non-reusable models. APPLES is intended for general use on different chemical plants in a variety of ways.

The stages in the development of the APPLES model were:

1. determination of the need for such a model and selling the concept to management,

2. development of specifications and definition of entities,

3. design and coding of the editor program and the input subroutines of the simulator program,

4. design and coding of the simulator program proper,

5. debugging, testing, and validation, and

6. production running of the model.

Selling the Model

Since both authors work for "service groups" within the corporation, it was necessary to sell money-supplying users of the benefits and cost savings to be derived from developing and using APPLES rather than an individual model for each manufacturing facility. These benefits included quicker results, a more "user-oriented" system, easier changing and correcting of programs, and, consequently, more cost effective modeling.

A model of a single facility, such as was used until 1976, took five to eight months and tens of thousands of dollars to develop. Because of costs and time, such models were impractical to develop for all but the largest plants and even then only when major engineering studies were being undertaken. The modeling was most often done under conditions of extreme stress, including long hours and at times other than 8 a.m. to 5 p.m. to assure faster computer turnaround. In contrast, APPLES now provides an off-the-shelf system that can be used for even a small study.

The specific models, developed under severe cost and time constraints, required information to be input in a form easiest for the program and the programmers. As a result, the users had to reform much of the data and usually had to make several computer runs to remove all errors and inconsistencies from the data. The users needed a knowledge of the program logic of the specific model in order to diagnose problems in the data and in the model. Since the new APPLES model is "user-oriented", most problems can be diagnosed either from the many error messages within the Editor or from the APPLES user's manual. An editor program allows the user to input his/her data in a logical fashion. This program also supplies defaults for most values not input and checks for missing information which could cause the program to terminate during execution. The users need only know the chemical plant being modeled and apply intelligently the information provided in the extensive APPLES user's manual.

Program maintenance, the inevitable changing and appending of programs, was very difficult on the single plant model. Changes to input parameters, which deviated much from those used to develop the

TABLE 3

Summary of Data on Materials

Material	Corresponding AIM	Made on Unit	Feed to Unit	Stored in group #	Shipped	Grade
X	X	init. unit	P. unit	2	no	aim
Y	Y	init. unit	P. unit & T. unit	3	yes	aim
XNA	X	init. unit	P. unit	1	no	near aim
YNA	Y	init. unit	P. unit	1	no	near aim
OFF	X&Y	init. unit	P. unit	1	no	off grade
PX	---	P. unit	T. unit	4	yes	aim

model, often caused logic errors which necessitated program updating. Since the programs were hastily written and designed for a specific set of input data, these alterations were often difficult to implement and debug. In fact, one to three months elapsed time was required for all but minor model revisions. APPLES, because it has been systematically designed and thoroughly tested, can be used for a wide range of parameters and operating conditions. There is only one well-documented program to maintain instead of a dozen programs, some of which were kept in source form on well-worn 80 column cards.

In the final analysis, what really sells a model is its cost effectiveness. Although APPLES cost roughly four times as much to develop, this cost is quickly being recovered. Running a study on a single plant can now be accomplished in a fraction of the time and for a fraction of the cost.

Development of Specifications and Definition of Entities

The second stage, development of specifications and definition of entities for the model, was lengthy and sometimes discouraging. The authors walked a thin line between being too general (which would result in a model too complex to use and too expensive to run) and being too specific (generating a model too difficult to use for modeling a variety of chemical plants or too simple to have meaningful results). This stage is critical since, unless it is done well, difficulties may occur in subsequent stages. Programmers frequently find such a lengthy planning stage traumatic because there is little concrete evidence (usually measured in lines of code) to show for one's labors.

During this stage, it was decided that our system should be "open-ended", i.e., it would be easy to change and changes would not cause undesirable repercussions. The open-endedness of APPLES allowed the authors to construct a basic model which was complete in itself but which could easily be changed and appended. To accomplish this, it was decided that:

1. All foreseeable program changes would be considered during the design phase.

2. The system would be well documented, i.e., readable code and meaningful comments in the program, and extensive user's and programmer's manuals.

3. Structured programming techniques would be used, resulting in code that would be easy to read, change, and debug.

4. During the design phase, users would be consulted as to their ideas about what the model should include.

In reference to the APPLES model, open-endedness means, e.g.:

1. There are no limits to the number of initiating units, materials, etc. that a user can input. This is accomplished by means of dynamic allocation of storage and list processing techniques.

2. New types of initiating units, storage units, etc. can be easily added when needed.

3. Changes to the programs of the system have no effect on the results from old sets of data.

Also in this phase, a rough layout of program and system design was set down. It was decided that the model would be run by executing two programs -- first a card editor program written in PL/I, then a simulator program written in SIMSCRIPT II.5. PL/I was chosen for the editor because it is a language with extensive list processing and string handling facilities. SIMSCRIPT II.5 was chosen for the simulator program because it is a simulation language that is efficient to run, essentially self-documenting, and well maintained.

Also in this stage, the specifications were written. Much of what went into the specifications was later to become part of a user's manual. This preliminary user's manual is, in effect, a written definition of the model [1]. While writing is often a task much disliked by programmers and model builders, the benefits of this careful documentation are great.

Design and Coding of the Editor Program and the Input Subroutines of the Simulator Program

In the third stage of development, the input cards were laid out, the editor was designed and coded, and the simulator program subroutines to read the editor output were written. The best model with poor input card layout and without printed forms can be a tedious tool to use. The editor is one significant difference between APPLES and the individual specific chemical plant models that Union Carbide used previously.

The APPLES editor makes two passes through the input cards. In the first pass, the cards are segregated by card type and obvious data errors, such as alphabetic characters in numeric fields are detected. The second pass reformats the data for use by the simulator program while detecting more subtle errors, such as a missing operating unit definition or a maximum run length less than normal run length. The simulator program reads the output of the editor program then performs a discrete event simulation of the chemical plant described in the data. All entities and arrays are dimensioned by values which the editor calculates and passes to the simulator program.

The editor program consists of 17 PL/I external subroutines each of which contains several internal PL/I subroutines. The modular structured design makes the program easy to maintain and change. For the interested reader unfamiliar with structured programming, a very good reference on the subject is Edward Yourdon's Techniques of Program Structure and Design [5]. Likewise, the read and initialize subroutines of the simulator were modularized. Many blank data fields are passed to the simulator program; this leaves space for each addition of new data items without change to other parts of the program.

Design and Coding of the Simulator Program Proper

In the coding of the simulator, much thought and care was given to assure that the program would be easy to maintain and that program revisions and additions would be transparent to the users. To help reach these goals, techniques such as elimination of constants, the use of long variable names, and structured programming were adopted. Most constants were assigned variable names, for example, the variable EMPTY is set to one. The use of meaningful variable names, regardless of length, was stressed, for example, LAST.TIME.UNIT.DOWN.OR. IDLE.

Structured programming techniques, including modularization of code and simplification of flow of program control were used. GO TO's and statement labels were allowed only for the "case" condition where a choice of one of several alternatives is made. Even there, all cases had to go directly to the same termination point.

SIMSCRIPT II.5 allows for implied subscripts on entity attributes. Both PL/I and SIMSCRIPT II.5 provide for defaulting variables not identified as global (external) to local (internal) variables. The use of these simplifying features was verboten, for what is clear at the coding stage will be confusing later to someone else doing program maintenance. The rule used was array variables are subscripted and all local variables defined as to type.

The clever coding of today is the confusing code of tomorrow. Only if it is superior should such coding be used, and only if accompanied by adequate documentation. The authors subscribe to the KISS principle, "Keep It Simple Stupid".

Experience indicates one tends to read and rely on comments rather than coding, and that inevitably some comments are wrong. The use of modular coding, simplified flow of program control, meaningful variables names and the "English-like" features of SIMSCRIPT II.5 made the code self-documenting. Comments were relegated to explanation of difficult algorithms and to header comments explaining the overall function of each routine.

Debugging, Testing and Validation

This stage was long and tedious. Test decks were designed and temporary print statements were used to check all calculational paths. Various program functions were localized by use of modular routines to reduce program interaction. Because the number of possible calculational paths through APPLES is astronomical, synthetic input, designed to test as many possible combinations of input and logic flow, was run and debugged.

Validation of the APPLES model was extremely difficult. Input from previous models was recoded and new results were compared with old. Output results were examined to see if they appeared to be realistic. Finally, the model was examined event by event to see if it conformed to model specifications.

Production Running of the Model

A comprehensive user's manual, including an explanation of all input cards, was written. For each card type, all fields are defined and defaults and sample values given. The manual shows a typical set of input cards and output data. A two day seminar is available for instructing users in the program's use. On-the-job training is available to new users for new applications for APPLES. In production status, the compiled programs are kept on a direct access data set and run by nonprogramming users with only Job Control Language cards and input data cards.

CONCLUSION

APPLES is a large discrete event model suitable for use on a certain class of chemical manufacturing facilities. It replaces individual models of these plants and eliminates the need for expensive, tailor made models for new and existing Union Carbide facilities. APPLES can simulate these plants more quickly, more easily, and less expensively than the individual models.

The road to a general model would have been rocky, indeed, without experience gained from the development of the specific models. These precursor simulation activities helped the authors to discern the important from the trivial, thus keeping APPLES to a manageable size, and to develop efficient algorithms, thus making APPLES a reasonably fast running system of programs.

While many of the techniques discussed in this paper may seem obvious, they combine to form an approach to model building which improves the chances of success where failure is a real possibility. The authors bring them together here so that others may benefit from their experience. Some of the methods, especially those in the coding phases, are really techniques of good computer programming and are not specific to model building.

Discrete event simulation models give practical, useful results. However, unless efforts are made to ensure that they are general enough to meet future needs, easy to change where necessary, and systematically programmed and debugged, these models quickly become expensive and obsolete, particularly in the rapidly changing world of industry.

REFERENCES

1. Donelson, W. S. "Project Planning and Control", in Datamation, June 1976, pp. 73-77.

2. Mead, G. F. and Prince, C. H. APPLES Functional Specifications Phase I, a private document.

3. Mead, G. F., and Salisbury, P. L. "Use of Stochastic Simulation in Design and Operation of Process Facilities" in Proceedings of the 1975 Summer Computer Simulation Conference, Volume 1, pp. 285-291.

4. Shannon, R. E. Systems Simulation the Art and Science. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

5. Yourdon, E. Techniques of Program Structure and Design. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.