Check for updates

The functional design of a generalized vehicle flight simulation program

by VILAS D. HENDERSON
Logicon, Inc.
Redondo Beach, California

INTRODUCTION

Aerospace missions have become exceedingly complex, requiring correspondingly complex vehicles in order to fulfill them and necessitating thorough analysis of the total mission/vehicle combination during the design, development, and testing phases. An important part of this analysis and synthesis concerns the behavior of the vehicle as it travels a flightpath toward the fulfillment of its mission. This behavior is so important that it actually may have considerable impact upon original mission objectives, over-all vehicle design, and various subsystem functions.

The dynamical process which describes a vehicle's flight has a solid foundation based upon the laws of the universe. A factor that characterizes the design of special-purpose vehicle flight simulation programs is the degree of realism and sophistication with which these physical laws are simulated. A prime factor that characterizes the design of a generalized flight simulation program is the extent to which this design can be made independent of the degree of realism and sophistication levels that may be used to model the physical laws. An elegant way to achieve a large measure of this independence is through functional software design.

The virtues of functional software design may be illustrated by contrasting a three-degree-of-freedom vehicle flight simulation and a six-degree-of-freedom simulation: there is no over-all functional difference between these simulations; it is only the degree of realism and the complexity of mathematical modeling that distinguishes them. Functional design permits either or both the three- or six-degree-of-freedom simulation capability to be incorporated into a generalized vehicle flight simulation program with little sacrifice or compromise in either simulation from the viewpoint of efficiency and flexibility.

It should be stated at the outset that specific simulation languages are not given attention in this paper. Functional design is considered all important; the simulation languages are considered a means of implementing a design, but in no way should they influence it. For this reason the design presented here is conceptually independent of simulation language, operating system, and computer design.

The genesis of the program described here was an Aerospace Corporation effort, undertaken in 1963, to develop a generalized digital flight simulation program for use principally on Titan III missions. The initial development effort, in which the author participated, was in assembly language on the IBM 7094. Subsequently this program has been developed by Logicon, Inc. for the Univac 1107/1108, using a mix of assembly and FORTRAN languages. Preparation of an all-FORTRAN-IV version of the program for the GE 625/635 and CDC 3600 computers is nearing completion.

The basic program has been extended into a system called the Modularized Simulation System. At least a hundred different applications have already been made of this system; these include studies in such areas as

Trajectory design and targeting
Open- and closed-loop guidance simulations
Three- and six-degree-of-freedom vehicle simulations
Error analysis
Radar tracking
Environmental effects
Flight mechanics and performance analysis
Sensor simulations, and
Digital autopilots.

Major mission applications have included Titan III, several satellites, a large-payload test vehicle, and advanced ballistic missile systems.

MATHEMATICAL PROBLEM DEFINITION

The description of a vehicle's flight involves the definition of a complex dynamical system. Mathematically this system can be concisely described by a vector differential equation of the form

$$\dot{x}(t) = F[x(t), p(t), c(t), t] \qquad (1)$$

where

$x(t)$ = the system kinematic state vector
$p(t)$ = the system vector parameter function
$c(t)$ = the system vector control function
t = the independent variable, time.

The kinematic state vector $x(t)$ is assumed to be of dimension twelve, consisting of three-dimensional components in velocity, position, attitude rate, and attitude. The vector parameter function $p(t)$ includes parameters such as vehicle weight, engine specific impulse, vehicle cross-sectional area, gravitational constants, and aerodynamic drag. A good many of the parameters may be time varying and perhaps even a solution to a differential equation; many others may be constant for a particular mission or vehicle. It should be noted that for different mission/vehicle combinations, $p(t)$ may vary in dimension, functional form, or both. As an example, an Apollo mission involves physical and gravitational parameters for both the earth and the moon, whereas a near-earth ballistic missile mission need only involve the physical and gravitational parameters of the earth.

The vector control function $c(t)$ is also dependent upon the particular mission/vehicle combination. The number and functional form of the control variables may vary from vehicle to vehicle depending upon the energy sources available and the number of system constraints that must be satisfied. Thus an accurately guided vehicle may utilize pitch, yaw, and roll steering policies to achieve a desired thrust vector direction. In contrast, a vehicle subject to no accuracy requirement may only utilize thrust termination time as a control variable; a free-fall mission may simply require a null control function.

The digital simulation of the dynamical system represented by (1) involves obtaining a solution of the form

$$x(t) = \int_{t_o}^{t} F(\tau)\, d\tau + x(t_o) \qquad (2)$$

For a given set of initial conditions and the functional form of $p(t)$ and $c(t)$, the problem of obtaining a solution to (2) becomes immensely simplified. Therefore, a simulation program designed to give a particular solution or a restricted class of solutions is not very difficult to design. It is a different matter, however, to design a generalized vehicle flight simulation program for which very few assumptions are made about the explicit form of $p(t)$ and $c(t)$. The functional design principles presented in this paper preserve the generality of (1) and (2) to a remarkably high degree.

MISSION PROFILE SPECIFICATION

The term *trajectory* is used to denote the time history of a vehicle's dynamic state. This state is only completely known when $x(t)$, $p(t)$, and $c(t)$ are all known.

A *trajectory phase* denotes a part of the total trajectory and as such it is initiated by some event and terminated by some later event.

An *event* is a discrete point in time along a trajectory. It represents the terminating point of the preceding trajectory phase and the initiation point of the ensuing trajectory phase. The occurrence of an event will usually imply a discontinuity in $p(t)$, in $c(t)$, or in both.

An *event criterion* is a policy for the determination of an event. Multiple criteria for a single event are admissible.

Mission profile is used in connection with a family of trajectories, all of which are characterized by the same sequence of events and trajectory phases.

Each trajectory consists of at least starting and terminal events which define the boundary points for the intervening trajectory phases. The trivial trajectory consists of a trajectory phase of zero time duration and concurrent starting and terminal events.

Recall that the solution to the dynamical system described by (2) involves a set of differential equations. If $p(t)$ and $c(t)$ are subject to change during the course of a trajectory, and if these changes are identified with the occurrence of an event, then a trajectory can be thought of as the solution of a set of piecewise continuous differential equations which require initialization at each event. It then follows that the digital simulation of this dynamical system is, in effect, continuous over each trajectory phase. This establishes the basis for the criterion that all initialization can be performed at events, and therefore an event essentially defines the nature of the ensuing trajectory phase.

Each event is given a t^-, t^0, and t^+ interpretation: the t^- interpretation pertains to the end of the preceding trajectory phase; the t^+ interpretation to the initiation of the subsequent trajectory phase; and the t^0 interpretation to initialization. It is noteworthy that for any given mission profile, the first event has no t^- associated with it while the last event encountered has no t^0 or t^+ associated with it.

Event Classifications

All events are classifiable in terms of whether they can be ordered absolutely with respect to each other. If all the possible events for a given mission profile are defined, then there is always a starting event and a desired terminating event. In the trajectory interval between the start and end of the mission profile, there will usually exist a sequence of events, most of which may be expected to occur in some predetermined order. All those events which must occur in absolute order with respect to each other are called *ordered* events. Any

event that cannot be given an absolute order is classified as a *roving* event.

There are instances when a designated event may be superseded by another event. The superseded event is classified as a *secondary* event; all others are called *primary* events. Hence there are four classes of events which may be specified:

Ordered primary
Ordered secondary
Roving primary
Roving secondary.

Event Sequences

An *ordered primary* event must occur during the course of a complete and successful simulation, and its occurrence must take place in absolute order with respect to all other ordered primary events. An *ordered secondary* event may or may not occur, but if it does occur, the occurrence must be in the interval bounded by two consecutive ordered primary events. To illustrate, let P_1 and P_2 designate two consecutive ordered primary events, and let S_1 and S_2 designate two consecutive ordered secondary events which must occur in the interval separated by P_1 and P_2 if they are to occur at all. Then the following three sequences of events are possible, given appropriate event criteria and the specified input shown as Sequence 1:

1. P_1, S_1, S_2, P_2
2. P_1, S_1, P_2
3. P_1, P_2

It should be noted that event criteria must be reasonable in order to achieve a given sequence. For example, if Sequence 2 is to occur, then the criterion for the occurrence of S_1 must be reasonable relative to the criterion for P_2; if it is not, P_2 will supersede it.

A *roving primary* event can occur at any time in an event sequence after the last ordered primary event it follows. If such an event, say P_1', is inserted into the previous sequence between P_1 and S_1, then the following event sequences are possible, again given appropriate event criteria:

1. P_1, P_1', S_1, S_2, P_2
2. $P_1, S_1, P_1', S_2\ P_2$
3. P_1, S_1, S_2, P_1', P_2
4. P_1, S_1, S_2, P_2, P_1'
5. P_1, P_1', S_1, P_2
6. P_1, S_1, P_1', P_2
7. P_1, S_1, P_2, P_1'
8. P_1, P_1', P_2
9. P_1, P_2, P_1'

Since an ordered primary supersedes all roving primaries, Sequences 4, 7 and 9 will not include event P_1' whenever P_2 is the last event in the entire input sequence. Thus it is possible that a roving primary event may not occur at all in the course of a simulation.

A *roving secondary* event must be specified when there are two secondary events that cannot be absolutely ordered with respect to each other. Let S_1' be a secondary event whose order of occurrence is unknown with respect to S_1 and S_2, and let P_1 and P_2 designate ordered primary events. The possible event sequences, assuming appropriate event criteria, are:

1. P_1, S_1', S_1, S_2, P_2
2. P_1, S_1', S_1, P_2
3. P_1, S_1', P_2
4. P_1, S_1, S_1', S_2, P_2
5. P_1, S_1, S_2, S_1', P_2
6. P_1, S_1, S_1', P_2
7. P_1, S_1, S_2, P_2
8. P_1, S_1, P_2
9. P_1, P_2

From this discussion, it is readily apparent that there are frequent occasions when a number of event criteria must be monitored simultaneously if secondary and roving primary events appear in the sequence. Use of multiple criteria also imposes this requirement. To illustrate, consider the input sequence shown in Figure

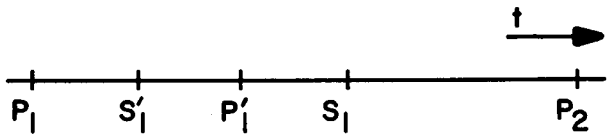

Figure 1 — Typical input sequence

1. All the criteria associated with events S_1', P_1', S_1, and P_2 are monitored as soon as P_1 occurs. If event P_2 is encountered, then all criteria associated with S_1' and S_1 are immediately dropped from the sequence, as well as the criteria associated with P_2. If P_1' is not encountered by the time P_2 occurs, then its event criteria will be further monitored, providing P_2 is not the last event in the total input sequence.

Each event criterion produces a time-to-go-to-an-event parameter. The occurrence of an event is triggered when one or more of the time-to-go parameters come within a specified tolerance of zero.

PROGRAM STRUCTURE AND ORGANIZATION

Flight Dynamics Elements

A vehicle's flight involves, of course, vehicle dynamics and an external environment. Superimposed upon these are the navigation, guidance, and control functions that together provide the means by which a vehicle is directed to its intended terminal point. Figure 2 illustrates the functional relationships between these

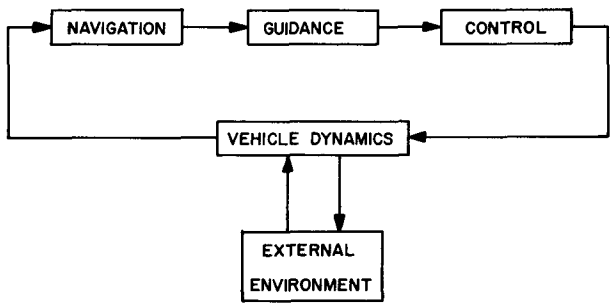

Figure 2 — Functional relationships in flight dynamics

basic elements. Vehicle dynamics can further be divided into kinematics and kinetics. Kinematics can then be subdivided into translational and rotational elements; and kinetics can be divided into aerodynamics, propulsion, and structure.

Navigation, guidance, and control functions are more effectively dealt with in a simulation if the software- and hardware-oriented functions are separated. Navigation involves sensors (hardware) and data processing (software); guidance is a software function; and control is mostly a hardware function. The sensor, guidance and data processing, and control elements are therefore chosen in preference to the navigation, guidance, and control elements.

The division of flight dynamics into the set of functional elements just described has been found to be optimum for a wide class of simulation problems. The functional interaction of these elements is shown in Figure 3.

Peripheral Elements

A generalized vehicle flight simulation program must consist of much more than the flight dynamics elements just described if it is to be of any utility. Some major additional factors that must be considered are the means by which a numerical solution to (2) is to be implemented, the sequencing and controlling requirements, and provisions for input and output. The approach taken here is that there exists a set of peripheral functional elements which complement the nine mathematically oriented elements; collectively they form the basis for the structure and organization of the total program. Descriptions of the peripheral elements follow.

The division of a trajectory into trajectory phases suggests a hierarchy of three executive elements for the control of flight profile sequencing:

- A *trajectory executive* element to control the simulation on a total trajectory basis;
- A *trajectory phase executive* element to control the simulation on a trajectory phase basis;
- A *cycling executive* element to control the simulation on an integration cycle basis.

The cycling executive element forms the hub from which the major simulation computations are performed. Three subservient elements complete the executive network as shown in Figure 4:

- A *data processing and guidance executive* element to control the data processing and guidance flight dynamic element and certain special functional

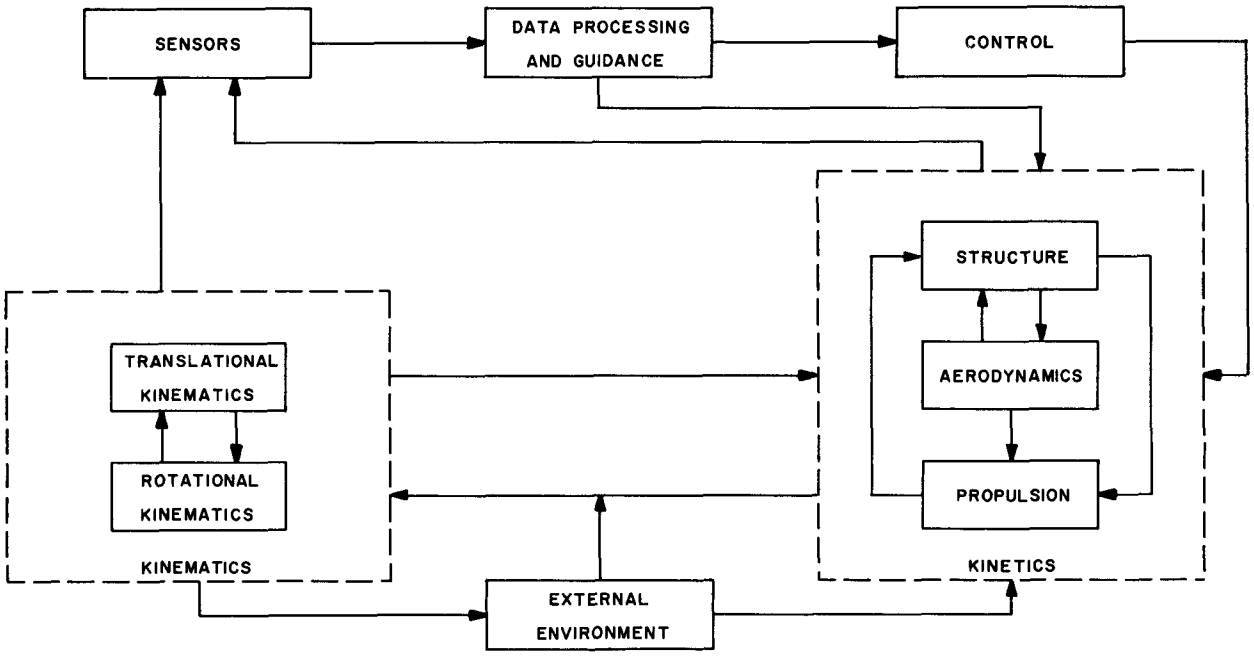

Figure 3 — Principal elements of flight dynamics

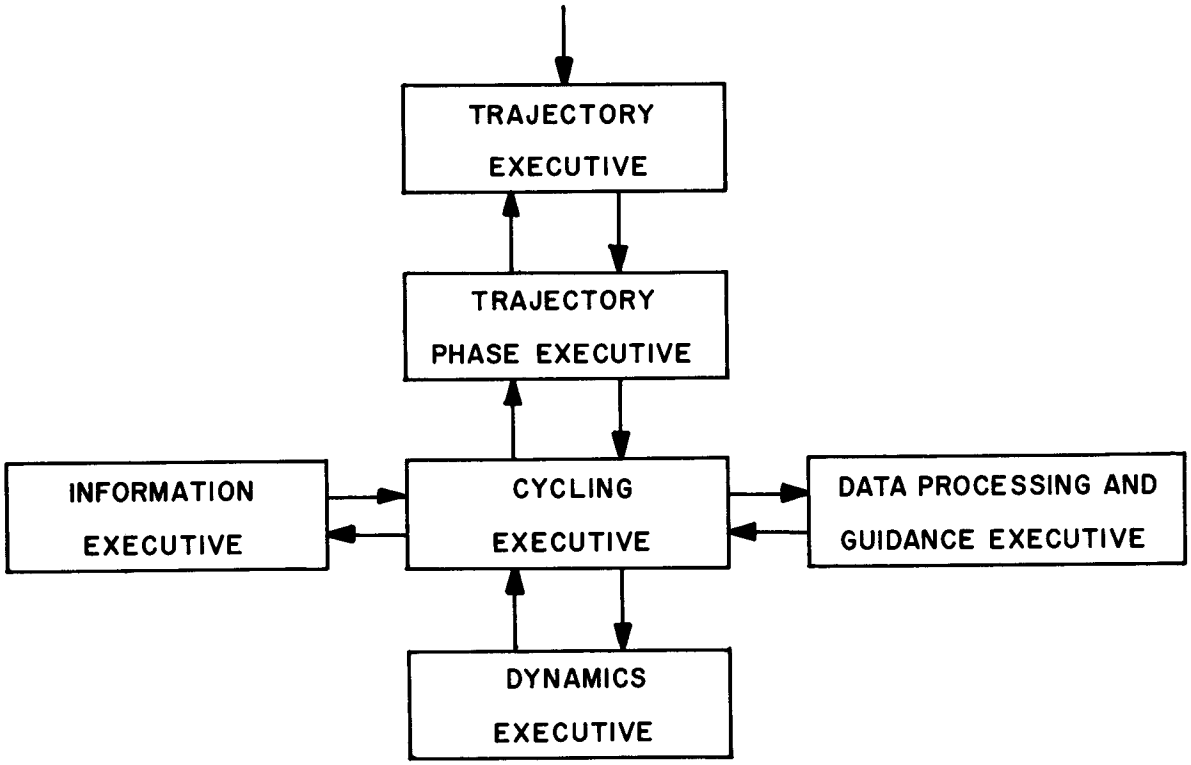

Figure 4 — Executive elements for vehicle simulation

elements required for monitoring event criteria and for providing generalized open-loop steering techniques;

A *dynamics executive* element to control the integration process and the eight elements of flight dynamics other than the one given above;

An *information executive* element to perform the what, how, and when functions for all information output processing.

Four more peripheral functional elements complete the basic set:

A *master program executive* element to provide the control for trajectory computation or other simulation functions and also to control input and output processing functions;

A *service* element to provide service features common to all elements;

An *input processing* element to enter and process all input;

An *output processing* element to edit, process, and output all special data formats such as that required for plotting.

At this point we have arrived at a set of functional elements which comprise the simulation program. Each of these elements appears in the simulation program as a software unit called a *module.*

Module/Model Relationships

Although the functional relationship between modules is inherently invariant, the character of the function that must be performed by a given module may vary considerably. This leads to the module/model concept, which can be expressed mathematically as follows: Let S represent the total set of functional elements which will comprise the vehicle flight simulation program. Thus S can functionally be written as

$$S = S(M_1, M_2, \ldots, M_n) \tag{3}$$

where M_i designates the i*th* functional element and $i = 1, \ldots, n$. Similarly, a given module M_i can be described functionally in the form

$$M_i = M_i)m_{i1}, m_{i2}, \ldots, m_{i\,l_i}) \tag{4}$$

where m_{ik} denotes the k*th* particular description of the function assigned to module M_i and $k = 1, \ldots, l_i$. The M_{ik} appear in the simulation program as software units called *models* and are always affiliated with a module.

A major goal was to design a generalized program that would permit the execution of a very specific program. This requirement has been satisfied by providing a mechanism by which the m_{ik} can be selected through input at run time. (For example, among the m_{ik} for module M_i is the unity transfer model which causes M_i to perform a null function.) The nature of any

particular simulation is thus dependent upon a model selection process that is placed in the hands of the program user. This feature also provides both flexibility and computational efficiency.

Module Characteristics

This program design discussion has now reached the point where the simulation program has been structured and organized about a set of functional elements called modules. A module M_i has well defined inputs and outputs and can be considered somewhat analogous to a system transfer function. Module M_i can be viewed in terms of Figure 5, in which

z_i = output vector of module M_i
$\bar{z}_i$ = output vectors from the M_j modules, $j \neq i$
p_i = externally supplied parameter input vector for module M_i
$\bar{p}_j$ = parameter input vector to the M_j modules, $j \neq i$.

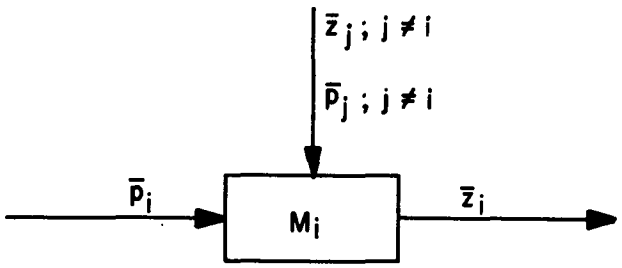

Figure 5 — Module input/output

To permit each module to be as autonomous as possible, it is necessary to very carefully define module interfaces. These interfaces are grossly known by the nature of the function assigned a module but are only precisely known when p_i and z_i are completely specified. Such detail is withheld from this paper but can be obtained from the formal program documentation.[1,2]

A physical structure for module M_i is now easily deduced which preserves the entity of each module in the simulation program. The segments which comprise this structure are the

Control segment
Input segment for p_i
Model segment
Output segment for z_i
Subroutine segment.

Additionally, a dictionary segment may be required depending upon the mechanization employed.

The control segment provides each module with a model selection capability. The input segment contains all the single-piece data components of p_i. In the case of variable-length-table or array input data, only a parameter which locates the data appears in the input segment. The model segment contains concise representations of the subfunctions assigned to a module, i.e., each model of any complexity primarily controls a set of subroutines appearing in the subroutine segment of that module or in the service module. The output segment contains all of the components of z_i; these components may be required for use by the module in which they reside, they may be necessary inputs to another module, or they may be required for information purposes. Each module subroutine segment contains a collection of building block material that may be used to construct a model unique to that module.

One of the most important characteristics of each module is that it is designed to initialize itself. A mechanism is included in the module control segment for selecting initialization models versus mainstream computational models. Recall that initialization is triggered only at an event when $t = t^0$. At this time each module reinitializes the components of p_i, which then remain constant for the subsequent trajectory phase. Certain computations may then be required to reinitialize the components of z_i also.

The models of module M_i are constrained by the requirement that they cannot alter the components of p_j and z_j for $j \neq i$. As shown previously, these vectors are available for model computations, in any module, but the values of the components of p_j and z_j are held sacred to the module in which they reside. A feature of functional design is that it significantly reduces the number of components of p_j and z_j which need to be referenced in module M_i

It is of interest to compare the vectors p_i and z_i for $i = 1, \ldots, n$ with the vectors x, p, and c which were defined by Equation (1): the p_i vectors are collectively a subset of p, whereas the z_i vectors collectively contain all the components of x, x and c and those components of p not represented by the p_i vectors.

Module Interfacing

Figure 6 shows the total interaction of modules in the program.

Input Processing Techniques

The following brief summary is indicative of the input philosophy used in this program. The design details concerning the input processing techniques which complement the other program design features are more thoroughly covered elsewhere.[2]

The problem of processing the collection of externally supplied p_i vectors ($i = 1, \ldots, n$) is especially acute for a generalized flight simulation program. First, there is ordinarily a large amount of data necessary to identify a vehicle, its environment, and the mission

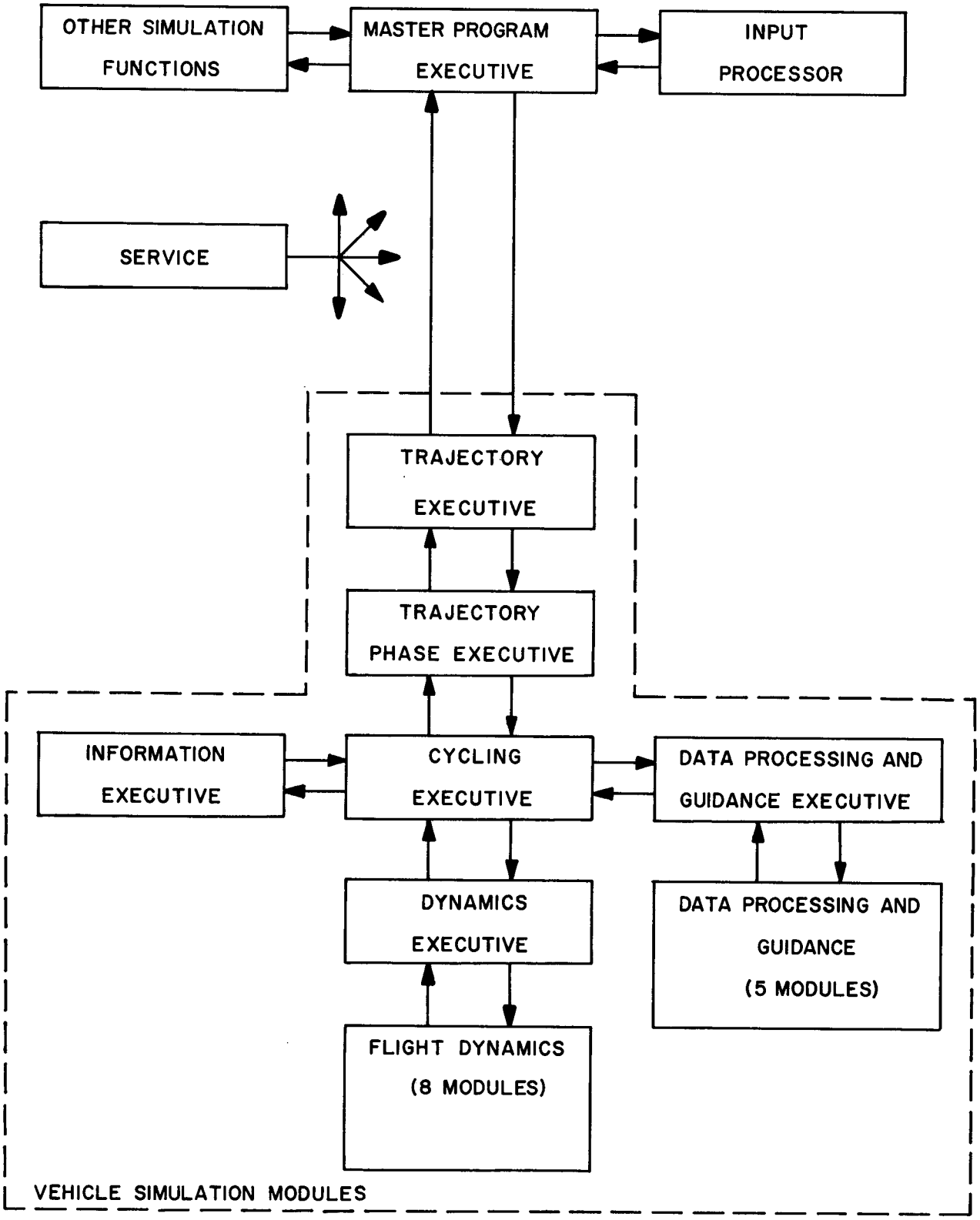

Figure 6 — Module interaction

profile. Second, much of the data is tabular in form, creating a problem of efficient storage allocation. Third, a good number of the input parameters take on a sequence of discrete values over the course of the trajectory; consequently there exists a dynamic storage allocation problem.

Previous discussions have dwelt heavily upon events and modules. All initialization occurs at events and all

modules initialize themselves. It is quite natural, therefore, that the components of the p_i vector be identified by the event at which they are first used and by the name of the module (M_i) for which they are to be input.

At execution time all the input data (p_i vectors) are processed and placed into a variable-size data region. The inputs are compressed and ordered according to the event and module which identify them. During the vehicle flight simulation, module M_i extracts the components of p_i that it requires at the event occurring. The data are entered into the input segment of the module and remain therein until replaced at some subsequent event. An exception is made for tabular data: this type is never extracted from the variable data region; only the location is determined and provided to the appropriate module.

HYPOTHETICAL VEHICLE SIMULATION EXAMPLE

The design principles discussed in this paper may be illustrated by a hypothetical vehicle simulation example. Assume a four-stage vehicle carrying a payload that is to be placed into some final orbit after being injected into an earth-parking orbit. The first three stages burn consecutively and provide enough energy to place the vehicle into the parking orbit. The final stage is fired twice to effect the transfer to the desired final orbit.

The vehicle is guided by three techniques:

1. Zero-lift steering (1st and 2nd stages)
2. Explicit-velocity-required steering (3rd stage)
3. Constant-attitude steering (4th stage)

Three models can be developed which efficiently implement these steering requirements for the hypothetical mission. These models are written for a data processing and guidance module controlled from the data processing and guidance executive as shown in Figure 6, and are called Models 1, 2, and 3.

Assume further that the mission profile can be specified via a sequence of ordered primary events as follows:

E_1 — liftoff
E_2 — end vertical rise, start zero-lift steering
E_3 — end 1st stage, start 2nd stage
E_4 — end 2nd stage, start 3rd stage
E_5 — end 3rd stage, start coast
E_6 — end coast, start 1st burn, 4th stage
E_7 — end 1st burn, start 2nd coast
E_8 — end 2nd coast, start final burn
E_9 — end final burn, enter final orbit
E_{10} — end flight (on basis of time, number of orbits, etc.).

Although only ordered primary events are shown in this input sequence, secondary and roving primary events are conceivable. For example, an event defining a certain orbital altitude might be specified between E_7 and E_8. If it were defined as a secondary, then the designated altitude would have to be reached before occurrence of E_8 or it would not occur at all. If it were defined as a roving primary event at the same point in the sequence, then it could occur at any subsequent time up to the occurence of E_{10}.

Returning to the event input sequence listed above: at E_1, Model 3 is specified for steering with appropriate parameters. At E_2 Model 1 is called, replacing Model 3. At E_4, Model 2 is called, replacing Model 1; and at E_5, Model 3 is again called with appropriate parameter values. Thereafter Model 3 is used, possibly with new initializing parameters at the subsequent events.

If we relate the inputs and outputs required for these models in terms of x, p, c, the result is that there are no components of x produced by models in the data processing and guidance module; p may have components (such as vehicle mass, guidance constants, and vehicle attitude constants) produced in this module; and c will have its major components (steering commands and engine discretes) generated there.

Other modules can be looked at similarly and models may be chosen, if available, for the particular mission functions that are required. Again, for emphasis, models are specific representations of a general function, and in this case the general function of vehicle data processing and guidance is broken into three specific steering models.

REFERENCES

1 V D HENDERSON
The logicon modularized simulation system Part I: A general description
Logicon Inc
Redondo Beach California May 1966

2 J TATUM M TURNER
The logicon modularized simulation system Part II: The mechanics of input
Logicon Inc
Redondo Beach California May 1966