A CORPORATE MODEL IN APL

by

R.E. Williams

Mutual Life Assurance Company

A. P. L. as a computer language for modelling has strengths and weaknesses . On the plus side are :

1. Its array handling and computational techniques ;

2. As an inter-active language it is relatively easy to debug programs and assemble systems ;

on the negative side are :

1. Documentation for input data and global variables is often inconvenient or difficult and hence inadequate ;

2. In making efficient use of A. P. L. it is desirable to try to minimize the use of nested loops and procedures ; this may however lead to more complex calculations .

This paper discusses a design for a corporate model which attempts to emphasize the strengths and compensate for the shortcomings of A. P. L. as a computer language for modelling .

Basic Principles

1. Flexibility

   It should be reasonably easy to modify or expand the model - e.g. add an expenses submodel ; make use of graphics ; modify submodules ; add new variables ; update and keep track of input assumptions ; suppress reports ; create new or temporary reports ; etc. .

2. Autonomous submodels

   The responsibility for making projections should reside in the area most knowledgeable about the separate items involved and their inter-relationships . Thus each line of business and service area is given control over the procedures used to project the results of their future operations . This is reasonable if there are A. P. L. programmers dispersed through the company .

3. Fully shared control

   It is desirable to avoid reliance on one coordinator or central authority for the use of the model or a submodel . Thus input assumptions can be modified and the model used to test such assumptions by any user who has an interest .

Design

Each submodel consists of :

1. a Sequential file

   The first record in the file describes its STATE and indicates whether the input data has been updated and when ; it also identifies the time of last update for each of the companion files associated to the other submodels and indirectly affecting some of the output data contained in the file such as net investment income and income taxes .

2. an A. P. L. shareable workspace containing

   a. Function-group to massage the sequential file(s)

      This is essentially a mainline program which accepts the name of the specified file and possibly some control information and proceeds to generate projections from the input data , writing the output to the same file and updating the STATE vector . The mainline would be supported by subprograms and fixed variables organized under a group name .

   b. Documentation character array

      This is a character array containing in each row a description of a corresponding row in the numeric array determined by the sequential file . The first sixteen columns contain an abbreviation suitable as a row heading .

   c. Fixed variables (e.g. to specify defaults)

      For example , fixed variables are needed to specify the name of the default sequential file , the rows used for input data , the rows containing specific data to be transferred to the allocation submodel .

   d. Function-group to produce reports

      This would be a mainline and supporting variables and subprograms needed to produce reports local to the submodel .

In effect each workspace represents READ ONLY information and procedures whereas the sequential files are shared for reading and writing data .

The CORPORATE MODEL consists in total of a submodel for each of the following lines or service divisions

1. Individual Insurance (II)
2. Individual Annuity (IA)
3. Group Annuity (GA)
4. Group Life (GL)
5. Group Health (GH)
6. Segregated Funds (SF)
7. Investment Services (IS)
8. Corporate Allocation (CA)

The flow of control and data is as follows

1. File names (if not to be defaulted) and control information concerning reports to be produced are passed to a driver program .
2. The file names are passed to the CA submodel .
3. The CA submodel reads the files associated to each of the lines , determining REVENUE ACCOUNT and cash flow information by line split par-non par where appropriate. In any case where a file has not yet been massaged , the CA submodel calls the related function-group to process the file before reading the file . The STATE of each file is indicated by its first record . The SF , IS , and CA files are updated with the available data .
4. The IS function-group is called to massage its file in effect determining new assets and investment income .
5. The CA submodel then reads the IS file and allocates the investment income by line ; it does the total company tax calculation and allocates the tax by line ; it updates all the files with this information changing the STATE record in each file to so indicate .
6. The driver program now calls the function-groups required in order to produce the reports specified . The CA function-group would produce the INCOME STATEMENT and BALANCE SHEET possibly split by line . Each submodel can produce its own STATEMENT of ASSUMPTIONS , SOURCE of GAIN AND LOSS , INTERNAL BREAKDOWN of OPERATIONS etc. .

Objectives of the Design

1. Consistency with the basic principles .

2. Facilitate the creation and update of input data .

3. Transmit processed data from any source .

4. Tie consistent input and output data together permitting separate files in response to what if questions .

5. Avoid unnecessary reprocessing .

6. Control the reports generated .

7. Adequate documentation of data .

System Software and ' Home-Made ' Utilities

1. VSAPL under TSO (MVS Release 3.8) .
   We currently have release 3.5 and are in the process of installing release 4.0 with Graphical Data Display Manager (GDDM) . These versions are distinguished by the Extended Editor and Full Screen Manager . The new version will have full graphics capability and an A. P. L.-Session Manager product .

   Extended Editor

   The advantages of the Extended Editor are implicitly relied upon in the design of the model . It is easy to create and modify character arrays which are used to document the file associated to a submodel . Functions and the documentation array can be edited in parallel . Lines of code can be moved easily between functions permitting a long program to be organized into integral sub-modules .

   Full Screen Manager

   The Full Screen Manager auxiliary processor (124X) is used to create a data entry program which ties the documentation array to the input data and has other useful options described below .

2. File I/O uses auxiliary processor 111 and OS Sequential files written in A. P. L. format (VAR option) .

3. TSO-Session Manager is a very useful complement to A. P. L. on line at a video terminal (with PF keys) . It permits the screen to be divided into windows which can be positioned to view any portion of the input stream and/or the output stream generated during a session . Furthermore relevant sections of either stream can be written to a data set or printed on the main printer avoiding the use of typewriter or decwriter terminals .

4. Dataentry Utility Function

   This function is used to create , update or view the numeric data in a file . Depending on which rows and columns are specified the screen is formatted so that documentation information such as the row number , column numbers , and character description are written on the screen at low intensity and the numeric data at high intensity . The numeric data can be overwritten or operators can be specified to repeat a previous field , multiply by a factor , or increase by a constant . Other options using the PF keys are planned.

5. Documentation-Trace Utility Function

   This function takes the name of a submodule , left and/or right arguments for this submodule , an array name referenced in the submodule and a documentation character array referring to the referenced array , creates a copy of the submodule , invokes the STACK auxiliary processor , and executes the submodule , pausing at each line of code in which the array is referenced and printing the description of the rows specified and other trace information required . This utility function makes it feasible to write submodules which process a single array by massaging its rows since it provides adequate documentation for the calculations taking place . The lines of code themselves will be visually unappetizing - the opposite of an ' English-like self documenting high level language ' such as PL/I , Cobol etc . However it is easier to make use of ' f/ ' and ' f.g ' for appropriate A. P. L. primitives f , g .

6. Printfile Utility Function

   This provides an unformatted dump of the numeric data in a file with description from the assosciated documentation array for selected rows and columns . Alphabetic or other row order can be specified with an alphabetic index .

7. Filewrite Utility Function

This assures a uniform file structure including a STATE vector as the first record .

8. Fileread Utility Function

This assures that the first record is not confused with the MODEL numeric data .

Evaluation

Since we are still in the process of testing and assembling the submodels it is premature to evaluate this A. P. L. application . Nevertheless the following comments can be made :

- the data management and documentation techniques have been well received ;
- there have been some scheduling problems depending on the level of sophistication attempted by each line of business; the Individual Insurance and Annuity lines are designing an extensive business-in-force model to feed their corporate submodel ;
- projections are made in all years before the books are closed in earlier years ; final adjustments in cash flow affecting cash and investment income must be made with care to keep the books in balance for each year ;
- the inter-active and computational nature of A. P. L. permits the data to be analysed on line to answer questions not anticipated by the Corporate Model .

*****

DISCUSSION OF PRECEDING PAPER

BILL JOHNSTON: You have one array for your input structure - do you have an array which carries the results of the output items or are the output items just generalized as a print down the side?

BOB WILLIAMS: No, the input and output assumptions or the input assumptions and the output results are tied together in the same array.

BILL JOHNSTON: There is one array carrying all input and all the primary outputs?

BOB WILLIAMS: Yes, there is one array that carries all that, and there is a convention with which to keep these separate if we want to do this type of calculation.

BILL JOHNSTON: Having all the data in one array would make it into a multiple work space type of structure. Do you generally work in just one work space at a time?

BOB WILLIAMS: There is one driver function which pulls in the various work spaces that are required; if I say one array, I mean that each sub-model for a line of business has one array because they are responsible for their own assumptions and their own operating results. But there is a driver program which will pull in their work space, the one that contains their version of their sub-model, if you like, and that one there will read their file and update it if it is necessary. It has raw input information in it. Without the output results it will call their sub-model processor and update it to a final stage where it is ready to provide input to the investment program.

BILL JOHNSTON: Now, say, data are on files and not in the work space.

BOB WILLIAMS: The data are on files, the programs are not. The data can be updated by anybody who wants to run the model. There is shared control, they do not have to come to me. If the people of that individual area want to run the whole sub-model by varying their own assumptions to see what happens, they can do that. They can specify separate output files so that they do not muck up the output files that were created at the last "standard" run of the model.

BILL JOHNSTON: You see, there are probably values assigned to computer language; people say a computer language is good if it is self-documented; if you can read a line of code and it is understandable by a layman; and other people say that a computer language is good if it is easy to program calculations, if they can be done precisely and efficiently.

BOB WILLIAMS: Well certainly this approach to running a corporate model in APL is not meant to make it self-documented and understandable, because all you see if you read a line of code is the name of the array - a bunch of operations on it. There is no way to understand who or what is going on that volume. You would have to take it apart. But if you add these functions, which operate in a sense on your programs, and sort of "undress" these things, which makes apparent just exactly what calculations are taking place, by tying the documentation array to the actual row, the actual line of code in which a row is processed or the series of rows are processed, it turns out to be just fine.

VOICE FROM FLOOR: With the APL corporate model, suppose you are wondering about the ordinary life line only. I use 7 or 8 arrays but they are all in one block in the file and they are all written as a group. After the run is over, I have a small FORTRAN program which is submitted and you simply print all these arrays on the high speed printer. You can probably do the same things with the files of the high speed print and the arrays are labelled by that program. And then, when I am working on the system, I have their names and labels across the top and down the side. I use that to indicate what the age matrix is, for example.

BOB WILLIAMS: Well, I mean, there are obviously twice as many as anybody could take care of, and in the use of APL there are "blind alleys" that you might find yourself looking down, and you might get terribly confused. For example, in the asset model we have, we end up with almost 700 variables. We are describing the investment program in terms of a commitment; we are keeping track of the outstanding commitments on each asset type. We have programs for 15, 16 different types of assets.

AL CHRISTIANS: Does your documentation in systems give you notice in some kind of automatic way if you change the code, so that you do not have to change the documentation, or do you have to just remember to change them both when you have changes?

BOB WILLIAMS: You should not be reordering your array; for example you should not add a new row between the thirteenth and fourteenth row.

But isn't there program documentation too? Could you, maybe,

insert some lines of codes?

The main programs that process the arrays have documentation, of course. They destroy what each line is doing. But it is not convenient in tracing calculations or something like that. You always have to look at the function code, at the function, as you are tracing.

There is nothing very magicical about this. Basically, you are simply maintaining, in parallel, a documentation array to correspond to your numeric array. So, for your documentation array, you have one variable, called "docu" or something, whatever you like, and you have some in the first row here. You have maybe some short abbreviation, premiums, and over there somewhere you can have a lengthier explanation - exactly what premiums those are. And you can have claims, you can have invested income, etc. The point is that you have this available and you can bring it to the function, when you peel the function which is executing and working on some array which is being specified on all sorts of indexes here, depending on what calculations we want to do. And you can use this documentation array to refer by using these indices. You can describe what calculations are taking place. If you do not have something like that, working with a function that just has a simple R all over the place, is a misery.

Now the other thing that I wish to point out is that the release we are using has an IBM version of APL, that has an extended editor for APL. If you do not have a version that has such a thing, I suggest you get it, because it is extremely useful - you can parallel edit functions and character variables and switch between them. So

it becomes very easy if you want to write some line of code here, which is going to determine net income, like adding a bunch of these rows and subtracting a bunch of others. If you want to know which rows to choose, all you got to do is parallel-edit your documentation array and pick out the numbers as fast as you get them in.

If any of you, who might want to try such a thing, would find that you run into problems, I would be happy to share our own experience with you and try to answer any of your questions.

VOICE FROM FLOOR: It sounds as if some of these things may be software-dependent and I do not think that you want to go too far afield here - I gathered yours is operating under TSO.

BOB WILLIAMS: Yes, we are operating under TSO, that's true. But this extended editor here that IBM provided (under release 73.5 and next release 4) can be used whether you operate under CMS or under TSO.

BOB WILLIAMS: Well no, we do run our APL under TSO, and I think there is another standard option and this extender is available.

The other thing that is very useful, by the way, for those of you who may have a chance to decide whether or not you get to 90 with APL is a session manager product. A session manager product is one which keeps a continuous log of all the work you have done during the session and you can go back to various things that you have done by strolling up or down on the screen. You can also print different sections of your work for documentation purposes. You can print different sections of your work on, say, the main printer or maybe on an auxiliary printer, or something like that. I am not sure and I

don't know, it might be a little expensive, but if there are enough people who would use it then it would probably be very worthwhile.

I know that I noticed that, since we have had the APL under TSO, I enjoy proramming in APL more than before that.

VOICE FROM FLOOR: You did not mention any constraints that you put on the programmers as to how big a statement they write.

BOB WILLIAMS: One of the problems that I find when I turn people loose is that they have the whole program in one statement. Well that's a question of style. I try to be as unrestrictive as possible when style is concerned. I want people to do their own thing when they are programming in APL, but, on the other hand, I wanted to provide enough tools so that they would not get away from following this approach here where, instead of using names like premiums for your array, or claims or whatever, you are always using the simple R or X or Z or whatever, which always makes it look as if it's sort of a similar code or something like that; it does not stop.

VOICE: You do not get any problem with modification then?

BOB WILLIAMS: No. I never did.