A DATA-MODEL INTERFACE FOR MODULAR DYNAMIC SIMULATION

Friedhelm Drepper

ABSTRACT

As an alternative to simulation languages like Dynamo a Fortran based simulation aid is presented which offers more flexibility at the cost of some conceptual guidance. This tool is essentially a Data-Model Interface (DMI).

It supports a modular model structure and is especially suited for the simulation of large dynamic systems with a simple event structure. Its simulation concept allows unsorted statements as well as coupled equation systems extending over several modules.

INTRODUCTION

At the 1977 Winter Simulation Conference the energy flow simulation model designed at the KFA (Kernforschungsanlage) Jülich was described in detail. /3/ It is a model which tries to simulate the energy economy of the Federal Republic of Germany. From a methodological point of view the main features of this model can be described as follows:

The model is a long term dynamic simulation model. The dynamics of the model do not originate from many discrete time events but from many exogenous time series and some differential- and integral equations. (differential or integral with respect to time)

The model is a relatively large one. It contains more then a thousand exogenous variables, most of them being time dependent statistical data and strategy or policy parameters. It consists also of a few thousand algebraic equations which reduce to a few hundred, when expressed as vector or matrix equations. Those equations evaluate nearly the same number of endogenous variables. A big portion of them takes part in linear or nonlinear simultanous equation systems. So far it sounds rather difficult to solve these equations but it turns out that the interdependency of the variables can be written as a recursiv equation system with a relative small back coupling.

For models of that size special programming aids in addition to general purpose languages, like Fortran and PL1, are necessary. Some of the most important additional needs for the implementation of large dynamic simulation models are the following:

Automatic input of exogenous time series and constants combined with a diagnosis of the completeness and redundancy of the data base.

Automatic output of time series for predifined variables.

Automatic handling of unsorted assignment statements.

These needs are perfectly satisfied e.g. by the dynamic simulation language Dynamo. /2/ For this reason the energy modelling in Jülich started by using this higher programming language. In the course of time two main disadvantages arose out of the fact that Dynamo is not a general purpose language, namely:

No possibility of including special algorithms such as for optimization /1/ and solving coupled equation systems. No possibility of introducing a modular model structure.

The second shortcoming turned out to become a vital point once the model had been developed to the point where it could no longer be maintained by a single person.

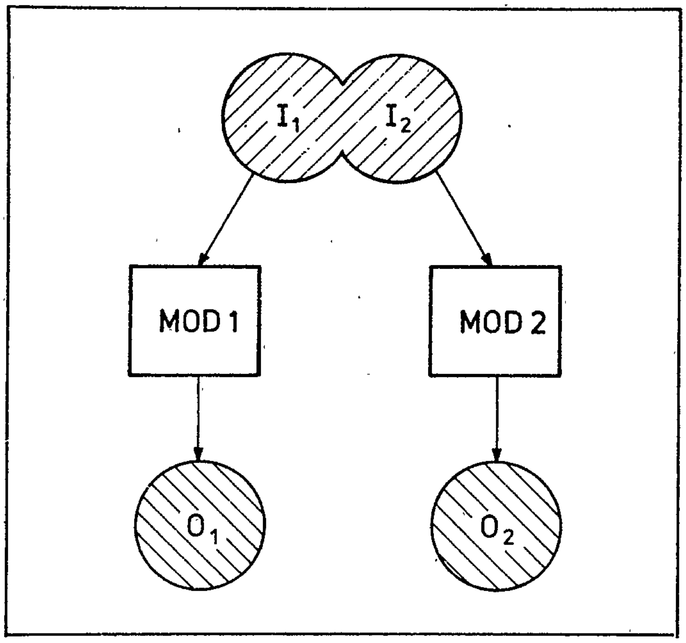
These disadvantages have led to the design of a new type of simulation facility at the KFA in Jülich. In contrast to Dynamo standard FORTRAN is used to formulate the modules. Basically this facility can be described as a Data-Model Interface (DMI). In the first instance this interface was designed to support the energy modelling in Jülich, but in principle it may be used for a wider class of large dynamic simulation models with a similar structure.

The introduction of modularity into dynamic simulation is associated with two major problems. In the first section the problem of data transfer from the data base to the modules as well as between the modules will be discussed. The second section will deal with the consequences of modularity with respect to the simulation concept. The last section will finally describe how these concepts are realized within the DMI.

DATA TRANSFER REQUIREMENTS FOR MODULAR SIMULATION

In this chapter the basic problems of data transfer in a large modular dynamic model system are discussed. The most basic requirement is that the input data are separated from the source programs and put into a special data set called the data base. The data base may contain constants as well as time series. We will refer to both as data items. Every data item is identified by the name of the exogenous variable to which it belongs. In fig. 1 we see two modules

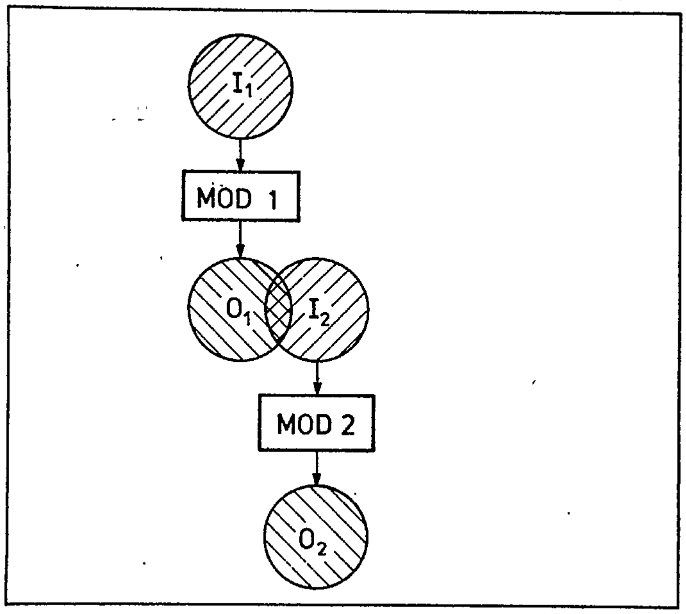
Figure 1
Two Independed Modules



getting a set of data items as input from the data base and producing another set as output which may be given back to the data base. The overlapping of the two input sets symbolizes that both modules have access to the same data base.

It can now happen that some of the exogenous variables of module 2 are endogenous variables of module 1. This is indicated in fig. 2 by an overlap between I_2 and O_1.

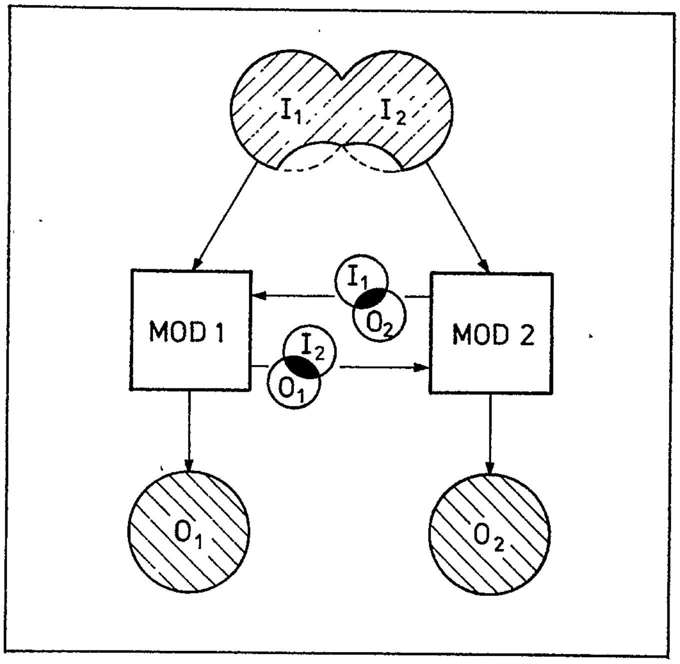
Fig. 2: Serial Coupling of two Modules



In this situation where only module 2 is dependent on module 1 and not vice versa the coupling can be achieved simply by executing the modules successively and using a part of the output of module 1 as input to module 2. The only requirement for this is that the output and input formats are compatible.

The linking of two dynamic modules becomes more difficult when there is also an overlap between O_2 and I_1 so that back coupling may occur. This situation is illustrated in fig. 3. The possibility

Fig. 3: Simultanous Coupling of Two Modules



of backcoupling requires that the input into each module comes only partially from the data base whereas the other part comes simultanously from the other module. The data transfer features of the DMI supporting modular dynamic simulation can be summarized as followes:

The DMI takes care of the data transfer between the modules via a Fortran COMMON Block for all global variables.

It takes care that the input data sets have to be reduced by the overlaps between I_1 and O_2 on the one side and I_2 and O_1 on the other.

It gives warnings if the data base is not complete .

It checks whether the output datasets are disjunct which means it gives a warning for variables which are endogenous in more than one module.

It avoids multiple input of the same data item for different modules symbolized by the union between I_1 and I_2.

To realize these features the DMI contains a precompiler with an analysing capacity to find and to distinguish between all endogenous and exogenous variables of the module system. It takes every standard Fortran IV source program as input and produces a so called Global Variable Map (GVM). The GVM has the form of a matrix with lines for every global variable name and columns for every module merged into the whole module system. Every non COMMON variable of type REAL is taken as global variable if it is not explicitly declared as local.

Every global variable which occurs either on the left hand side of an assignment statement, in a READ-statement or a DATA-statement is recognized as endogenous. All other global variables are treated as exogenous variables. In addition to this implicit type assignment there is also the possibility of an explicit type assignment. This is necessary e.g. when an argument of a SUBROUTINE is endogenous within that SUBROUTINE and not in the calling program. In that case this variable must be declared explicitly as endogenous. On the other hand, when an endogenous variable is explicitly declared as exogenous, this variable belongs to a third type. For a simple variable this third type means that the endogenous variable needs an initial value from the data base. A diagnosis is given in the case when such a variable gets a whole time series from the data base. For vectors or matrices there is the additional possibility that the elements of these arrays belong to different types. In that case only a lower level warning is given on encountering a time series in the data base.

In addition to the information whether global variable is endogenous, exogenous or both in a particular module each line of the GVM contains also the name and the dimensions (if any) of a global variable.

The sequence and length specification of the global variables are valid for the whole module system. Adding a new module or changing an old one never changes the sequence of the already existing lines of the GVM. The lines for new global variables are always added at the end. The final action of the precompiler is to introduce a COMMON BLOCK containing the global variables of that particular module and dummies for the global variables of all the other moduls into the source of that module. Changing a modul requires only this module to be processed again by the precompiler.

A SIMULATION CONCEPT ALLOWING SIMULTANOUS EQUATIONS AND MODULARITY

A second major problem in a modular model system is the handling of unsorted statements. A reordering of all assignments of a modular system is in general not possible without destroying the modules as a unit. Therefore the Jülich DMI takes a different approach to this problem.

Let us consider a system of 3 equations which would be simple recursive when evaluated from bottom to top.

$$\begin{aligned} Z &= g(x,y) \\ y &= f(x) \qquad (1) \\ x &= c \end{aligned}$$

Interpreting this equation system as a set of assignment statements we can define a series of vectors (x_i, y_i, Z_i) $(i = 0,1, \ldots)$ by repeating the execution of the statements (1).

$$\begin{aligned} Z_1 &= g(x_0, y_0) & \quad Z_2 &= g(c, f(x_0)) \\ y_1 &= f(x_0) & y_2 &= f(c) \\ x_1 &= c & x_2 &= c \end{aligned}$$

$$\begin{aligned} Z_3 &= g(c, f(c)) \\ y_3 &= f(c) \qquad (2) \\ x_3 &= c \end{aligned}$$

One can see that after 3 executions of the system (1) the solution becomes independent of the original values (x_0, y_0, Z_0), so that all further executions result in no change of (x,y,Z). For this reason (x_3, y_3, Z_3) is the solution of the equation system (1).

The only disadvantage of this method lies in the fact that the computational effort is 3 times as big as in the simple recursive case. But one can easely see that all other arrangements of (1) result in a smaller number of iterations to get the final solution. In fact it turns out that even in a relatively large module which is not statistically but logically designed it is no big effort to keep this number below 10, especially when the user is aware of this problem during the testing phase of the modules.

This disadvantage becomes completely negligible when one makes use of the fact that the same method can also be applied to a wider class of equation systems, namely to recursive equation systems with a relatively small additional backcoupling. In fact this is a very important class of equation systems in economic modelling.

It is useful to prove the convergence of this method first only for linear systems of equations with small back coupling. With the introduction of n-dimensional vectors and matrices such a system may be formulated as follows (in the following we will reserve capital letters for matrices and underline the vectors).

$$\underline{v} = (A + \varepsilon B)\, \underline{v} + \underline{c} \qquad (3)$$

In the first instance we assume A to be a matrix corresponding to a simple recursive equation system

$$A = \begin{pmatrix} 0 & & & \\ a_{21} & 0 & & 0 \\ \vdots & \ddots & \ddots & \\ a_{n1} & \cdots & a_{nn-1} & 0 \end{pmatrix} \qquad (4)$$

and B may be a general n x n matrix whose norm is of the same order of magnitude as that of A. Then weak backcoupling can simply be expressed as

$$\varepsilon \ll 1 \qquad (5)$$

we define now a series of vectors $\underline{v}_i$ (i = 0, 1, ...) in the same way as before. That means we interprete (3) as a set of assignment statements and take the values of $\underline{v}$ before execution of (3) as the old values and the values after execution as the new ones. This construction rule must not be confused with simple matrix multiplication according to equation (3). However the evaluation of (3) including insertion of the newly assigned values into the statements below can also be written as matrix multiplication.

$$\underline{v}_1 = P(\varepsilon)\, \underline{v}_o + Q(\varepsilon)\, \underline{c} \qquad (6)$$

$P(\varepsilon)$ and $Q(\varepsilon)$ are also n x n matrices the elements of which may contain all powers of ε up to the power of n. With the introduction of these matrices the difference of two succesive vectors of the series assumes the form

$$\underline{v}_m - \underline{v}_{m+1} = P^m\, (\underline{v}_o - (P\underline{v}_o + Q\underline{c})) \quad (7)$$

To show that $/\underline{v}_m - \underline{v}_{m+1}/$ converges to zero for $\varepsilon \rightarrow 0$ and $m \geq 1$, it is useful to decompose P (ε) with respect to powers of ε.

$$P\,(\varepsilon) = P_o + \varepsilon P_1 + \ldots + \varepsilon^n\, P_n \qquad (8)$$

Now we consider the case $\varepsilon = 0$. Then the set of statements corresponding to (3) becomes simple recursive which means that $\underline{v}_1$ is already independent of $\underline{v}_o$ and

$$P\,(o) = P_o = 0 \qquad (9)$$

This means that for small ε, P is proportional to ε and according to (7) $/\, \underline{v}_m - \underline{v}_{m+1}/$ becomes proportional to ε^m.

This proof of convergence can be extended to more general systems of equations. The first generalization is that the lines in the matrix A do not have to be in proper order but may be permutated . The effect is that (9) has to be replaced by

$$P_o^{\,i} = 0 \qquad (1 \leq i \leq n) \qquad (10)$$

where the exponent i is an integer function of the permutations of lines in A.

With (10) $/\underline{v}_m - \underline{v}_{m+1}/$ becomes proportional to ε^j, where j is the greatest integer $\leq m/i$.

The second generalization is that the equation system does not have to be a linear one. The whole argumentation holds as well when the elements of A and B as well as of all other vector expressions are not only linear functions of $\underline{v}$ but general polynomial functions of $\underline{v}$. Finally it can be extended to systems where the right hand side can be expanded in a multidimensional Taylor expansion around the exact solution.

So we may speak of a solution method for general non linear systems of equations where the right hand side can be written as a recursive part plus a relatively small back coupling part. The recursive part needs not be in proper order of sequence.

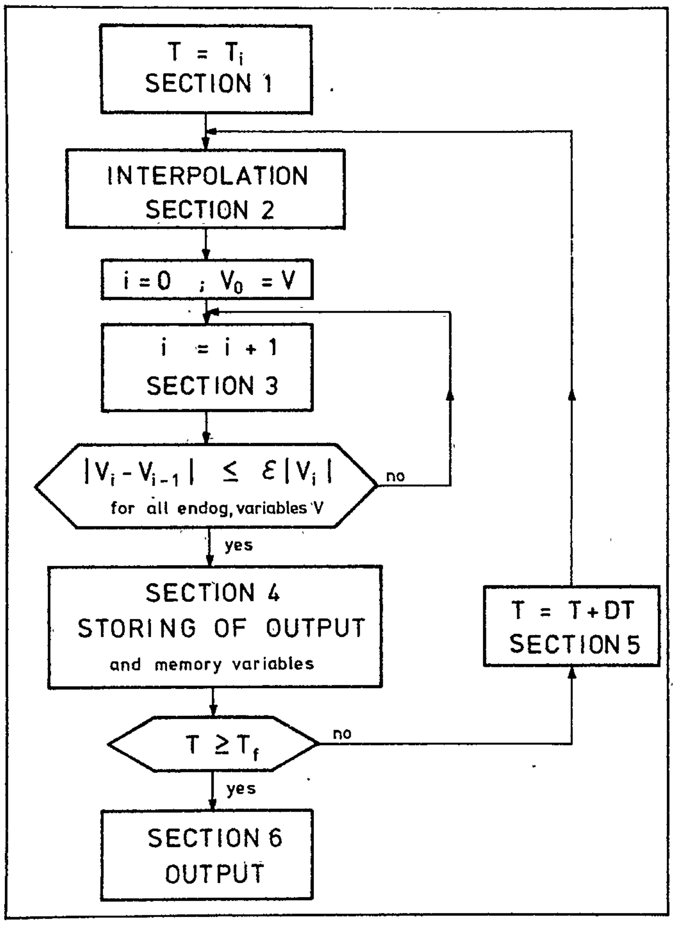
It is a safe method in the sense that, if the method converges, it converges to a solution of the problem. It has the advantage that the specific structure of the back coupling may remain hidden in the whole equation system. As can be seen from equation (7), it can also be an extremly fast method of solving large coupled equation systems. Furthermore it makes use of the fact that in dynamic simulation the solution for a given time step does usually not differ substantially from the solution at the preceding time step.

For the case that the back coupling is too large for the perturbation approach to converge another less efficient but completely general method for solving coupled equations can be used within the same simulation concept. This method minimizes the sum of squares $(\underline{v}_m - \underline{v}_{m+1})^2$ as a function of $\underline{v}_m$. A standard routine which does not need derivatives of the residuals is used for that purpose.

No we come to the question of how these solution methods for coupled equations can be combined with a scheme for solving differental and integral equations to form a complete dynamic simulation concept. This is illustrated in fig. 4.

Figure 4

Flowchart of the Simulation explaining the function of the different sections of the modules.



To be able to combine both solution methods it is necessary to divide each module into different sections so that similar operations can be synchronised in different modules. For the understanding of the simulation concept only 3 sections are important.

Section 1 is for all kinds of initializations. Section 3 is for block recursive or coupled equations (corresponding to the auxilary- and rate equations in the Dynamo concept) and section 5 is for the solution of differential equations and integrals e.g. by the simple rectangle method. (corresponding to the level equations in dynamo). The sections 2 and 4 have been introduced mainly to be able to reduce the computational effort by making section 3 smaller.

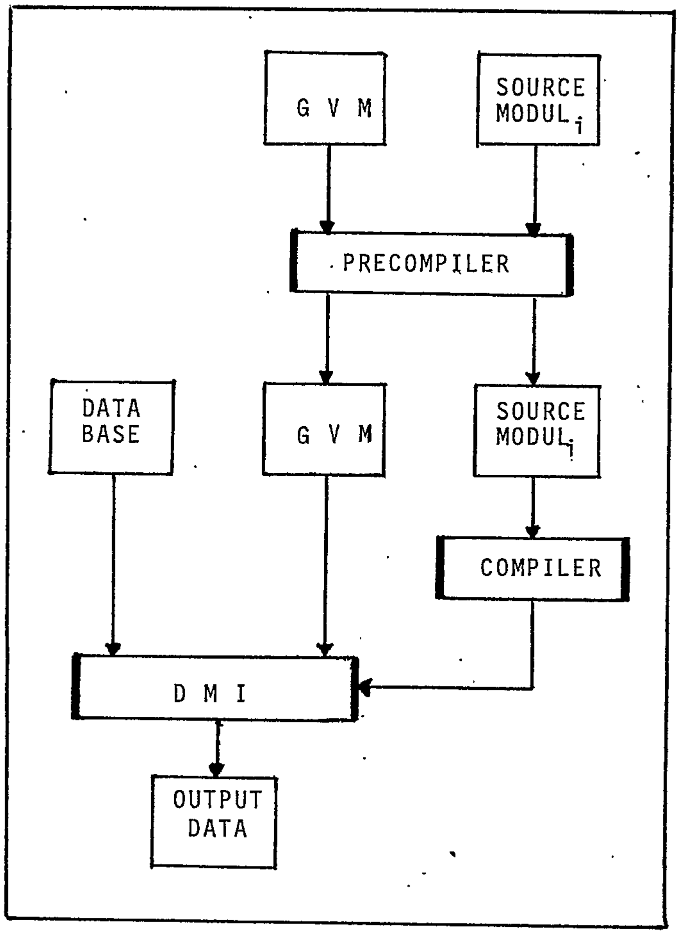
If not otherwise defined every global variable is represented by one single value which is its momentary value. If the simulation requires also previous values the corresponding variable can be declared as a "memory variable".

The time evolution of all variables is represented on a discrete equidistant time grid with a characteristic elementary step length. The step lengths ofthe individual modules can be different integer multiples of the elementary step length but these integers are fixed in time.
The same applies to the step length between the interpolations for the exogenous variables and to the output step length. whereas the output step length is equal for all output variables the interpolation step length of exogenous variables equals the step length of the modules to which they belong. When an exogenous variable belongs to several modules the minimum of their step lengths is taken.
The time grid on which the input data are defined can be either equidistant or not. It may also differ from item to item in the data base.

THE REALIZATION OF THE SIMULATION CONCEPT

The most important feature of the DMI is the Global Variable Map (GVM) which is created by a precompiler and used by the DMI at execution time of the model. This is illustrated in fig. 5.

Figure 5
General Flowchart of the DMI



The coincidence of the sequence of global variables in the GVM and the COMMON Blocks of all the modules allows a twofold access method to the values of all global variables.

First there is the access by name from the side of the modules used for all operations on global variables within the modules.

Then there is the access by the position index in the GVM. This access method is used for all operations on global variables within the DMI.

The actions of the DMD at execution time follow the scheme illustrated in table 1.

First the DMI selects certain subsets of global variables. These subsets are either

Table 1: Operations of the DMI at Execution Time

Subsets of Global Variables	Definition of the Subsets	Operations connected to the Subsets
exogenous variables	in global variable map	Interpolation of input time series
endogenous variables	in global variable map	Convergence checks after iterations of section 3
output variables	in global variable map	Buffering of output time series
memory variables 1) Derivatives 2) Time shifts 3) Delays	in section 1 of the modules	Storing of memory variables
Simultanous Variables for which the perturbation approach does not converge	after section 3 of the modules	Execution of minimization algorithm

predefined in the GVM like the endogenous and exogenous variables or they may appear at execution time like the memory variables and those variables of coupled equation systems for which the perturbation approach did not converge. For each subset, characteristic operations are performed at certain time points of the model run. The step length of these operations (exept for the output step length) are always related to the step lengths of the modules. For operations on endogenous variables the step lengths of these modules are taken in which these variables are endogenous. For an exogenous variable the minimum of the step lengths of all the modules is taken for which this variable is exogenous.

The main operations of the DMI at execution time are the following: At the beginning of a time step those variables whose values are needed in the currently activated moduls are provided with external data.

At the end of every iteration of section 3 the convergence of the endogenous variables of all active modules is checked. If the convergence cannot be achieved the minimization method can be automatically activated for those variables for which the perturbation approach did not converge.

At the end of section 4 the output variables are stored into the output buffer and the memory variables into the memory buffer.

At the end of the model run the DMI offers several options for displaying the output buffer. In the interactive version of the DMI the different output formats (numerical and graphical) can be chosen by dialogue.

SUMMARY

Summerizing one can say: The Jülich Data-Model Interface supports a convenient simulation of large dynamic systems with many endogenous and exogenous variables but with a relatively simple event structure. Its simulation concept is oriented to the one of Systems Dynamics. It gives less conceptual guidance compared to a simulation language like Dynamo, but on the other hand it gives the possibility of a modular model structure and a lot more flexibility which is only limited by the skill of the model builder to use the Fortran language.

ACKNOWLEDGEMENTS

I am grateful to Dipl. Phys. R. Heckler, U. Hermes and Dr. H.P. Schwefel for their help with the realization and testing of the DMI.

BIBLIOGRAPHY

/1/ Heckler, R. and Schwefel, H.P., Superposing Direct Search Methods for Parameter Optimization onto Dynamic Simulation Models, to be presented at the 1978 WSC, Miami, Florida.

/2/ Pugh III A.L., Dynamo Users Manual, The MIT Press, Cambridge Massachusetts.

/3/ Schmitz, K. and Schwefel, H.P. An Integrated Energy Simulation Model of the Federal Republic of Germany as a Decision Aid for Analysing and Planning the Energy System, 1977 WSC, Gaitherburg, Maryland.