ACM Technical Report ACM-TR-108

CR 151606

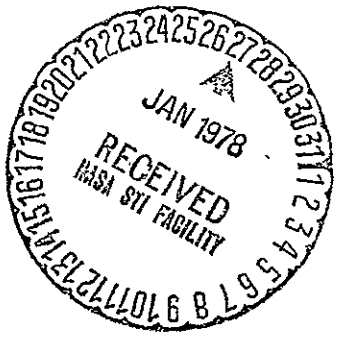
(NASA-CR-151606) A COMPARATIVE STUDY OF THE UNIFIED SYSTEM FOR ORBIT COMPUTATION AND THE FLIGHT DESIGN SYSTEM (Analytical and Computational Mathematics, Inc.) 53 p HC A04/MF A01

N78-15157

Unclas

CSCL 22A G3/16 01820

A COMPARATIVE STUDY OF THE UNIFIED SYSTEM FOR ORBIT COMPUTATION AND THE FLIGHT DESIGN SYSTEM



ANALYTICAL AND COMPUTATIONAL MATHEMATICS, INC.

A COMPARATIVE STUDY OF THE UNIFIED SYSTEM FOR ORBIT COMPUTATION AND THE FLIGHT DESIGN SYSTEM

WERNER MAAG

ANALYTICAL AND COMPUTATIONAL MATHEMATICS, INC.

1275 SPACE PARK DRIVE, SUITE 114,

HOUSTON, TEXAS 77058

NOVEMBER 1977

This report was prepared for NASA/Johnson Space Center under Contract NAS9-15171.

CONTENTS

A COMPARATIVE STUDY OF THE UNIFIED SYSTEM FOR ORBIT COMPUTATION AND THE FLIGHT DESIGN SYSTEM

by

Werner Maag

1.0 INTRODUCTION

The high flight rate anticipated for the Shuttle Transportation System (STS) era beginning the 1980's is causing the review of the mission design and analysis computer software structures. The rather individual case-by-case planning procedures and software products which were used for earlier missions are no longer adequate when considering the projected budgets for the STS operations. Standardized planning tools are absolutely necessary in the environment of shuttle flight design since many tasks are repetitive. Also, profit must be taken from the experience gained with earlier missions and from the advances made in the areas of flight design techniques and computer hardware and software standards.

In this report, two recent software systems will be described and compared. One is the Flight Design System (FDS), currently being developed at MPAD of NASA/JSC[†]. The other is the Unified System for Orbit Computation (USOC), operational at MAD of ESA/ESOC[††].

The FDS software consists of a set of functional and utility processors, components of the data base structure and

---

[†]Mission Planning and Analysis Division of NASA/Johnson Space Center, Houston, Texas.

[††]Mission Analysis Division of the European Space Agency/ European Space Operations Centre, Darmstadt, Germany.

the executive logic. It is designed as an efficient production tool for flight design and analysis tasks, to be accomplished for the support of operational shuttle missions and orbital flight tests. It also comprises an interface with a system for automated generation of flight planning documents.

USOC has been used for the planning, analysis and design of spacecraft missions, both earth orbital and interplanetary. It makes use of an automated approach to the *generation* of problem specific application programs from a library of functional modules. A module selector program interfaces with the user and selects the necessary modules for a particular application. The USOC design expressly facilitates quick modifcations by the analyst.

The FDS is a *production* oriented software product, mainly designed to satisfy flight design needs of the class of shuttle missions which are considered as standard missions or deviations of such. It is not intended to satisfy the total analysis needs of MPAD . One of the goals of this study is to evaluate the approach of USOC for the planning and design of *unique* shuttle flight phases. It will be demonstrated in this report that in fact, a software system having USOC-like structure would be appropriate to *compliment* the FDS . The structural details, however, would need to be redesigned to meet the specific needs of MPAD .

The comparisons that are made in this study are based on published documentation of the general requirements for the Flight Planning System (reference 1), general FDS requirements (reference 2), documentation of FDS-1 (reference 3), and lecture presentation material on FDS-X (reference 4). The author relies also on extensive interviews with MPAD personnel and acknowledges their assistance in this study. FDS-Y could not be included in this study since no detailed documents have been published. On the side of USOC , the comparisons are based on its documentation (references 5 and 6) and on personal

information of the author on the current activities of the further development of the USOC

2.0 COMPARISON OF SCOPE AND PURPOSE

It is immediately clear that the structure of a flight planning support program system depends very much on the emphasis given to its various application functions. Three main areas will be distinguished:

a) Mission analysis and feasibility studies

b) Mission design and planning

c) Operations planning

For USOC , the priorities have been defined in descending order from a) through b) to c). For FDS, the priorities are set in the opposite order.

2.1 Mission Analysis and Feasibility Studies

At the European Space Agency, feasibility studies are typically initiated by a request to the Mission Analysis Division, which owns and maintains USOC and is partially responsible for such types of studies. USOC is therefore required to be able to give quick answers to questions arising from new types of projects. These may comprise, for example, payload requirements which were previously not known because they result from a new idea on a new experiment. It is clear that usually some ad-hoc software modifications are necessary to give the answer.

The most fundamental requirement for USOC is, as a consequence, adaptability. That is the property of allowing easy modification. Practically, of course, the master version of USOC will not be changed. The modifications are done on copies of respective modules. Application of USOC for such tasks is based on the fact that often only a very few modules have to be changed, while the framework can still be used.

It is also desirable that USOC be able to support mission analysis development tasks. Examples are the development of

of new methods in the field of orbital mechanics and efficient algorithms for the investigation of payload requirements. Eventually, new methods are added to the system's standard capabilities. The requirement that USOC be an analysis tool results, too, in the basic requirement of adaptability.

In contrast to USOC , the FDS does *not* have a high priority requirement to be a tool for studies concerning the feasibility of new types of projects. FDS might, in certain cases be useful for such tasks but this would be a welcome side effect. Definitely, it is not a research tool, since it is not designed to allow ad-hoc program modifications. First, the code of the processors is not accessible to the normal user and the executive system would not allow duplication of processors. Secondly, in a production environment, user modifications would be very dangerous to the reliability of the produced data and to the system as a whole. However, modifications can easily be made in the FDS concept, but only at the maintenance level, not by the user himself. Flexibility must be implemented by foreseeing a sufficient number of user accessible options and parameters within the processors. Finding the optimal degree of freedom is certainly the crucial task of processor design. It is mandatory that the experience gained with applications of FDS be taken into account when designing new processors.

2.2 Mission Design and Planning

Mission planning can be defined as the task of finding the detailed specifications of an approved mission. This includes orbital and payload aspects as well as questions arising from the operations of the mission. This area is common to USOC and FDS . The set of functional capabilities, however, is limited in USOC compared to FDS . For example, any type of calculations concerning manned flights, consumables as well as launch and re-entry phases are not included in USOC . For details see Section 3.2.

Mission Planning includes, for example, parametric studies for the optimal satisfaction of payload requirements, the definition of the orbital parameters, and calculations concerning launch opportunities. The accuracy requirements of output data range from moderate to fairly precise. USOC offers generally different levels of accuracy which can be selected according to the actual precision needed and the tolerable computational expense.

Within FDS, System X is designed generally to be ready to accomplish the mission planning tasks. However, in certain cases, e.g. for missions involving deployment or retrieval of payloads, it may be necessary to apply methods which are beyond the capabilities of System X . The corresponding planning mechanism appears to be missing in the documents on Flight Planning (references 1 and 2). System Y will probably satisfy the accuracy requirements but it is only foreseen to use it at Level D planning, which is rather late in the planning process. It might be possible that System Y could be used at an earlier stage and that it have the necessary flexibility. An alternative approach is to develop a new system that has the necessary flexibility, plus a wide range of accuracy capabilities.

2.3 Operations Planning

Finally, there is the level of operations design. This task consists of the generation of detailed flight profiles which includes the flight trajectory, maneuvering and attitude control procedures, detailed observability and tracking considerations and much more. Basically, little analysis is performed since the necessary parameters have been determined at the mission planning level. Additional parameters describing the hardware configuration in detail enter the calculations. However, all mission planning effort has to result in a plan which uses a large, but a finite number of hardware resources,

well defined activities and flight phases. These elements will not change very much from flight to flight and the type of parameters and the number of options is complicated, but manageable. Therefore, this level of flight design is the one which can be standardized best, at least for the shuttle flight design requirements.

USOC does basically not deal with operations design (since this task is not assigned to Mission Analysis Division). But it has happened several times that certain functions of operations design were requested from USOC in new or unique situations where the effort to modify standardized software would have been too large. This capability of USOC was not a requirement, but is a welcome side product.

Many of the operations design functions can be performed independently in relatively small segments. This is a strong argument for the concept of independent processors that is used in FDS . As mentioned before, the nature of this task allows its breakdown into a set of predefined planning activities, in the case of standard missions. There will remain a residual of special cases which need software modifications. For operations design tasks, the current USOC concept to generate *one* complete problem-specific application program would not be efficient. The problem of merging all the required functions into a large overlay structure would probably be very difficult. The solution would be to allow the generation of processor-like functional units. This, in turn, could introduce the flexibility in the sense of adaptability to unforeseen problems on the operations design level.

2.4 User Categories

The FDS is explicitly required to support the utilization of various skill levels of its users. There will actually be a mix of skill levels which range from the technician to the analyst, with the emphasis given to the technician level. An

important consequence is that the system design and components of the system must ensure a maximum degree of "foolproofness" in order to guarantee the reliability of data products. It can generally not be assumed that the user has sufficient knowledge of the methods he applies and that he will always make intelligent use of the system. Software development and maintenance must assure the quality of produced data. This is in practice a very hard requirement which can be satisfied only by introducing rigorous checking procedures at the level of the executive system and of the processors. The implied overhead in system complexity and the manpower effort for the realization of maximum foolproofness is only worthwhile if the system can be stabilized to a high degree with respect to the frequency and amount of modifications. This in turn, requires that the spectrum of capabilities be large and the branching into selectable cases be detailed. As a consequence, the system becomes rather difficult to modify in order to deal with cases which are not foreseen.

USOC, in contrast to FDS, is not specifically designed to support repetitive tasks. The user category that is addressed is assumed to be the analysts. It is generally assumed that the user is aware of possible limitations in the methods and programs that are available in the system and that he understands the concept of the system. It is attempted to assure foolproofness on the level of modules. However, there are no checks for validity of the combinations of modules or compatibility of modules with particular data. Since the system is designed to support modifications of individual modules, validity checks are necessarily limited and sometimes even not desirable, since the user might like to make experiments near some performance limits. Also, the requirement of of simplicity in the sense of visibility of the structure of the system does not allow too much checking. The general guideline is that the modules and programs generated by the preprocessor (if intelligent combinations of input are given)

are safe. Pitfalls are avoided through clear documentation. The rest is left to the user. The fact that individual modules can be modified (if actually necessary) helps considerably to keep modules simple since the degree of branching into various preprogrammed cases may be kept small. This, in turn, improves greatly the overall reliability of the system.

3.0 SCOPE OF APPLICATIONS

The comparison of application capabilities of FDS and USOC is complicated by various facts. The main obstacles are the different method of development applied for the two systems, together with the different status of maturity of their development. Also, the basically different software structure does not allow a one-to-one comparison.

The design of FDS has been based on experience gained with several ancestors and the requirements can be expressed on the background of capabilities of various existing programs. The application functions of the system are concentrated on the shuttle operations. It is possible and, in view of the selected system organization, it is desirable to define the scope of applications and the detailed capabilities more or less precisely in advance.

The full concept of USOC, in contrast, has no predecessors, at least in the field of space research. (The idea of program generation has been used in the area of business computer applications.) The software must be able to deal with all types of orbits of space vehicles in the solar system. The structure of the system is explicitly designed to allow the easy extension of the range of capabilities offered as standard options by the preprocessor and the library. Extensions are therefore usually performed only as required for the actual near future. The potential scope of applications is an open field.

We can mention before going to a more detailed comparison that USOC and FDS give different emphasis to the various functions of flight design. USOC puts more emphasis on orbit prediction sophistication while offering a limited selection of on-orbit operations analysis functions. The generation of detailed operations planning products is left to another software system. Launch and re-entry simulations are not required,

since these are outside the scope of ESOC operations. FDS assumes the exhaustive end-to-end operations analysis and the generation of corresponding timelines as the main task. Complex orbital problems are probably beyond the scope of applications.

The following more detailed comparative lists of applications are founded on that information (which is currently available to the author) which contains sufficient technical substantiation to allow a comparison. The functional requirements to the FDS System X are compared with the implemented functions of USOC, followed by a brief indication of planned extensions of USOC . The structuring of the lists is derived from reference 2, p.28. FDS is described mainly in terms of functional requirements of processors, as listed in reference 4, while taking into account documentation in reference 3, Vol.5. The list does not claim to be exhaustive; additional requirements are listed in a generic form in reference 1. The list on USOC is derived from references 4a-d., and from the author's personal information. It gives functional capabilities which may or may not coincide with functional modules.

3.1 Application Functions of FDS System X

Trajectory Generation

- orbit generation methods
  - conics approximation
  - AEG (Analytical Ephemeris Generator), which includes certain perturbations
  - (- numerical integration, in System Y .)
- basic orbit-related functions
  - base time ephemeris parameter initialization
  - coasting flight predictor
  - search time, stopping at certain condition(s)
  - coordinate transformation tasks

- orbital environment
  - shuttle orbit
  - some shuttle payload aspects

Launch Window Calculation and Launch Targeting

Launch Simulation

Maneuvering Functions

- general-purpose maneuvers
- rendezvous multiple impulse maneuvers
- two-impulse maneuvers
- V_∞ targeting maneuvers
- special targeting
- IUS targeting maneuvers
- midcourse maneuvers

On-Orbit Operations Analysis

- vehicle-to-vehicle AOS/LOS (Acquisition/Loss of signal)
- station tracking, AOS/LOS times
- orbital lighting conditions, sunrise/set, % darkness
- relative motion between shuttle and another vehicle (range, range-rate etc.)
- EREP sun elevation angle
- ground-track generation
- EREP site screening and pass
- moonrise and set
- glitter/glory
- ascending node counter
- beta angle
- attitude processing
- shuttle TDRSS viewing

De-Orbit/Entry Analysis

Consumable Analysis

- quick investigation of non-propulsive consumables
- OMS/RCS propellant usage

Display Functions

- listings
  - flight planning table
  - state information
  - on-orbit operations analysis data
  - scanning parameters
- plots
  - on-orbit operations analysis data
  - world map
- documentation generation system interface
- control functions
  - parametric scans
  - conditional and unconditional branching in processor sequencing

3.2 Application Functions of USOC

Trajectory Generation

- orbit generation methods
  - conics; generalized method for all types of orbits
  - improved AEG-like orbit generator
  - semi-analytical method (analytical with numerical updating each revolution)
  - numerical integration; all types of orbits, various levels of force evaluation
  - multirevolution method, superimposed on semi-analytical or numerical orbit generation method.
- basic orbit-related functions
  - base time ephemeris parameter initialization
  - coasting flight prediction
  - stopping functions, e.g. perigee passage, critical height
  - coordinate system change

- orbital environment
  - all types of orbits
  - earth and planetary orbiters
  - interplanetary probes
  - lunar orbiters
  - short and long-term predictions

Launch window calculations (scan for orbital stability only)

On-Orbit Operations Analysis

- station tracking, AOS/LOS times, visibility of planetary orbiters from Earth.
- orbital lighting conditions, sunrise/set, % darkness, earth and planetary
- ground-track determination, earth and planetary
- avoidance of radiation belts
- geomagnetic phenomena, scientific payload planning: magneto pause, bowshock, neutral sheet and neutral point

Display Functions

- listings
  - state information
  - on-orbit operations analysis data
- plots (stand-alone utility program using recorded information)
  - orbital elements versus time
  - on-orbit operations analysis data
  - world map with ground-track etc.
- control functions
  - parametric scans
  - cyclic execution of phases for any iteration procedure.

Examples of project-oriented user adaptions implemented in the past within USOC framework:

- operational maneuver software for ISEE satellites (reference 7)

- handling of two satellites simultaneously (reference 7)
- precision ephemeris generation used in conjunction with operational attitude problems
- investigation of dynamics of tethered satellites (reference 8)

Approved extension of scope of application:

- general-purpose maneuver capabilities
- interplanetary trajectory generation
  - patched conics solution
  - precision targeting method
  - controlled flight (e.g. solar sailing)
  - optimal control algorithms (study)

4.0 COMPARISON OF SYSTEM STRUCTURES

Any software that is to be considered a *system*, must comprise the following logical elements:

- set of application functions,
- set of data structures,
- control and management functions.

These elements may or may not be joined in corresponding physical blocks. In fact, their realizations in FDS and USOC are quite different, as will be described in this section.

The units of application functions, physically implemented as processors, modules, etc., are required to be compatible units of work by means of which a complex task can be built up. The data structures contain information which is globally assigned to the system (e.g. ephemerides of natural bodies or physical constants), and they contain information which can be specifically assigned by the user or is generated or modified by the functional units. The data structures are standardized such that they can jointly be accessed by compatible functional units. Finally, the control and management functions have the task to automate as much as possible the integral execution of the system and the data management. The purpose of automation is to increase both the overall efficiency and the practical reliability (reduction of user errors). The control and management functions also provide the interface of the user with the system.

4.1 Structure of FDS (System X)

4.1.1 Brief Description of FDS-X

The user always communicates interactively with the executive of the system in a dialogue that resembles a very high level language. The basic statement calls a processor which is a self-contained unit of work and associates an interface

table with the processor, which defines input and output parameters. Various modes of operation exist: *Manual* (immediate input of processor calling control statements and prompting by system), *Semi-automatic* (execution of predefined sequence of processors with optional manual interaction and prompting) and *Automatic* (execution of fully predefined sequence of processors with prompting). Finally, there exists the capability to create batch jobs.

Sequences of processor calls can be coded on sequence tables through an editor. Input and output data of processors are specified on data tables (called interface tables). The data interface tables are accessible through an editor program. The user has no access to the interior of the processors, only to the data. Interface of processors is established by means of data elements which are referenced in the interface tables.

A high degree of automation and foolproofness is an important goal of the system design. No knowledge of computer programming is necessary. The user has to be familiar with the flight design functions and their interrelations in order to be able to organize correctly the sequencing of processors and their interfaces by means of data elements.

Besides the normal user, only the level of system maintenance exists. Ad-hoc user modifications are not possible.

4.1.2 Functional Units

The functional unit of FDS-X is a *processor*. The processor can be classified into application processors and utility processors. Application processors are designed to perform, usually, a complete sub-task of flight design. The complexity of a single application processor ranges from moderately complex tasks (e.g. sunrise/sunset determination, given the vehicle ephemeris, etc.) to quite complicated tasks (e.g. deorbit/reentry targeting). Other examples are launch simulation and design, launch window calculations, general purpose

maneuvers, orbital maneuvers, IUS targeting, special targeting, coasting flight prediction, attitude and pointing, AOS/LOS , etc.

Utility processors have simple tasks such as data initialization, coordinate transformation and display functions. They are mainly needed to link the application processors and to display information to the user. A few utility processors are (in principle) a user definable part of the control logic. These are the processors needed to set up scanning loops over a sequence of processors or for conditional or unconditional branching within a sequence of processors. Conditional branching may depend on information stored in data elements. Examples of the first group are physical dimension specification, load state vector, print state vector, transform state vector etc. Examples of the second group are SCAN/ENDSCAN , IF , GOTO processors.

All processors are treated by the executive on the same level, irrespective of their complexity. Complete flight planning tasks are performed by chaining the necessary processors together, either interactively or by a predefined list of instructions (sequence table). Each processor can be executed independently, provided the necessary input data has been supplied by the user or by processors executed at previous stages. The processors are stored on a disk-resident library and are dynamically loaded and called by the control and management function (executive system) which obeys the immediate orders of the user or a list of directives. All users share the same processors.

4.1.3 Data Structures and Data Flow in FDS

Each processor fetches the necessary input data and stores produced data from and to, respectively, the processor associated standardized data structure, called interface table. Besides housekeeping the data, an interface table contains all necessary information on the input/output parameters as e.g.

storage class (disk or central memory), type of data (numerical, literal), completion flags (indicating if the information is complete or incomplete) and the data itself or a reference to a named location (data element) which contains the data. Processors do not directly access the interface tables. Information is transferred between processor and interface table and/or data elements by a service function of the executive system. This allows the provision of standardized procedures which guarantee the overall integrity of produced data. For example, a test on the completeness of input data can be made. This is very important since the processors can be executed independently. Eventually, the user will be prompted for missing information or the execution is terminated. A second source of input data, besides interface tables, are global read-only data files as e.g. JPL ephemeris data. Such information is read directly by the processors.

Interface tables are created and modified by the user by means of the interface table editor of the executive system. Default interface tables are stored in a library. The necessary actions of the user are to specify values directly to the parameters or to refer indirectly to the values stored in data elements by specifying the (unique) data element names instead of values.

Data elements are the vehicles to establish communication between the processors. These are named arrays stored on files or in memory which contain numerical information. Data which is the output of one processor and which is needed as input by another processor must be stored in a data element. A data element is created by the executive if the user specifies a data element name for an output parameter, and is referenced by another processor if the user specifies the same name as input parameter to that processor. Clearly, data element names must be unique in the user's active work area. As a consequence of the described feature, the user is responsible

for the organization of the data flow in a sequence of processors.

4.1.4. Control and Management Functions in FDS

In FDS-X the control and management functions are implemented in a program package which is commonly referred to as the FDS executive. This is an FDS-unique development running under the host computer real-time executive (RTE) operating system. The FDS executive generally controls the execution of FDS processors and provides the communications between the processors and between the user and the system. It also manages the permanent and temporary FDS data sets.

The following list displays a number of subsystem functions of the FDS executive. It is derived from more detailed documents on FDS-1 (reference 3, section 3 and 6) but the listed functions appear to be fundamental to the FDS concept.

The *configuration management* function links the user terminal with its associated linkage tables in the FDS manager. It provides the allocation of necessary computer system resources at session initiation.

The *FDS management* provides the control functions to run FDS under the host computer operating system. Besides the attention function which allows unsolicited user actions, e.g. interruption of sequence table execution, it provides, among others, the following functions:

- Schedule FDS executive and processor tasks.
- Control the communication of data structures between associated tasks.
- Management of the memory and disk resident data sets. This includes particularly the maintenance of a table of contents and of tables of pointers to sequence tables, interface tables and data elements.

The *executive director* includes the following functions:

- Provides terminal communication, inclusive prompting

of the user and extended (explanatory) prompting.
- Interprets user directives and takes corresponding actions, e.g. by calling the FDS manager to perform data set manipulations as to delete, copy or list complete data sets. Also, the sequence table and interface table editors are called by the executive director function. Other directives concern the mode of FDS operation, as manual, semi-automatic, automatic or batch mode.

The *execution controller* monitors the execution of processors through sequence tables by requesting the corresponding task scheduling by the FDS manager. Also, the associated interface tables are checked for completeness and the user is prompted for missing data.

Table editing is done by the sequence table editor and interface table editor. The sequence table editor serves to create and modify sequence tables which have a relatively simple format. The interface table editor serves to create and modify interface tables. This program has to store various classes of data into the relatively complicated structure. Also, some checking on the validity of user input is performed. An interface table contains information as e.g. overall size of the table, name of the associated processor, completion flags, data element names, possibly the data itself, and the size and dimension of data.

Processor services are used to transfer parameters and attributes of parameters between processors and the physical location of data in a standardized manner via reference to the interface table. These tasks are quite complex since data of various types may be stored in various ways on various places and devices.

Finally, there is the necessary function of *library maintenance* which must be used to add or modify processors or

default interface tables in the library. It will also record version numbers on the library members to avoid the association of incompatible versions of processors or interface table. It will also maintain a log of all changes.

4.2 Structure of USOC

The description of the structure of USOC in comparison to FDS is complicated for several reasons. The functional units are not all treated at the same level. Instead, there exists a hierarchy of them which is in addition dependent on the actual application. The data structures are organized in a modular form as are the functional units (modules). The control and management function is dispersed among the functional modules and a large portion of it is left to the operating system (OS) of the host computer. Finally, there is the feature of the preprocessor which does not contain but reflects the control and management functions of the application program which is to be generated.

4.2.1 Brief Description of USOC

The user starts USOC by answering mission analysis related questions to the interactive preprocessor. Accordingly, the preprocessor generates a sequence of job-control-language (JCL) statements to be used for the building of a problem specific absolute program from a library of relocatable subprograms. Also, it produces a series of input data groups to the absolute program. The sequence of JCL statements is transmitted to the standard host operating system service program which builds up the absolute in remote batch mode. Finally, the absolute is executed in remote batch mode.

The most basic level of user needs only to answer the questions of the preprocessor. He needs to be familiar with the methods used in mission analysis and planning. Foolproofness is not the highest priority requirement. Interface

of functional modules is established automatically. The user does not need to organize it.

A second group of users, which is strictly on the level of analysts, is able to perform problem dependent modifications by modifying individual subroutines on copies of the master versions and to include them in the absolute program in place of the standard versions. This feature allows a high degree of flexibility concerning new and unique situations. Obviously, the overall quality assurance of data produced on this level is partially moved to the responsibility of the analyst. Such a user also must have a good knowledge of FORTRAN and the host computer system. In addition, he must be familiar with the algorithms he wants to modify.

4.2.2 Functional Units

The fundamental unit of USOC is the *functional module*. This is the logical building block of the absolute application program, physically one or several subroutines. A functional module (FM) is defined by its distinct subtask in terms of performed work. A functional module may be made up of several FMs appearing in different positions of the calling hierarchy. A FM is identified by its highest entry name in the hierarchy which gives the (only) entry point of the whole module. Elementary FMs are usually physically represented by one subroutine (SUBROUTINE or FUNCTION type FORTRAN program) or sometimes by a group of subroutines which are always used as a complete unit. In some cases different versions of the same elementary FM exist. Each performs the same work, but internally the code may be different. A distinct version of an elementary FM is the basic physical building unit and is called a *library module* (LM) . If several LMs exist, the different versions are called homonymes, since they have the same FORTRAN entry point name, but are not the same code. A LM is identified by the element name (UNIVAC terminology) on which the module resides.

Examples of FMs are given by functions:

- execute the whole program,
- execute a phase,
- initialize physical constants,
- propagate state vector from initial to final state
- compute AOS/LOS ,
- take one step of numerical integration,
- compute atmosphere density,
- transform state vector,
- print orbital information,
- store information on graphics file, etc.

One can see that there is no correspondence between FDS processors and USOC modules. In some cases a FM contains the functions of several FDS processors, in other cases the functions are similar. The elementary FMs are usually much smaller than a FDS processor.

Examples of LMs are:

- computation of drag forces,
- initialization of Mars' gravitational potential coefficients,
- solve quartic equation,
- drive one phase
- evaluate increments for scanning procedure.

Examples of homonymes are the modules for the initialization of physical constants of planets. There exists a different version for each central body.

According to the present application of USOC, the topmost two levels of calling hierarchy are fixed. Each application program is built up from a number of phases, which may be executed in a cyclic manner. Each phase is broken down into the same subphases. From there downwards the same functional modules appearing in different phases may be built up in different ways.

Scanning and iterative procedures are implemented by using the cycling capability which means that a sequence of phases are performed repetitively. Any desired data can be iterated in this loop if corresponding user-modified LMs are inserted. Conditional branching within the sequence of phases has so far not been required, but could easily be implemented.

The LMs reside on an indexed library. They are built into the application program by specifying their inclusion in standard job-control language to the standard linkage program of the host computer (MAP-processor, linkage editor, composer, etc.).

There exists a master version of all LMs which is the default version for inclusion, but each user may copy it to his own library, modify it and include it instead of the standard version.

4.2.3 Data Structures and Data Flow in USOC

The data base of a USOC application consists of user input data, global read-only files, COMMON blocks, interface files, print files and plot files. -All components are standard FORTRAN.

User input data are given by NAMELIST data groups which, due to their literal form, are understandable by the computer and the user. They contain all run parameters except the ones which are no longer free, in case a particular version of a homonyme has been used or the input data selection has already been made at preprocessing time.

Each phase may need several input NAMELIST data groups. They are produced by the preprocessor on one file in the correct order. Due to their mnemonic names, they are easily identified. NAMELISTS are read directly by the modules which need the data.

Global read-only data belong to the system as a whole and are also used by other users working with different software on the same computer. They are read directly by the modules which need the data and are managed by standard FORTRAN library routines which are included automatically and work without explicit USOC control (except REWIND, BACKSPACE, etc.). Examples are sun, moon and planetary ephemerides, coordinates of standard tracking stations, etc.

COMMON blocks (CBs) have basically the function of establishing a data pool for the whole application program execution. Two classes can be distinguished. First there are "read-only data", e.g. physical and mathematical constants and run parameters read from NAMELIST and stored in CBs . Global information such as physical and mathematical constants is initialized by specific initialization library modules which are by default included from the USOC master library by the preprocessor. If a user wishes to use different values, he has to supply his own versions of these modules. The second class of CBs contain data which are possibly modified by one module and used by another module. These CBs could be called interface vehicles.

The set of CBs is structured in a modular way. The individual blocks are relatively small and always contain data which are logically related to each other. Examples are CBs for the state vector, epoch information, control parameters for force evaluation, temporary ephemeris information on planets, etc. Each module which needs certain information has simply the CB included which contains the desired information. The preprocessor makes sure that a module is included which contains the same CB and which produces or initializes the necessary data. Automatically, only those CBs are included in the absolute application program which are actually needed.

If the amount of data to be transferred between functional modules is too large, e.g. whole satellite ephemeris' data,

then the information is written to and read from an interface file. An example is the generation of ephemeris data and subsequent computation of sunrise/sunset information. The files can be reused at a later run or be used by different software.

Produced results are edited by print library modules and put to a printable file. Also, a plot information file is produced which can be read and displayed optionally by a stand alone program. Various levels of detailed output exist. However, if the user wishes to print additional data or to use a different format he is free to supply his own print modules.

The general rule is that a user produces all data and any specialized library modules for each project under a separate group of files identified on the USOC host computer (ICL 4/72) by a group identifier specified as a normal terminal directive. (The group identifier corresponds approximately to the qualifier on UNIVAC EXEC 8.) In this way, he may own several versions of USOC applications simultaneously without confusion.

All data files are automatically catalogued by the host operating system (OS) after the run of the application program. They are fully at the disposal of the user for additional runs or modifications. Any modifications to user input files, to the job-control images which define the composition of the absolute program and to the modification of modules, if necessary, are performed by the standard text editor of the host OS .

Interface between modules is established automatically by the modular concept of COMMON blocks. No user actions are necessary to ensure the correct data flow within the generated program. This method, which is very convenient, could no longer work if an independent phase-by-phase analysis would be desired.

4.2.4 Control and Management Functions in USOC

As mentioned before, many control and management functions of USOC are delegated to the host OS (which includes standard

FORTRAN features). The remaining part is dispersed among the library modules. The general philosophy is that each control operation is done at the lowest possible level in the hierarchy. This allows maximum flexibility to replace different versions of functional modules since control and management actions are only to be taken if actually required. This introduces the idea of modularity to a certain degree even on this level.

No executive system in the proper sense is present. It is nevertheless interesting to indicate where the control and management functions identified for FDS are to be found in USOC .

Configuration Management. The user terminal is linked to the computer system resources and to the USOC program control by the standard terminal interface of the host OS . This includes the interactive execution of the preprocessor.

Execution and Data Management. The only available attention function is to "kill" a running job if it turns out to produce nonsense. The program phases are linked by standard job-control language (JCL) while the full flexibility of overlay features is used. Data structures communicate automatically through FORTRAN features. The management of memory and disk-resident data is fully automatic by the host OS .

Executive Director. Terminal communication, inclusive of prompting is done by host OS and FORTRAN modules. User directives to manage files are performed by normal terminal command language. Editing is done by the terminal text editor.

Execution Controller. Linkage of FORTRAN modules is trivial. For the correct phasing, the standard OS control logic with associated control tables for overlay execution control is extensively used, but this is transparent to the user.

Editing Programs. The corresponding information to the sequence table of FDS is the sequence of JCL commands for the inclusion of library modules in USOC . This table may be

edited by the terminal text editor. The same applies to the input NAMELIST data files. Completeness checks, of course, cannot be performed. Syntax errors lead to terminal, OS or application program error messages. Error analysis may be difficult, mainly for people who are not familiar with the host computer.

Library Maintenance. Only library modules are to be updated. This is done by modifying and testing modules as if they would be a temporary user modification on his own library. If the modification is approved, the new relocatable element is put into the master library. Only one version of the same library module can exist. If updates are incompatible, the name of the library module must be changed and two versions of the preprocessor must be used.

4.2.5 The Preprocessor of USOC

The basic approach selected for USOC is "program generation". More specifically, in USOC a particular application program is built up from a library of relocatable program modules. Based on the guideline that the features of today's large computer operating systems should be used as much as possible, the linkage of the modules is not performed by a USOC system program but by the host OS . Consequently, the preprocessor generates a sequence of JCL-language items which is submitted to the OS collector program, which collects the specified modules into an absolute code. In addition, the preprocessor must provide an input data file to the generated program.

The whole logical structure of the system must therefore be mapped in a certain way into the logic of the preprocessor, such that a correct absolute code always results. The preprocessor has to process all branching decisions and has to take actions in the form of generating the necessary JCL commands. This includes the generation of the appropriate overlay structure.

The preprocessor is an interactive program, fully programmed in FORTRAN. As it is executed, it displays a question in terms of the mission analysis problem (not in terms of modules or processors etc.) and lists the available options that make sense, depending on previous answers. The user gives the answer in the form of single numbers or arrays of numbers (integer or real). These numbers are used first to control further questions and later for the generation of the output files. When all information is collected, it is processed to finalize all decisions and, as the last step, the JCL control cards and the input data file for the generated program are written. This information is forwarded automatically or manually to the next steps, which are collection and execution.

5.0 HARDWARE/SOFTWARE REQUIREMENTS TO HOST COMPUTER SYSTEM

This section gives a brief discussion of the requirements to the host computer. Due to the scope of the study, it is necessarily limited to itemizing some components of the hardware and software, as described in various documents.

5.1 Hardware Components

5.1.1 Flight Design System

The current development of the prototype system FDS-1 which contains a subset of the projected FDS-X application functions, is being installed on a configuration in which a Hewlett-Packard 21MX computer is the primary computer dedicated to FDS . It has a link to the Daconics documentation system, to a number of user terminals and to peripherals such as printers, plotters, disk-drives and tape-drives. The user terminal link also provides the capability to use tape cassettes.

In the planned final configuration the primary computer will be a larger computer on which most of the FDS-X functions are implemented. It will be linked to a large computer (UNIVAC 1108) which contains System Y , and to the HP21MX computer. It will support a number of user terminals. The HP21MX computer provides the interface to the Davonics documentation system. All computers will have their own peripherals such as printers, plotters, disk-drives, tape-drives, card-readers and microfilm units.

5.1.2 USOC

The USOC system is fully implemented on one large computer (ICL 4/72, similar to an IBM 360). The needed peripherals are the ones commonly installed in the environment of

general computer applications. These are printers, disk-drives, tape-drives, card-readers, card-punchers, paper and microfilm plotters, interactive graphics. A large number of user terminals and telecommunication interfaces are linked to the computer.

5.2 Host Operating System Software

For both systems, FDS and USOC , the operating system (OS) requires a time-sharing executive. This includes the option of running interactive application programs. The OS must supply the common service and utility programs such as compilers, a subroutine linkage program and data management utilities, as needed for copying, creating, merging files, etc. Both host operating systems must allow several users to work simultaneously without interference, possibly on different projects using the same software and parts of the same data base. The main difference in the two systems is in the extent of use of the OS . The host OS is transparent to the FDS user since he communicates only with the FDS executive, which provides the linkage with the host executive. In contrast, the USOC user communicates directly with the host executive, except when he runs the preprocessor.

6.0 FLEXIBILITY CONSIDERATIONS

Flexibility is a dazzling term; everyone has his own opinion as to its meaning. In fact every software system claims to be flexible.

In an ideal sense, flexibility in a software product means its readiness to treat "any possible" problem. This leads in practice to attempts to implement an exhaustive branching into predefined cases and subcases. However, it always happens that new problems occur which have not been foreseen. In such a case the software must be modified by adding new capabilities. In other words, the system must be adapted to a new requirement.

The design of FDS intends to offer the user a wide scope of application processors. Within the processors a large number of options will be available. This allows a high degree of flexibility within the scope of standardized capabilities. A result is that the executive and the processors become rather complex. Another fundamental requirement for FDS is that it must be able to be used by non-professionals. This requires detailed checking procedures which again adds complexity to the executive and the processors. If these requirements to FDS are satisfied, it will certainly be a very efficient and safe flight planning tool.

From the FDS user's point of view, the given task must be resolved into the offered capabilities. If this is impossible then the task cannot be accomplished. The user must request a modification or extension of the system at the system maintenance level. Modifications will affect the processors and the executive at various places; e.g. the format of interface tables and the prompt tables are affected. Data management functions must also be changed and the execution control logic must be modified. If different versions of a

processor are needed, the whole processor must be duplicated and added to the system as if it were a new processor. Even though adaptability is possible, it does not appear to be a high priority goal of the FDS design.

USOC , too, tries to offer a relatively large number of standard options that can readily be used through the preprocessor. However, in order to keep the structures simple, the variety of options does not require extensive branching into possible subcases. This is possible since the user may modify almost any module, if necessary. If, as in many cases, several mutually exclusive options exist, then the preprocessor selects only the appropriate modules for the generation of the program. *Very little unused code is included in a generated program.* However, once the executable program is built, the user has a limited set of options available. If he changes his mind, he must rerun the preprocessor. In those cases where it is likely that the user might wish to make experiments with options, and if the overhead in required storage is small, the possibility of switching options is retained in the modules. Also some flags are available to switch off certain functions in an executable program.

For the analyst who has to solve problems that have no ready-made solution in USOC but which are near to the standard capabilities of the system, there exists the feature of ad-hoc adaptability of the software. Starting with the identification of a functional module which would represent the solution to his problem, he will copy the module (in source code) which is closest to his requirement, to his own library file. Then he will modify the module and compile it. He will prepare a sequence of job-control images which specify the composition of the executable program. This is done by executing the preprocessor. Within the generated control images that name the library modules to be included, he then replaces the names of the standard modules by the names of his own modules. Then he executes the subroutine linkage program.

If additional or non-standard input data are required, he also has to modify the generated input data file. No other modifications are required.

For USOC , the requirement of flexibility has first priority. "Foolproofness" can be achieved only with users that communicate exclusively through the preprocessor.

Both systems include the feature of being extended continuously by addition of new capabilities. FDS is extended by an addition to the library of processors, with updates made to the executive at various places. USOC allows an addition to the library of modules and a corresponding update to the preprocessor. Compatibility problems are present in both systems.

7.0 TRANSPORTABILITY CONSIDERATIONS

System transportability means a measure of the readiness of the system to be transferred from one computer to another computer of different make. There are two aspects of transportability, one concerns the user's point of view, the other concerns the system programmer's view.

7.1 FDS Transportability

The FDS user communicates with the system in a FDS-unique language. The link to the host operating system is provided by the FDS executive. Therefore, transportability from the user's point of view is only a question how far the FDS language can be implemented on the new computer. Since the processors are written in standard FORTRAN, they should be easily transportable, unless the word length is reduced. If all features can be fully implemented on the different structure of the new computer system, then the transportability from the user's point of view is perfect.

The FDS system programmer is in a much worse situation since the implementation of many of the executive functions are strongly machine dependent. This is mainly due to the fact that almost all makes of computers use different modes for storing alphanumerical information. Also, a portion of the executive is programmed in assembler language, which is extremely machine dependent. The structures of the executive programs could stay nearly unchanged, but the detailed coding would need considerable modification. The subroutines in the executive of FDS-1 have therefore been recorded in a structure describing program design language.

7.2 USOC Transportability

For USOC the situation is quite different. Since the

module library and the preprocessor is coded in FORTRAN, the USOC-unique parts of the system can be transported easily. Some input/output details of the preprocessor would possibly appear slightly different to the user. The user that wishes to communicate with the system only through the preprocessor would have to learn a small subset of job-control language in order to be able to perform the basic steps of execution under the new host operating system.

The user that wishes to take advantage of the feature of adaptability, however, would need to have somewhat more knowledge of the new computer system. His ability should be on the level of scientific or engineering programmer. This includes the knowledge of the terminal command language, the job-control language and the data management utility programs. A particular requirement is the knowledge of the overlay facility.

From the USOC system programmer's view, changes will be necessary in the area of utilization of overlay features. The library and the question/answer procedure of the preprocessor could be left unchanged. There may be some small changes required in FORTRAN programs. No changes in the executive system would be necessary because no USOC executive exists. Also, no assembler language modules are present. The part of the preprocessor dealing with the job-control language image generation would have to be adapted to the new language conventions.

8.0 OTHER CONSIDERATIONS

8.1 Storage Requirements

The numbers given on FDS are taken from reference 2 and are preliminary estimates. The numbers on USOC reflect the actual experience. (One word assumed to have 36 bits.)

FDS System X is estimated to need one million words (4.5 M Bytes) and a executable code requirement of 40k words (180k Bytes) per user. FDS System Y is estimated to be four million words (18 M Bytes) of executable code.

The present USOC relocatable module library needs approximately 600k words (2.7 M Bytes). The corresponding source module library requires approximately 670k words (3 M Bytes). The core required per user varies between 25k words (112k Bytes) and 34k words (153k Bytes). Common data files, as e.g. JPL planetary ephemeris files are not included in the numbers on the USOC library. These files are shared by all users on the ICL computer system.

8.2 Experiments on UNIVAC 1110

A few experiments on the UNIVAC 1110 computer under EXEC8 have been performed to evaluate the possibility of implementing a USOC-like system structure. The first tests concerned the correct building of the executable program by the MAP-processor. An important problem was how to handle the multiple appearance of the same entry-name for FORTRAN modules. The MAP-processor does not operate in the same way as the ICL 4/72 composer, but a convenient solution was found.

A second test was undertaken to check the correct allocation of COMMON blocks by the MAP-processor. In this case, the MAP-processor coincided with the ICL composer.

A third group of experiments concerned the automatic creation of job-control images and their submittal to the executive system. This is a key part of the preprocessor. Some inconveniences were identified (e.g. conversion from files to elements or vice versa), but no basic problems appeared.

9.0 UNIQUE FEATURES OF EACH SYSTEM

The purpose of this section is to identify, in a summarized form, the main unique features of each system. Features which are common to both systems are not listed. Listing a certain feature means that it is not present in the other system, or is present in a considerably reduced form. Some of the features would be contradictory to the requirements or to the structure of the other system. Also, some are simply not implemented but could be implemented in the other system. Some of the features to be discussed here are irrelevant to the other system.

9.1 Unique Features of FDS

- The concept of using self-contained processors allows for a stage-by-stage building up of the whole required mission design data base. Also, the analysis of individual phases is facilitated by this concept.
- Input/output data of processors are separated from the processors in the data base established by interface tables and data elements. Parts of the data base can be accessed independently of the processors.
- Various modes of automation of the overall execution exist (interactive, semi-automatic, automatic and batch mode).
- The system allows to have the man in the loop at execution time to take some data-dependent decisions.
- The system supports operation by personnel on the skill level of technician. This is achieved by providing:
  - machine-independent FDS-unique conversation scheme. This could be termed a type of high-level conversation and programming language.
  - prompting of the user for some classes of errors and for missing data.

- The scope of application processors in the planned final version provides end-to-end flight design capabilities, from launch to reentry, within a set of standardized functions.
- The standardization of processors and data structures is intended to provide maximum efficiency for the design of routine missions.
- Flexibility of the system is achieved by providing a large number of user-controllable parameters, in order to cope with the variability of the flight design problems, without recoding the processors.
- The system provides an interface to a semi-automated documentation system (Daconics) in order to allow the error-free and quick insertion of flight design results into standardized documents.

9.2 Unique Features of USOC

- Particular application programs are generated automatically or manually from a library of functional modules. A relative hierarchy of functional modules exists in the sense that a functional module may be built up from functional modules of a lower level, etc.
- The problem-dependent part of the data base is organized in a modular way. It is established automatically by the library modules by means of incorporated COMMON blocks.
- The automatic generation of a specific application program is made possible by the interactive preprocessor. The conversation of the user with the preprocessor takes place in terms of problem related questions/answers, not in terms of modules or processors.
- The interface of phases is established automatically; no user actions are necessary.

- All levels of accuracy/execution speed are available within the same framework.
- The system is designed to allow the solution of complex problems which need the interaction of phases. Feedback of arbitrary data produced in one phase into an earlier phase is possible. This feature is accomplished by executing a sequence of phases in a cyclic and iterative manner. (This will usually need the problem specific adaption of some modules).
- Adaptability to new, unforeseen and unique problems has high priority. The user can easily insert his own versions of any module into the generated program.
- Maximum advantage is taken of the available host computer operating system. No explicit executive program is present in USOC .
- Although a single program is generated for each application, almost no additional storage is required for complex multi-phase applications. The code for the different phases is never simultaneously in core.
- Loading only the modules (code and data) actually needed for the particular application of the phases allows the minimization of overall memory storage requirements.
- Output data can be stored on files and reused by another program that is generated in a different manner, or even by other software.
- The program library allows generation of high level functional units which act as processors for specialized mission planning tasks.

10.0 CONCLUSIONS

Although there exists some overlapping of the scope of applications of USOC and FDS , the structures of these two systems are very different, according to their different basic requirements. Their requirements are in fact, incompatible. While the FDS is designed to meet the requirements of a standardized production tool, USOC is designed for rapid generation of particular application programs. The main emphasis in USOC is put on the adaptability to new types of missions.

A software system having a USOC-like structure, adapted to the specific needs of MPAD , would be appropriate to support planning tasks in the area of unique STS missions. There appears to be a need for such an additional system within MPAD

REFERENCES

1. Davis, E.L.: "Proposed Goals and Concepts for STS Operations Flight Planning", JSC Internal Note No. 76-FM-10, March 1976.

2. Folk, R.A. and Young, W.A.: "Flight Planning System II Level A Requirements for Flight Design System", JSC Internal Note. No. 76-FM-71, August 1976.

3. "Flight Design System 1, System Design Document", JSC Internal Note No. 77-FM-18.

   a. Volume I: Section 1 - Introduction, Section 2 - Overview, Section 3 - User Interface, April 1977.
   b. Volume III: Section 5 - Application Processor Library, February 1977.
   c. Volume IV, rev.1: Section 6 - System Architecture and Executive, July 1977.
   d. Volume VI: Section 9 - Standards, May 1977.
   e. Volume VII: Section 10 - Utility Support Software, May 1977.

4. "Flight Design System (FDS) System X Design Concepts", lecture presented by Bob Davis, NASA/JSC/FM6, June 1977.

5. "Unified System for Orbit Computation - USOC",

   a. Maag, W. and Mueller, A.: Volume 1 - Software Organization, Orbit Generation Part, User's Guide, ACM Report, July 1976.
   b. Maag, W. and Starke, S.: Volume II - Orbits in the Solar System, ACM Report, June 1977.
   c. Starke, S. and Maag, W.: Volume III - Auxiliary Calculations, ACM Report (to be published 1977).
   d. Boissieres, J.: "Interplanetary Orbits in USOC", ACM Working Paper, (in preparation).

6. Graf, O. and Mueller, A.: "A New Approach to Software Systems for Spacecraft Mission Planning and Analysis", ACM Memorandum No. 158, February 1977.

7. Janin, G.: "Preliminary ISEE-B Orbit Maneuver Software for the Routine Phase", ESA, MAD-Working Paper No. 27, January 1977.

8. Maag, W. and Wehrli, R.: "Interim Report on Dynamics of Tethered Satellites", ACM Report, August 1977.