AN INTERACTIVE DATA REDUCTION SYSTEM FOR REAL-TIME MILLIMETER WAVE EXPERIMENTS

by

Steve R. Kauffman

Thesis submitted to the Graduate Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

W. L. Stutzman, Chairman

C. W. Bostian

A. W. Bennett

March 1978

Blacksburg, Virginia

TABLE OF CONTENTS

An Interactive Data Reduction System

For Real-Time Millimeter Wave Experiments

	Page
Table of Contents	ii
Chapter 1. Introduction	1
Chapter 2. Experiment Definition and Acquisition System Characteristics	3
2.1 Experiment Description	3
2.1.1 General	3
2.1.2 Acquisition System Hardware and Software	5
2.1.3 Functional Analysis of the Acquisition Task	6
2.1.3.1 Basic Operation	6
2.1.3.2 Special Considerations	9
2.1.3.3 Data Storage and Transfer	9
Chapter 3. Data Processing Requirements and Considerations	11
3.1 General Processing Goals	11
3.2 Specific Processing Goals	14
3.3 Factors Imposed by the Processing Environment	14
3.3.1 Executing Applications in the Batch Environment	15

Page

3.3.2 Executing Applications in the CMS Environment 16

3.3.3 Summary 19

Chapter 4. Software Design Methodology 21

4.1 The State of the Art 21

4.2 Segmentation of the Task and Detailed Design . . 22

4.2.1 Overall System Aspects 22

4.2.2 Preprocessing 24

4.2.2.1 Basic Program Organization . . . 24

4.2.2.2 Preprocessor Data Flow 26

4.2.2.3 Preprocessor Internal Construction 29

4.2.2.4 Performance 32

4.2.3 The Preprocessing/Analysis Interface . . . 32

4.2.4 The Data Exception Control System 33

4.2.5 Reduction Tasks 35

4.2.6 The Interactive Interface 41

4.2.6.1 Defining the Command Language . . 42

4.2.6.2 Implementation Considerations . . 44

4.2.6.3 The FORTRAN Interface Module . . 45

4.2.6.4 The Interactive Modules 47

Chapter 5. Conclusions 49

Bibliography 51

Vita 57

Chapter 1

Introduction

This document presents the design procedure employed in the generation of the ONWARD (ON Line Wave Analysis Reduction and Display) system and its associated software components. First the software task is defined. Some current software engineering methods are then explored to determine their applicability. Next, a functional analysis and design is performed to yield the final software configuration standards. Techniques are presented for the organization and access of real-time experimental data using modern database methods. Finally, some conclusions are drawn on the utility of the methods presented in the general real-time data reduction case.

This thesis forms the first part of a two part document and is concerned primarily with pre-coding engineering aspects of the system. The companion document (1), contains a detailed description of the operation of the code, and will be referred to frequently in the remainder of this paper.

The author wishes to gratefully acknowledge the following individuals, without whom this work would not have come into being. , and made possible this learning experience through their confidence, patience and counsel.

., who elegantly programmed the data acquisition system and some of the ONWARD modules, is to be thanked for his frequent good advice, too frequently not taken.

, was very helpful in making his wide experience in data processing available to me. provided considerable assistance in the coding, maintenance and documentation of the statistical modules. Finally, no acknowledgement given here can adequately describe the speed and quality of work given in this effort by , who typed this thesis and its adjunct document (1).

Chapter 2

Experiment Definition and Acquisition System Characteristics

2.1 Experiment Description

2.1.1 General

Since January 1972, the Electrical Engineering Department at Virginia Polytechnic Institute and State University has been involved in experiments investigating the depolarizing effects of precipitation at millimeter wavelengths. This work has progressed through various phases, until now a rather complex three frequency system is in operation, using the CTS and COMSTAR communications satellites and an earth terminal located on-campus. The experiment task may be put most simply as an attempt to correlate variations in communication channel performance with various precipitation events . Single event studies as well as overall statistical performance are required results from the experiment. The entire experiment is under the control of a PDP 11/10 minicomputer and is completely automated. The remainder of this chapter is intended to describe the experiment so that its data processing requirements and constraints can be accurately characterized. For this reason, considerable detail will be omitted in area which are not relevant to the data processing problem. The interested reader is referred to the large body of publications which cover this experiment in depth (7,8,9,10,11,12,13,14,15,16).

The on-campus earth terminal houses three complete receiving systems, operating at 11.7, 19 and 28 GHz. Although these systems are specially tailored to the experiment, they are in many respects quite similar to the satellite earth terminals in use around the world. Each receiving system produces outputs which may be classified as signal parameters and status parameters. Both are monitored in an asynchronous fashion by the computer. A group of meterological instruments is employed to measure current weather conditions. These include an electronic thermometer, wind speed sensor, wind direction sensor, a system of tipping bucket rain gauges, and a 15 GHz weather radar. These are also monitored by the computer.

The data processing system also performs certain control functions over the experiment hardware. The antenna for the 11.7 GHz is mounted on a digitally controlled pedestal which receives periodic pointing commands from the computer. The true position of the pedestal is reflected back to the computer over a seperate hardware channel, so that the computer can judge the integrity of pedestal operation. The operational state of the weather radar is also under computer control, which generally commands it into operation only during weather events. Finally, the computer has the capability of testing the rain gauge network in order to detect failures in the telephone lines which connects the rain gauge network to the computer.

2.1.2 Acquisition System Hardware and Software

The primary element of the on-site computer system is a PDP 11/10 central processing unit. Attached to the CPU, via the Digital Equipment Corporation (DEC) UNIBUS system, are an RK05 moving-head rigid disk drive, a CT60 dual cassette drive and a Digi-Data nine track, 800 BPI tape system. Additionally, there are three low speed terminals attached for system communication and a thirty two bit parallel interface for communication with external experiment hardware. The thirty two bit interface is connected directly to a device known as the experiment controller. The controller provides a central interface point between all other experiment hardware and the data processing system. It is capable of performing analog-to-digital conversions, interrupt, timing and identification functions. It is constructed entirely of standard TTL logic and was designed and contructed locally. The computer employs the controller hardware to execute all acquisition and control functions.

The software associated with the computer consists of DEC supplied system software and locally developed applications software. The system software consists of a resident monitor and a group of device handlers which collectively supply high level I/O support and a structured file system. The version of the monitor being used is capable of supporting a single ongoing task in a high level language, but an interrupt structure of considerable complexity can be supported at the assembly language level. The application software is written in FORTRAN and assembly language. A FORTRAN preprocessor known as MORTRAN was also used to lend some structuring to the more complicated modules. This software performs all acquisition and control functions and additionally handles communication to the three system terminals. For more information on the DEC software components see (17,18,19).

2.1.3 Functional Analysis of the Acquisition Task

2.1.3.1 Basic Operation

It is useful at the outset to neglect all device and format considerations associated with data collection and to examine this function from a purely logical viewpoint. In this context some useful definitions may be made. Every logically complete unit of information acquired through any sequence of events shall be referred to as a data record.

Each data record consists of four pieces of information. These are the Julian date, the time (using a Coordinated Universal Clock), a binary code indicating the source of the record, and a binary code indicating a data value. The binary code which reflects the source of the data record is referred to as the 'what number'. The only additional information required to determine entirely the epoch of a data record is the year, and this is embedded in the data flow as a special record inserted at appropriate times. Beyond the special year record, the data records may be subdivided into three types. These will be called 'data points', 'status points', and 'rain gauge points'. A data point is a record associated with the acquisition of an analog value from some piece of experiment hardware. Under normal conditions, these constitute the bulk of the data flow. A binary code reflecting the analog value is embedded in records of this type. A status point is a record associated with a change in some aspect of experiment operation which is can be described as being in one of two states, true or false. In this case, that portion of the data record that would otherwise contain a binary equivalent of some analog value is used to contain this true/false indication. A rain gauge point is a record that indicates a signal was received from one of the tipping bucket rain gauges. The field used by the previous two types of points for either data or status is always zero in this case. The rain gauges consist simply of a measuring bucket and a switch mechanism. When the bucket fills with rain water, it tips and activates the switch mechanism momentarily. The computer then constructs and stores a rain gauge point. These points supply two types of information. The total number of trips can be transformed into a total rain accumulation. The time interval between trips can be converted to a rate of rain fall.

The hardware/software mechanism by which the aforementioned data records are accumulated consist of two basic categories. The status and data points are the result of sampling, and the rain guage points are the result of direct interrupts. The sampling of status and data points in controlled by controller hardware, which provides a firmware selectable range of sampling intervals. When a sampling interval expires, the eligible what number is flagged and the computer is interrupted. This causes the point to be transferred into the computer. The software then applys significance criteria to the point and either saves it as meaningful or discards it. A slightly different procedure is used by the rain gauges. When a gauge trips, the controller immediately interrupts the computer , which records the trip. Trips may occur at any time.

2.1.3.2 Special Considerations

In certain cases, special processing must be applied to the data flow to ensure its correct interpretation. These situations are the cases of program startup and program stop in the on-site computer system. Two special what numbers have been assigned for the purpose of recognizing these events. Whenever a start or stop indication is encountered in the data flow, all data points become undefined and all status points become false. In the program stop case, this situation persists at least until the next program start record. Data points then become defined as they are encountered in the data flow. Status points are altered from their false state by a similar mechanism. In addition to controlling this undefined state, the program start and stop records perform the additional function of embedding the current year in the data flow. This information is contained in what would normally be a data or status field in both records.

2.1.3.3 Data Storage and Transfer

Data records are accumulated by the on-site computer as a direct access file on the rigid disk unit. This type of storage greatly facilitates report generation and editing. When the disk file approaches capacity, its contents are copied to the nine track tape. This tape is then carried to the university computing center where it is copied by special software onto the two duplicate database volumes.

The data are now available for the reduction task.

Chapter 3

Data Processing Requirements and Considerations

3.1 General Processing Goals

At the outset of the project, the ultimate aim of the data reduction system was seen as the production of some time history plots. However, it soon became clear that the end desires for data reduction would change as our familiarity with the data increased. The goals of the data reduction system were thus formulated as follows. First and foremost, it was desired to make the writing of a reduction application (i.e. a type of analysis) as easy as possible. Second, it was desired to make the internal structure of the reduction application entirely independent of the structure of the experiment data. Third, it was deemed important that a standard be established to enforce a consistent interpretation of the tape among experiments. Finally, the decision was made to try for an optimum in execution performance consistent with a reasonable development interval. The following paragraphs will detail these points.

From the preceeding descriptions, it is clear that the execution of anything but the most straightforward type of analysis will be a problem of some difficulty. This problem is made more formidable by the requirement of efficient operation. The difficulties are primarly the result of the asynchronous monitoring of data sources and the computations required to obtain a rain rate from tipping bucket rain gauge data. The asynchronous monitoring would force all analysis routines to keep running values of all parameters of interest (some of which may not even be accesssible at a given point on the tape). Additionally, all system status points relevant to validating the data would have to be maintained. The computation of the rain rate from tipping bucket rain gauges eliminates the possibility of processing the tape data in a single pass. To compute the rain rate at a given instant, the preceeding and subsequent rain gauge trips must be located. The time lapse between these trips can then be transformed into rain rate. An easily used access technique for the experiment data, should thus insulate the analysis programmer from these difficulties.

The second goal of program/data independence was felt to insure the project against a large amount of recoding, in the event of a subtle acquisition or processing system change. If all reduction tasks access the data through an interface of fixed design, the reduction tasks become immune to changes beyond this point. The interface must be defined generally enough that any conceivable type of analysis would be possible. It is easy to visualize how a small change at the acquisition system could have disabling effects on all analysis software not constructed in this manner. This type of database technique is widely covered in the literature (20).

Since it was expected to have several experimenters analyzing data in diverse ways, the potential for inconsistent interpretation of the data between groups is clear. It was deemed necessary that either an administrative standard be generated for software processing of the tape (and hope that it would be obeyed) or force compliance by eliminating all types of calibrative, validitating, and rain rate processing from the reduction tasks. This would necessitate some sort of pre processing. The possibility of publishing internally inconsistent data would hopefully be avoided by one of these plans.

The goal of efficient execution is one that is pursued as an end in its self by many programmers, but unfortunately (or perhaps fortunately) this was not the motivation in the current situation. The sheer volume of the data which this completely automated and extremely reliable acquisition system was capable of assimilating surpassed all expectations. Under the proper circumstances , the data collection rate may rise as high as twenty points per second. As a result, it was decided to make every effort to produce code which contained no wasteful constructions. It was felt that every reasonable effort that could be made in this effort, short of hand optimization of object code, would buy additional processing power in the end.

3.2 Specific Processing Goals

In addition to the general goals presented in the previous section, some specific capabilities were established. Due to the very large volume of data and the need to process it it in a non time sequential manner, some method of selectively examining the raw realtime data was felt necessary. Also since most of the data would have to have some sort of human screening before detailed reduction could begin, an interactive approach to this function was dictated. This was seen as a potential application for interactive graphics. Finally, since many members of the project staff would potentially be interested in a given analysis, the ability to produce multiple copies was desired. This would help reduce the secretarial effort associated with analysis.

3.3 Factors Imposed by the Processing Environment

It is certainly a fair assessment to state that the University Computing Center at VPI&SU offers the scientific programmer considerable processing power. The hardware configuration consists of two System 370 model 158 central processing units, equipped with a total of nine million bytes of core storage and complete arithemetric hardware, eleven 3350 disk drives, two 3330 disk drives, five 3420 tape units, a 3705 communication processor and various terminals, printers and card equipment.

Certain logical aspects of this system organization are important. This hardware supports two systems, one being an MVS (Multiple Virtual System) batch system, and the other being a VM/370 (Virtual Macine Facility/370) multiple virtual machine system. The VM system is utilized primarily as a time sharing service through the facilities of CMS. The virtual machine facility is also capable of running virtually any 370 type software concurrent with its normal function. This is frequently used for operating system testing. The user has at his disposal, effectively, the capability of a dedicated 370 CPU and can load and run any operating system software available to him (2).

3.3.1 Executing Applications in the Batch Environment

From the preceeding description, one might conclude that the facilities available to the user are quite extensive, in terms of processing power and direct access storage. Unfortunately, while the former is true, the latter is not. The 370/158 is definitely in the class of high performance mainframes, although new technology pushes it ever farther dcwn the list. It is equipped with a large main store which is argumented by the virtual memory hardware of the model 158. Thus it can handle very large programs, if the programmer is willing to trade off execution speed. The direct access space situaticn is another matter. Although the ccst of disk allocaticn is quite low, the amount of space that may be obtained by a given user is limited by an administrative algorithm. My experience has been that a large user may not find the space required. Fortunately the tape pclicy is quite liberal. A user may rent computer center tapes to be retained in the center's library, or he may check his own tapes into and out of the center. Jobs requiring multiple tape vclume mountings are permitted. The tape drives are all nine track and read all ccmmon densities and formats.

3.3.2 Executing Applications in the CMS Environment

As previously mentioned, the default operating system provided to VM/370 users is CMS. CMS is somewhat unique in that it provides single user support in a large virtual machine. It is essentially a conversational system which permits the user to manipulate the resources of his virtual machine conveniently. In order to execute the conventional IBM OS language processors in a CMS environment, CMS simulates to the extent necessary an OS environment. Most of the data management macro instructions are provided

(LOAD, LINK, XCTL, GETMAIN, FREEMAIN, etc.) as are most of the I/O access methods (BSAM, QSAM, BPAM, and BDAM are directly supported). VSAM is also supported to some extent, but is sufficiently difficult to use that its popularity is low. The Indexed Sequential Access Method and multitasking are not supported. The interface to the batch system is somewhat tenuous. Data may pass into the batch system only via submitted card image job streams. Data may pass out of the batch system into VM/370 through either the VM spooling mechanism or by access to OS disks. The OS disk access is restrictive in the following aspects. Only certain volumes may be accessed and only read access is permitted. Only datasets written with BSAM, QSAM, or BPAM may be read and experience indicates that this interface also suffers from comparatively poor performance. These restrictions are discussed more fully in the IBM documentation (2).

The above capabilities allow almost any type of logical process to be performed on CMS. The hardware is simulated in the most minute detail (although there are a few exceptions), even to the extent of simulated communication lines and virtual channel to channel adapters. The gap between simulation and reality is, however, an important consideration. Any virtual hardware machine must, in itself, execute more slowly than an actual hardware machine. The claim has been made by IBM, however, that a system composed of multiple virtual machines will have throughput exceeding the single hardware machine (2). Unfortunately this is not a possibility in this application. In all fairness, it must be stated that many of the VM functions are simulated by hardware and as a result perform well (2).

There is an additional level of simulation occuring, however, in the application of the OS language processors that must be considered. This is the simulation of the OS environment. Two types of simulation are provided. First, all I/O performed by the OS program is handled by a simulated access method. When an OS I/O macro is executed, a CMS module performs the actual I/O. Second, certain required supervisor calls must be simulated to handle the supported data management macro instructions. This is also handled by a CMS module. Neither of these points is on the surface of any concern to the user, in fact these simulations could in fact perform better than their OS counterparts. There is, however, an important point to be considered in the execution of I/O. CMS performs all physical I/O in 800 byte blocks, regardless of specified blocksize or record format. As a result, reading what was programmed as a 6000 byte physical block, will actually require eight I/O operations. The problem is particularly aggrevated by a random access operation, where much of the data in a given block will be unused. Further information is available in the IBM reference manuals (2,21).

3.3.3 Summary

The previously mentioned facts impose certain restrictions and considerations on the final software system design. Since tapes may only be accessed on the batch side of the system, the first stage of processing must logically occur here. Once the data are moved to an appropriate disk (in whatever format), it will be accessible to both batch and interactive applications. The limited availability of disk space dictates that organization of the disk data file organization be as compact as possible. Since the type of dataset organizations readable from the VM/370 side of the system are limited, an organization must be chosen to provide both batch and interactive compatability. The simulated nature of the OS access methods on CMS results in an additional difficulty. There exsists no blocksize which affords maximum performance (minimum physical I/O) on both sides of the system. A blocksize of 800 bytes provides maximum performance on CMS, while a blocksize of roughly 6000 bytes provides best results on the batch system (6000 gives the best blocking factor for both 3330 and 3350 disk drives). I felt that constructing the file for best operation on the more conventional batch system was the preferable choice, since more lengthy statistical computations would probably be performed on the batch machine.

Chapter 4

Software Design Methodology

4.1 The State of the Art

The body of literature dealing with software engineering has grown in recent years at a rapid rate and its emergence as an important technical pursuit is a virtual centainty (26). As in any new discipline, the foundations must pass through a formative, if not controversial phase. I feel fairly confident that the following two generalizations will cause little dispute. First , successful software engineering is the application of some methodology to software system design in order ot obtain the maximum result for minimum cost, and second , structured programming is one such methodology. Structured programming is usally defined as consisting of a mangement aspect (chief programmers, programming teams, librarian, etc.) and a coding aspect (modularity, structure, functional design, interface design, etc.). Since the major portion of systems design and implementation was concentrated in a single person, no situations were provided for the application of the management aspect (except perhaps in the area of the author's personal affairs). A considerable amount of the structured code approach has been applied. The application of these principles will be discussed in the following section.

4.2 Segmentation of the Task and Detailed Design

4.2.1 Overall System Aspects

One of the most useful techniques provided by the structured programming approach is the reduction of the major task into logically complete and manageable functions. In addition to being logically complete and manageable, the final group of functions should be non-redundant and a concise expression of the basic logical operations that support the overall task.

From the preceeding goals and considerations, (Chapter 3), the first level of structure can be formulated. This is shown in Figure 4.1. This diagram reflects all the basic requirements for the reduction system. The preprocessing scheme would permit the selection of a portion of the experiment database (thus conserving precious on-line disk space) and also perform all remaining calibration and rain rate computations. This fixes the interpretation of the tape, concentrates the logic of interpretation in a single place, and greatly eases the processing burden of the reduction tasks. The output of preprocessing would then be made available to the reduction task through some sort of

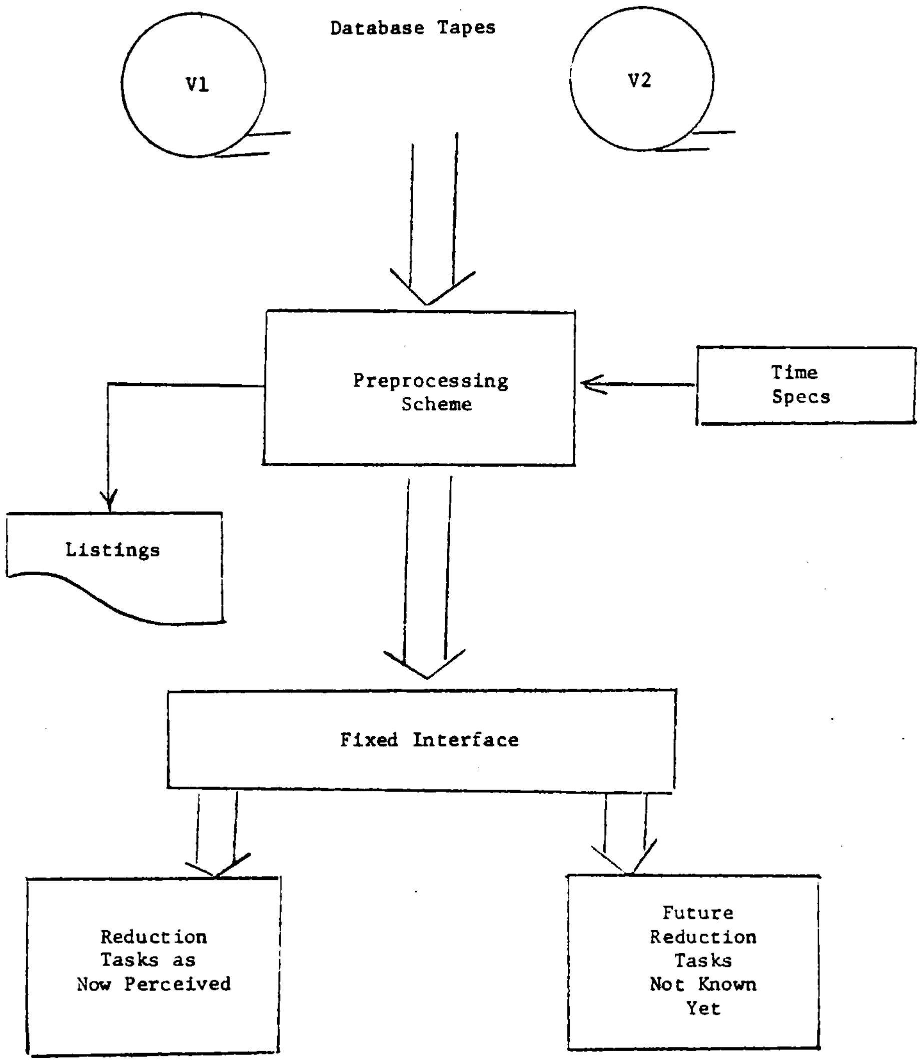


Figure 4.1. Basic System Data Flow

logically fixed interface. This would provide the desired program data independence.

4.2.2 Preprocessing

4.2.2.1 Basic Program Organization

To further refine the functions indicated in Figure 4.1, the preprocessing scheme was attacked first. This design stage is detailed in Figure 4.2. It is interesting to note, however, that the fixed interface between preprocessing and reduction permits the two primary phases to be considered individaully. This flexibility could potentially allow programming teams to proceed asynchronously on these two aspects in strucured programming management fashion. In keeping with the goal of minimum processing cost, the preprocessing scheme was specified to have the ability of selective what number retrieval. This would prevent processing of undesired parameters. In a similar vein, any data printed by the preprocessor was to be selectively controllable. In the face of these two requirements, a general execution option facility was specified to accomodate the aforementioned input and other as yet unknown control options. In order that the most utility be obtained from a given preprocessing run, preprocessor outpt should be available to the largest number of reduction tasks. This was satisfied by specifying the output of the preprocessor as residing in a permanent on-

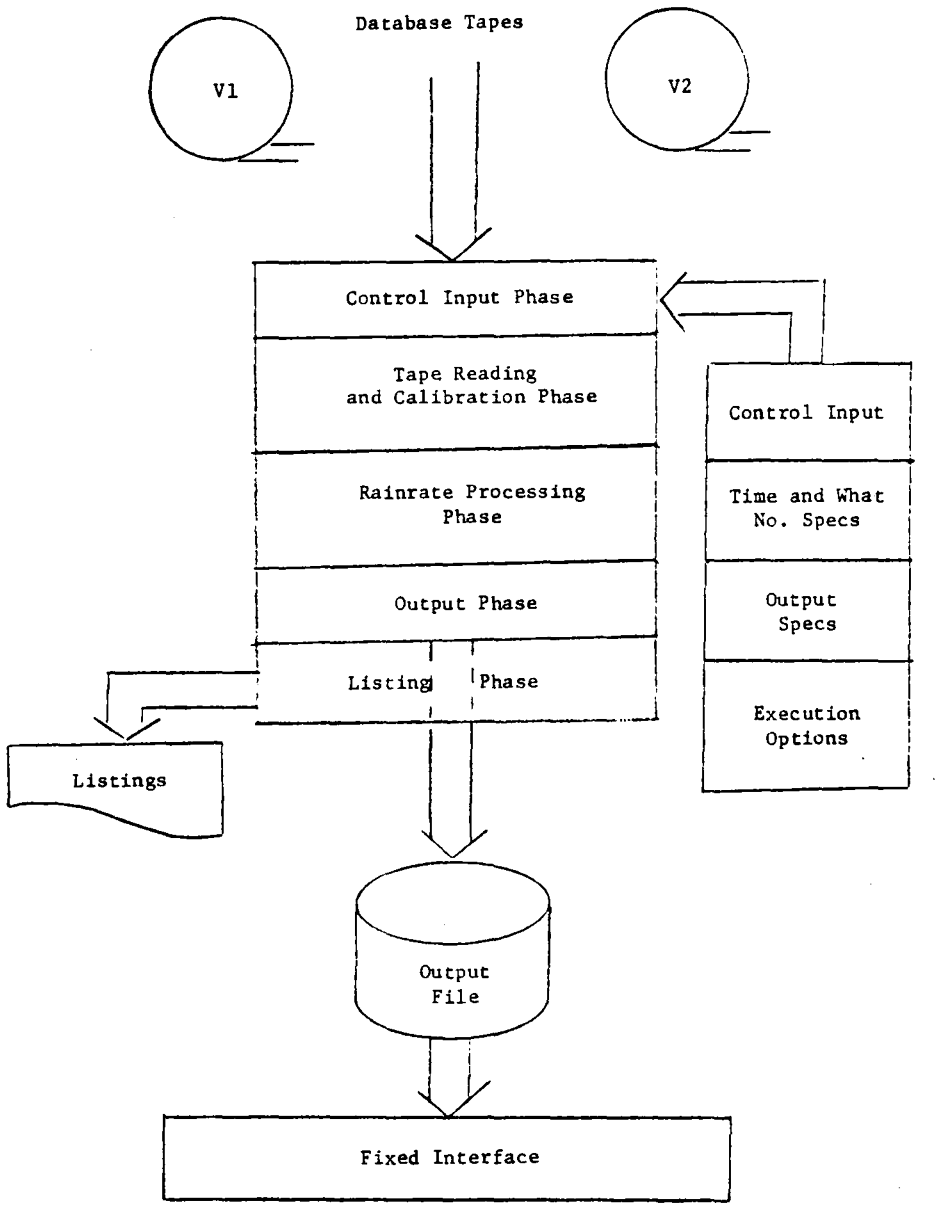


Figure 4.2. Preprocessor Phase Divisions

line disk dataset, accessible to all users of the system. Tentative processing phases of operation were assigned to the preprocessor at this point, based on knowledge of the logical processes required. A number of iterations were performed on the allocation of work between phases. The divisions shown in figure 4.2 are the final result.

4.2.2.2 Preprocessor Data Flow

As mentioned previously, the essential requirement of this preprocessing phase is to provided a compactly organized, easily utilized and efficiently produced interpretation of the tape database. Toward this end, the following specification was established for the format of the preprocessor output, or process file, as it will henceforth be termed. The process file shall consist (logically) of a group of time sequenced records, accessible in any order, each of which is a complete reflection or snapshot of the experiment at a given time. The 'completeness' of this reflection will, of course, be tempered by any selective options activated in the preprocessor. The timing of the snapshots will be determined by changes in experiment data. When a parameter changes (data or status) a new snapshot (or record) is created. Figure 4.3 demonstrates this concept.

On the surface, the job appears relatively straightforward. The contents of the tape must be used to

- A file of data containing the time history of the experiment over the specified generation time and subject to the specified generation options. Each record is referred to by number (1 through the last record), and contains the values of all experimental parameters at the time represented by the record.

(Starting Time Specified by User)

1

2

.

.

.

.

i [DATE] [TIME] [YEAR] [WHATC] [DATA VALUES] [STATUS VALUES]

.

.

.

.

N

(Ending Time Specified by User)

- The user accesses the file by calling the common access routine, GETDAT, specifying the Record Number and the What Number for the quantity desired.

```
                 PASSED
CALL GETDAT (I,IWHT,IWHTC,DATA,ITIME,IDATE,IYEAR,IERR)
                                RETURNED
```

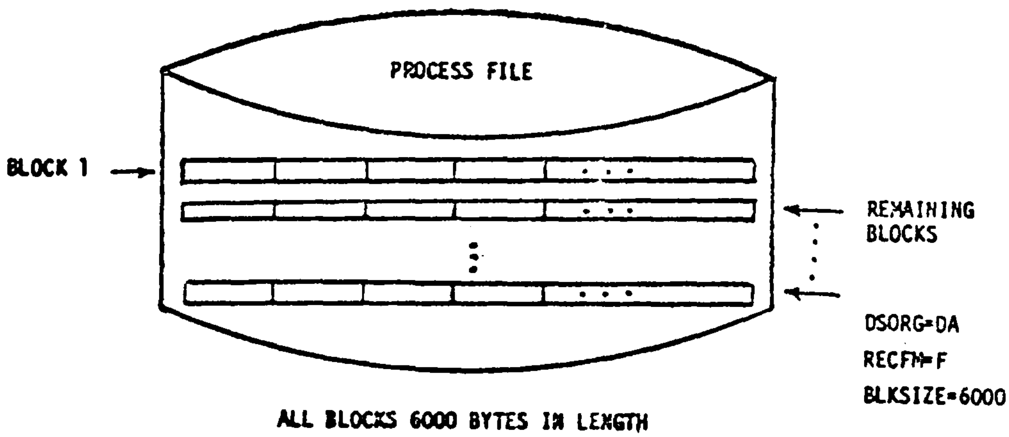
Figure 4.3. Snapshot Data File Concept

produce a sequence of updated records over some specified time interval, subject to the specified options. This is, in essence, what the preprocessor does. However, the details of implementation, when coupled with a requirement for efficient execution, presented some interesting difficulties. It was viewed as an absolute necessity to avoid making more than a single pass over the tape, since its contents would probably become very extensive. This inability to preview the data before actual processing, caused two immediate problems. First, since it is unknown which what numbers might be encountered in a given run, some means of dynamically accomodating them was necessary. Second, since the rain gauge trips cannot be processed in a single pass, these must be directed to some form of auxiliary storage. The first difficulty was overcome by dynamically allocating record storage as required and in effect expanding the records as processing proceeded. The second problem was solved by writing the rain gauge trips to an auxiliary direct access file, for later processing.

The ability to accomodate the input data stream dynamically posed one final difficulty. The resulting contents of the process file would be entirely dependent on the data that was actually encountered. Hence, the reduction side of the system must be provided with a mechanism for correct interpretation of the file. The final design solved this problem by borrowing two techniques used in database systems for some time: (a) an organizational or header record and (b) , a common access routine (CAR). This technique is illustrated in Figure 4.4. The organizational record contains complete information on the contents of the data records. On first access to the process file, the header is read by the common access routine into permanent storage and is then used to construct pointers into the data records so that the data is properly accessed. All physical I/O occurs in 6000 byte blocked which are dynamically segmented into the appropriate logical record size. The blocksize of 6000 was chosen to give the most efficient utilization of space on both the IBM 3330 and 3350 disk drive units. This design permanently and completely solved all access and program/data independence problems.

4.2.2.3 Preprocessor Internal Construction

The final decision to be made prior to the actual coding was the selection of source language. The complexity of the job dictated that a higher level language be used. Unfortunately, some of the preprocessor functions required features not directly available in most higher level langauges. These features were principally dynamic storage allocation, manipulation of data at the bit level, character handling facilities, and enhanced file system support. Essentially two courses were possible. The preprocessor



<u>BLOCK 1 CONTENTS</u>

[HEADER] [DATA RECORD] [DATA RECORD]

HEADER:

[IREC] [L] [D] [S] [2 BYTE WHAT NOS.] [1 BYTE WHAT NOS.]

DATA RECORD:

[DATA] [TIME] [YEAR] [WHATC] [DATA VALUES] [STATUS VALUES]

Remaining blocks consist purely of data records as above.

Figure 4.4. Organizational Header Technique

could be coded primarily in COBOL or FORTRAN and the additional capabilities required provided by a library of assembly language subroutines. An alternative was to code the program entirely in PL/I. The latter option was selected for two reasons. First, it would allow the entire program to be coded in a high level language effecting a considerable time savings. Second, the syntax of PL/I was felt to permit the straight forward application of some structured programming techniques.

The program was built and tested in a top down, phase by phase manner. It consists of one large main program (ILPGENR) and three auxiliary routines (IMPTC1, IMPRGID, IMPCOM). The auxiliary routines provide some straight forward but frequently required functions, such as time conversion, page header generation and message printing. The individual phases of ILPGENR were sufficiently simple that informal pseudo-code (22) analysis was sufficient to plan the coding. ILPGENR is essentially a non-modular program, yet it adheres to a very basic structuring principle. Each phase possesses a single input, a single output, and a simple logical function. This non-modular structure additionally permits some economy in storage requirements. No attempt was made to rigorously avoid 'GO TO' type transfers, although they were avoided in a great many cases.

4.2.2.4 Performance

The final version of the preprocessor has performed very well. It represents a generalized solution to a common real-time processing problem, and as such its basic structure is useable in a wide variety of situations. The actual cost of processing is roughly $00.008 per tape data record. This does not include on-line and off-line storage charges. This high level of performance is attributed to repeated refinements of the phases. A considerable amount of this was done by examining the compiler output and reorganizing poorly implemented code at the source level.

4.2.3 The Preprocessing/Analysis Interface

As mentioned previously, this interface consists of two elements, an on-line disk file and a common access routine. Only the common access routine will be covered here as the nature of the on line file has been discussed in 4.2.2.2.

The objective here was to provide a software interface to the output of the preprocessor. This interface should appear logically constant, irrespective of the actual generation characteristics of the process file, and should be useable in all anticipated analysis modes. It would also be desirable for the interface to function as efficiently as possible. These goals were satisfied by a carefully coded FORTRAN routine, known as IMFGDT. This module manages all process file accesses. This routine can be utilized from another FORTRAN program, an assembly language program, or PL/I routines compiled by the optimizing compiler (24). When IMFGDT is first called, an initialization sequence is performed to obtain the header record from the process file and build a table of internal pointers. These pointers are subsequently used to decompose the process file data blocks. This entire process is completely transparent to the user, who can access the routine through a fixed calling sequence.

The implementation of this database technique has been an unqualified success. In addition to freeing the reduction programmers from cumbersome file manipulations, the routine has proved to be very efficient. One reason for this, is the dynamic allocation of records within a block. which precludes the presence of unused data space. Blocking factors between 90 and 150 are routinely achieved.

4.2.4 The Data Exception Control System

As the experiment progressed, it became clear that there were many situations in which the acquisition system would accept invalid data. These cases were invariably the result of a loss of experiment integrity due to some condition of which the computer was unaware. The need for some means of integrating human judgement into the experiment database was clear. Since the preprocessing and common access routine approach had been so successful in handling the primary data stream, a parallel approach was taken in this case. However, instead of processing data, this parallel system would process exceptions to the data. The system was named the Data Exception Control System (DECS).

The similarity between DECS and the data handling system is so strong, that little need be said here about its design philosophy. The system is covered in detail in the documentation (1). A few words about its integration into the rest of the processing picture in order.

The realization that some abnormal event has occurred must first take place. It may be apparent from an erroneous dial reading, or from a data summary provided by the aquisiion system. This realization may even be delayed until the interactive reduction of the data. In any event, the bad data has been incorporated in the normal experiment data flow, and the job of locating it and removing it may be formidable. Instead of attempting to excise this data (which constitutes a miniscule percentage of total storage), the DECS system provides for the creation of a parallel 'data-exception base'. This file, which is maintained on duplicate tape volumes, is a time sequential history of all known irregular conditions. As part of this history, information is present giving the affected channel, a brief textual explanation, and disposition information. The disposition information permits the observer of the irregularity to instruct the reduction software on how the situation is to be handled. The DECS utility program (ILPDECS) handles the maintenance of this tape, and all other files associated with the DECS system. Before this exception data may be used by a reduction task, ILPDECS processes it into a dynamically blocked, database type organization and stores it in an on line disk dataset. This dataset in turn may be read by a special access routine, which presents a fixed logical inteface and handles all file manipulations. An analysis task may then use the information returned by the access routine to validate process file data.

There are several important advantages to this type of mechanized exception system. First, as mentioned, it eliminates the need to edit a very large sequential data file and update it on multiple volumes. Second, exceptions can be added at any time, without interfering with the normal data flow. Finally, the DECS exception tape, when combined with the experiment database, provides a complete, and accurate, record of experiment activity.

4.2.5 Reduction Tasks

The final phase of the data handling procedure, consists essentially of using the process file to produce various displays and listings. As previously mentioned, certain immediate goals for this part of the system had been established. These were the capability to generate time history plots for all experiment parameters, and the capability tc produce these plots in hard copy or in an interactive graphics mode. These two specifications provide a convenient starting point for the segmentation of the entire reduction task.

Figure 4.5 diagrams the starting point for this part cf the software design. This figure is essentially a detailed expansion of Figure 4.1 from the fixed interface downward. All established design features are indicated. Three areas remain in which the design is incomplete. They are the design of the reduction tasks themselves, the design cf support code for the tasks, and the design of a user interface.

The design of the reductior tasks and the task support code is actually a single procedure. the method used is a top down 'exploding' type of design. Since the actual scientific analysis will be performed by this portion of the code,it was felt desirable to make this code as understandable to the user community as possible. Tcward this end, a completely mcdular scheme is followed, with nc more than a single logical function per module. The mcdules initially specified for support reflected the needs of the currently knewn tasks. These support needs were determined by an informal pseudo-code expression of the known reduction

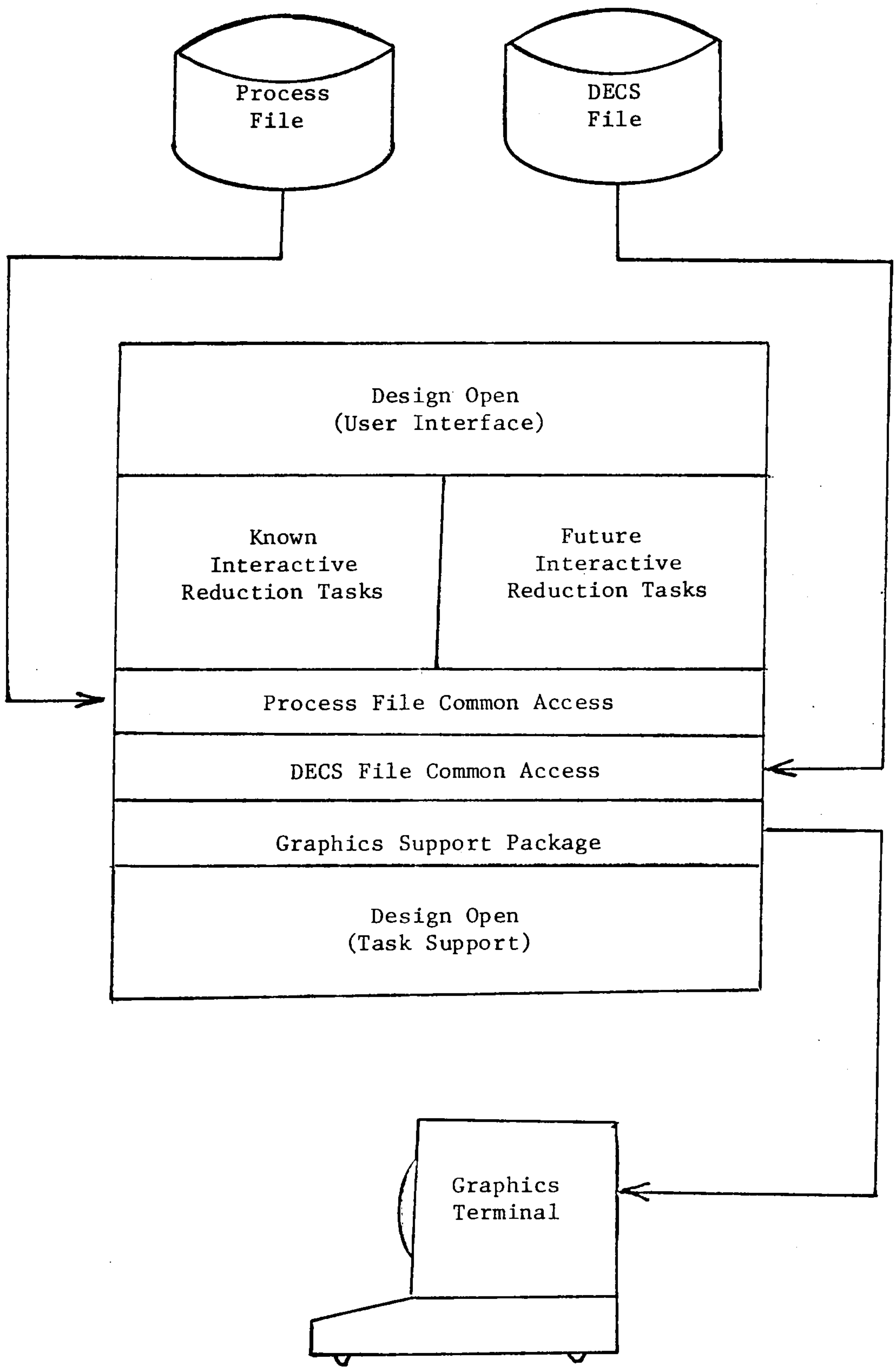


Figure 4.5. Reduction Task Expansion

tasks. A typical comparative pseudo-code analysis of a rain rate versus time plot and statistical plot is presented in Figure 4.6.

TYPICAL PSEUDO CODE ANALYSIS

Rainrate vs. Time

```
1)  Check Passed Parameters;

2)  Locate Range of P.F.;
        Locate First Record;
        Locate Last Record;
        End;

3)  Locate Data Extremes;
        Initialize Locations;
        Scan File;
        Save Extremes;
        End;

4)  Draw Axes and Plot I.D.;

5)  Plot Data;
        Scan File;
        Scale Data;
        Plot;
        End;

    End;
```

% Time Exceedance Attenuation

```
1)  Check Passed Parameters;

2)  Locate Range of P.F.;
        Locate First Record;
        Locate Last Record;
        End;

3)  Locate Data Extremes;
        Initialize Locations;
        Scan File;
        Save Extremes;
        End;

4)  Draw Axes and Plot I.F.;

5)  Perform Statistical Comp.;
        Initialize Bins;
        Scan File;
        Bin Data;
        End;

6)  Plot Data;
        Scan Bins;
        Scale Bin Contents;
        Plot;
        End;

    End;
```

Figure 4.6. Typical Pseudo Code Analysis

There is clearly a high degree of common function between these two analysis tasks. In fact this is the case for all the ONWARD analyses. Based on this type of comparison the following support code functions were initially defined. They are a generalized time computation routine (function A), a search routine (function B), a axis computing routine (function C) and an ID plotting routine (function D). There support modules were then subjected to analysis themselves, and in some cases further divisions were made. At the completion of this design step, the identity and function of every module in the system (from the reduction tasks downward) are known. In addition to breaking the problem down into easily coded units, this procedure also automatically defines the data and control interface between the modules (25).

At this point it was necesary to begin coding of the reduction tasks, so that badly needed report data could be obtained. FORTRAN was selected as source language since it was familiar to project personnel. The support modules were coded first and tested individually. Although this is contrary to the desirble top down coding procedure, it was felt that the interface design of the system was clear enough to prevent conflicts as the system was built up. Coding then proceeded concurrently on the analysis modules (only three were required at the time) and a module to control the selection of and parameters input to the desired analysis. This control module, or FORTRAN control program as it later was to be called, accepted a card image stream of sequential commands. Each command consisted of a numerical code (control code) and usually a numerical parameter. The control codes performed three possible functions. These were the setting of some numerical parameter, the setting of some execution option, or the initiation of the analysis. The required command streams were set up in files for debugging purposes. The desired plots were obtained with a total of two months development effort from reduction system conception.

Admittedly, the method established for interaction between the control program and the user was somewhat crude and inefficient. However, it was simple and quite general, and provided a basis for easy interface to the eventual interactive extension.

4.2.6 The Interactive Interface

The final enhancement that was planned for ONWARD was the implementation of a true interactive mode of operation. It was hoped this would permit the rapid review of large amounts of experiment data at an interactive graphics device. The graphics routine provided by the computing center were already arranged to suport the local graphics devices. The only task that remainded was to design and code an interctive front end for the system and complete the details of interface. The primary goal of any interactive computing system is, of course, ease of use. This desirable feature is the result of two things. First, the command language should be easy to use. Second, the interactive software should be able to recover from operator errors. The course taken to meet these goals is the subject of the following sections.

4.2.6.1 Defining the Command Language

The designer of a command language for a specific application environment has a considerably simpler job than the designer of such a language for, say, a general time sharing system. In the specific application case the language can be structured to fit the task. In fact, obvious logical similarity between the task and the language will aid in its correct use. The author has had exposure to a number of command language environments over the past few years and thus the language presented here embodies the best features of many sources. My intention here was to produce a language sufficiently powerful for the task, yet free of syntactic complexities, and clearly tailored to the logical function of the task.

The precise syntax of the ONWARD command language is treated in detail in the adjunct document (1). This definiition will not be repeated here. A few words are appropriate about the development process for the language.

Command languages are perhaps the most common mode of man/machine communication. Job control languages, time sharing languages and some application languages fall into this category. Without becoming immersed in the rigor of formal languages, two particular types of construction can be informally identified in command languages. These are parameter specification by position and parameter specification by keyword. Usually a command language utilizes some combination of the two. OS/360 Job Control Language is a particularly well known example. The relative merits of these two types of construction may be summarized as follows. A purely positional command language is somewhat difficult to use but is very easy to syntax check, since all parameters must appear in precise positions. A purely keyword command language is easy to use, though somewhat wordy, and is fairly easy to check for syntax . A mixture of the two usually gives concise expression, but is syntactically complicated and difficult to check for syntax . Those having experience with JCL error diagnosis on OS/360 will confirm this.

In view of these consideration, the ONWARD command language is primarily keyword oriented. Positional parameters are used to a very limited extent. A typical ONWARD command line has the following format.

BASIC-COMMAND KEYWORD(PARAMETER) KEYWORD(PARAMETER)

The basic command defines the basic function of the command line. It is a positional parameter and must appear as the first word of every command line. The keywords and their parameters are used to assign values or set options. The keywords and their parameters may appear in any order. Keywords which accept multiple parameters are permitted and the parameters may appear in any order. All command and keyword strings may be abbreviated to any extent, provided that the first letter is correct and the remaining characters are unique.

4.2.6.2 Implementation Considerations

In the interpretation of any command language, the first step is to dissect the command line into its logical components. These components must then be examined for a correct syntactic relationship. If an error is found, it must either be corrected by the user or the command is aborted. Finally, the information obtained from the command line is used to manipulate the application in the specifid manner.

The design of software to accomplish the above must consider several points. First, the application is completely character oriented. Second, an interface to the FORTRAN software must be available. Finally, the ability to handle a diverse range of error conditions would be helpful. These capabilities may be obtained through three different routes. The entire system could be coded in assembly language, but the development time would be excessive. A great deal of the code could be written in FORTRAN, and assembly language routines used for the special functins. The most attractive alternative was, once again, PL/I, which would allow all coding to occur in a high level language. The latter course was selected.

There is one final point to be considered in the interactive design. Even if a command line is syntactically correct, it may not contain enough information to successfully execute the specified function. A solution to this is certainly of no great difficulty, but the method of implementation can reatly affect the flexibility of the resulting software. If the required parameter checking information is placed in the application code (the interface module to be exact), then this software becomes a functionally complete package and can be utilized in a stand alone fashion if desired.

4.2.6.3 The FORTRAN Interface Module

As discussed in Section 4.2.5.1 a driver module to manipulate the analysis routine under the control of a card image input stream was written. The initial intent of this routine was simply as an interim measure to allow some badly needed analyses to be completed. The concept of communicating with the FORTRAN software on a command plus parameter basis was, however, attractive as a final design characteristic for the following reasons. First, the need for a large common external storage area between the interactive modules and the FORTRAN modules was eliminated. This is especially undesirable in a PL/I-FORTRAN environment, since these languages each access storage in different manners. Second, the interface can be performed through a set of simply defined entry points, using scalar parameters whose interpretation in both languages is identical. Finally, the extension of this type of interface to batch use would be a trivial task. Only a single additional module to read input from cards and direct it through the appropriate entry point would be necessary.

Based on these considerations, the interface module, or FORTRAN control program (IMF099) was coded. The temporary driver routine was used as a basis. Tables were added to provide appropriate data interpretation, parameter checking and control transfers. Also performed here are all initialization required by the FORTRAN code. This was arranged as executable code for two reasons. First, it permits the start up environment to be reestablished at any time, and second, it allows a clear arrangement of the tables. A detailed description of this module's operation is given in Reference 1.

4.2.6.4 The Interactive Modules

At this stage of the design, the command language and the interface to the application are well defined. The purpose of the interactive software is to close the connection between the two. The source language has been established as PL/I. The design and coding of the software can now proceed.

Six specific responsibilities have been identified with the interactive software. They are inputing a line, scanning and parsing it, checking its syntax, attempting error correction, checking for required parameters and manipulating the application. Of these , the scanning of the command line was attacked first. It was desired that a general routine be developed for this purpose. This was satisfied by a short PL/I routine (IMPCSP) which is capable of extracting the significant elements from a command line of the defined format (see 4.2.6.1). IMPCSP is capable of recognizing keywords and their abbreviations, and also provides automatic numeric conversion of eligible fields. It is random access in operation and can retrieve elements from the command line in any order. The remaining interactive tasks were concentrated in another module (ILPPTG). This module runs as the main program when CNWARD is used in interactive mode. The details of the code are discussed in Ref. 1.

Chapter 5

Conclusions

This paper has presented the design aspects of a comprehensive data reduction system for millimeter wave experiments. The system has been entirely operational for approximately one year at this writing and has proven its utility in both interactive and batch operation. The application of modern database techniques to this task has resulted in a basic system configuration suitable for a wide variety of real-time data reduction situations. The value of the structured programming approach has also been demonstrated in the implementation of this moderately complex system by a single programmer.

In retrospect , the method of applying calibration curves to the experiment data is a potential area for improvement. A preferable procedure might be the establishment of a database of calibation curves which could be updated at will. This database would include all required calibations and could be applied to the data at preprocessing time. This would prevent the permanent application of erroneous calibration curves and provide an historical trace of system calibration changes.

Bibliography

1. Kauffman, S. R., 'ONWARD System Documentation', Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, Va., March, 1978.

2. IBM Virtual Machine/370:System Programmers Guide, Order No. GC20-1807, International Business Machines Inc., White Plains, New York, March 1976.

3. Yourdon, Edward, Techniques of Program Structure and Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

4. Fox, Jerome, Ed., Proceedings of the Symposium on Computer Software Engineering, Polytechnic Press, Brooklyn, N.Y., 1976.

5. Katzon, Harry, Jr., Advanced Programming, Von Nostrand Reinhold Company, New York, 1970.

6. Hopcraft, John E., and Jeffrey D. Ullman, Formal Languages and their Relation to Automata,

Addison-Wesley Publishing Co. Inc., Reading, Mass., 1969.

7. Bostian, C. W., S. B. Holt, S. R. Kauffman, E. A. Manus, R. E. Marshall, W. L. Stutzman, and P. H. Wiley, 'Rain Depolarization and Attenuation Measurements at 11.7, 19.04, and 28.56 GHz: A Description of the Experiment and Some Preliminary 11.7 GHz Results,' Annales des Telecommunications, accepted for publication in 1978.

8. Kauffman, S. R. and W. L. Stutzman, 'Real Time Control of Satellite Tracking Station Operation for Millimeter Wave Experiments and Associated Data Processing', Proceedings of 1977 IEEE Southeastern Convention, Williamsburg, Virginia, pp. 248-251, April 4-6, 1977.

9. Bostian, C. W., S. B. Holt, S. R. Kauffman, E. A. Manus, R. E. Marshall, W. L. Stutzman, and P. H. Wiley, 'Rain Depolarization and Attenuation Measurements at 11.7, 19.04, and 28.56 GHz: A Description of the Experiment and Some Preliminary Results', Proceedings of the URSI Commission F Open Symposium on Propagation in Non-Ionized Media, LaBaule, France, Spring 1977.

10. Wiley, P. H., W. L. Stutzman, E. A. Manus, R. E. Marshall, S. B. Holt, and S. R. Kauffman, 'Rain Depolrization and Attenuation at 11.7 GHz on the CTS Spacecraft Downlink', 1976 Meeting of the International Union of Radio Science, Amherst, Mass., October 11-15, 1976.

11. Bostian, C. W., W. L. Stutzman, E. A. Manus, P. H. Wiley, R. E. Marshall, S. R. Kauffman, W. P. Overstreet, and R. E. Persinger, 'Current Results of 11, 19, and 28 GHz Sattelite Attenuation and Depolarization Measurements,' National Radio Science Meeting, Boulder, Colorad, January 9-13, 1978.

12. Bostian, C. W., E. A. Manus, R. E. Marshall, H. N. Pendrak, W. L. Stutzman, P. H. Wiley, and S. R. Kauffman, 'Quarterly Technical Progress Reports I, II, III, and IV on a 20 GHz Depolarization Experiment Using the ATS-6 Satellite', Contract NAS5-21984, I-June 10, 1974; II-October 7, 1974, III-December 31, 1974; IV-April 15, 1975.

13. Bostian, C. W., E. A. Manus, R. E. Marshall, H. N. Pendrak, W. L. Stutzman, P. H. Wiley, and S. R. Kauffman, 'Final Report on a 20 GHz Depolarization

Experiment Using the ATS-6 Satellite,' Contract NAS5-21984, September 10, 1975.

14. Bostian, C. W., S. B. Holt, S. R. Kauffman, E. A. Manus, R. E. Marshall, W. L. Stutzman, and P. H. Wiley, 'A Depolarization and Attenuation Experiment Using the CTS Satellite', Final Report (on first year), NASA Godard Contract NAS5-22577, November 18, 1976.

15. Bostian, C. W., S. B. Holt, S. R. Kauffman, E. A. Manus, R. E. Marshall, W. P. Overstreet, R. F. Persinger, W. L. Stutzman, and P. H. Wiley, 'A Depolarization and Attenuation Experiment Using the COMSTAR and CTS Satellites,' Quarterly Technical Progress Reports I (Dec. 22, 1976), II (June 15, 1977), and III (July 19, 1977) for NASA Contract NAS5-22577.

16. Bostian, C. W., S. B. Holt, S. R. Kauffman, E. A. Manus, R. E. Marshall, W. L. Stutzman, P. H. Wiley, 'A Depolarization and Attenuation Experiment Using the CTS Satellite', Quarterly Technical Progress Reports I (Feb. 28, 1977), II (March 31, 1977), and III (June 30, 1977) for NASA Contract NAS5-22577.

17. RT-11 System Reference Manual , DC-11-RUGA-C-D, Digital Equipment Corporation, Maynard, Mass., 1976.

18. RT-11 V02C System Release Notes , DEC-11-ORNRA-A-D, Digital Equipment Corporation, Maynard, Mass., 1976.

19. RT-11 Fortran Compiler and Object Time System User's Manual , DEC-11-LRFDA-A-D, Digital Equipment Corporation, Maynard, Mass., 1974.

20. Cohen, Leo, J., Data Base Management Systems , Q.E.D. Information Sciences, Inc., Wellesly, Mass., 1976.

21. IBM Virtual Machine Facility/370: CMS Command and Macro Reference , Order No. GC20-1818-0, International Business Machines, Inc., White Plains, New York, March 1976.

22. McGowan, Clement, 'Structured Programming : A Review of Some Practical Concepts', Computer Magazine, IEEE Computer Society, Long Beach, California, June 1975.

23. Buelum, Barry, 'Structured Programming: A Quantitative Assessment', Computer Magazine, IEEE Computer Society, Long Beach, California, June 1975.

24. OS PL/I Checkout and Optimizing Compiler: Language Reference Manual, Order GC33-0009-4, International Business Machines, Inc., White Plains, N.Y., 1976.

25. Fairley, R. E., 'Modern Software Techniques', Ref. 4.

26. Yeh, Raymond T., 'Software Engineering', Computer Magazine, May 1975.

27. Brooks, Frederick, P., Jr., The Mythical Man Month, Addison-Wesley Publishing Co., Reading, Mass., 1975.

28. Palmer, Jan R., Data Base Systems: A Practical Reference, Q.E.D. Information Sciences Inc., Wellsley, Mass., June 1975.

The vita has been removed from the scanned document

AN INTERACTIVE DATA REDUCTION SYSTEM

FOR REAL-TIME MILLIMETER WAVE EXPERIMENTS

by

Steve R. Kauffman

(Abstract)

The nature of data acquisition in a realtime millimeter wave experiment is discussed. Goals for a data reduction system are presented and the available processing environment analyzed. Some currently popular software design methodologies are then reviewed and selectively applied to a complete system design based on modern database methods.