ADVANCED HYBRID COMPUTER SYSTEMS. SOFTWARE TECHNOLOGY

Harriet B. Rigas

Washington State University

Prepared for:

Army Materiel Command

June 1974

SOFTWARE TECHNOLOGY

ADVANCED HYBRID COMPUTER SYSTEMS

JUNE 1974

PREPARED BY:
DR. HARRIET B. RIGAS
WASHINGTON STATE UNIVERSITY

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U. S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

HQ. US ARMY MATERIEL COMMAND
ALEXANDRIA, VA 22333

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: Advanced Hybrid Computer Systems Software Technology	5. TYPE OF REPORT & PERIOD COVERED: Final June 1974 / 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Dr. Harriet B. Rigas	8. CONTRACT OR GRANT NUMBER(s): DAAG-39-74-C-0140
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Washington State University, Pullman, Wash 99163	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 1E865803M730
11. CONTROLLING OFFICE NAME AND ADDRESS: Aldric Saucier, Hqts, US Army Materiel Command, 5001 Eisenhower Avenue, Alexandria, VA 22333	12. REPORT DATE: 31 May 1974 / 13. NUMBER OF PAGES: 31
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Same	15. SECURITY CLASS. *(of this report)*: Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Prepared with the Advanced Hybrid Computer Systems Working Group

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Hybrid Computer
Compiler Development
Simulation
Simulators
Software Technology
Automatic Patching
Programming Languages
XHK
Analog Software
Digital Software
Hybrid Software

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The Software Technology Final Report evaluates advances made in Advanced Hybrid Computer System software technology. The report describes what automatic patching software is available as well as which analog/hybrid programming languages would be most feasible for the Advanced Hybrid Computer System. Compiler development, it is suggested, should be limited to refinement of existing simulators since a great deal of work has already been spent on compiler software. The problem of how software would interface with the hybrid system is also presented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

SOFTWARE

TECHNOLOGY

A D V A N C E D

H Y B R I D

C O M P U T E R

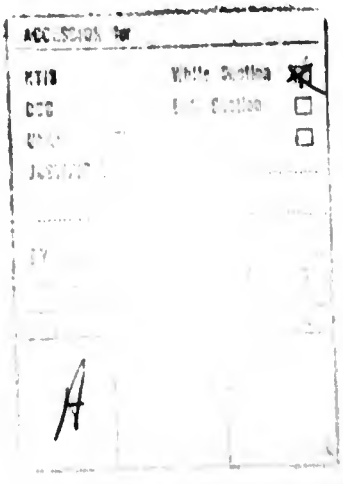
S Y S T E M S

JUNE 1974

PREPARED BY:

DR. HARRIET B. RIGAS
WASHINGTON STATE UNIVERSITY
ARMY CONTRACT NO. DAAG 39-74-C-0140

HQ, US ARMY MATERIEL COMMAND
ALEXANDRIA, VA 22333



HEADQUARTERS, US ARMY MATERIEL COMMAND

SOFTWARE TECHNOLOGY REPORT

PREPARED FOR: MR. ALDRIC SAUCIER,
Project Leader for Advanced
Hybrid Computer System

ALEXANDRIA, VIRGINIA 22333

JUNE 1974

ADVANCED HYBRID COMPUTER SYSTEM
SOFTWARE TECHNOLOGY REPORT

ABSTRACT

This Software Technology Final Report evaluates advances made in Advanced Hybrid Computer System software technology. The report describes what automatic patching software is available as well as which analog/hybrid programming languages would be most feasible for the Advanced Hybrid Computer System. Compiler development, it is suggested, should be limited to refinement of existing simulators since a great deal of work has already been spent on compiler software. The problem of how software would interface with the hybrid system is also presented.

COLLEGE OF ENGINEERING
RESEARCH DIVISION

Research Report No. 74/20-38

ADVANCE HYBRID COMPUTER SYSTEM
SOFTWARE TECHNOLOGY EVALUATION

May 31, 1974

Project No. 12F-3820-1311

Contract No.

Sponsor Project No.

Sponsor Contract No. DAAG 39-74-C-0140

Department of the Army

Harriett B. Rigas
Principal Investigator

G. L. Hower
Branch Manager

Pullman, Washington

CONTENTS

FIGURES

TABLES

INTRODUCTION 1

CHAPTER I - LANGUAGE SPECIFICATION 2

A. MIMIC 5
B. CSMP/360 and CSMP - III 7
C. CSSL - III, RSSL 7
D. SL-1 8
E. DYNAMO 8

I.2 BLOCK-ORIENTED ANALOG COMPUTER SIMULATORS 8

I.3 CONTINUOUS/DISCRETE LANGUAGES 10

CHAPTER II - SOFTWARE FOR AUTOMATIC PATCHING 11

II. 1 HOI 11

II. 2 APSE 11

II. 3 HAL 13

II. 4 ACTRAN, HIFIPS, HCSI 13

II. 5 PATCH 13

CHAPTER III - SOFTWARE RECOMMENDATION FOR THE AHCS 15

III. 1 COMPUTER OR COMPUTERS FOR COMPILATION 15

III. 2 MULTI USER ENVIRONMENT 16

III. 3 COMPILER DEVELOPMENT 19

III. 4 CHOICE OF COMPILER LANGUAGE 21

III. 5 PORTABILITY OF COMPILER 21

CHAPTER IV - CONCLUSIONS

FIGURES

1. FLOW DIAGRAM TO ILLUSTRATE RELATIONSHIPS BETWEEN OS/360, DEJCSMP, AND THE FOUR PHASES OF CSMP 4

2. MULTI USER AHCS CONFIGURATION I 18

3. MULTI USER AHCS CONFIGURATION II 20

TABLES

1. CURRENT EQUATION-ORIENTED CONTINUOUS SIMULATION LANGUAGES 6

2. BLOCK- ORIENTED CONTINUOUS SIMULATION LANGUAGES 9

3. SOFTWARE FOR PATCHING SYSTEMS 12

INTRODUCTION

For an automatically "programmed" hybrid computer it is necessary to provide a high level language in which a user may describe his problem. The word "programmed" is preferred to patched since one must think of the resulting computer realization as a complete program consisting of the patched analog computer subsystem and executable code for the digital subsystem which together provide an executable hybrid computer program.

The language used for specifying the problem should provide sufficient power to be applicable to a great number of different problems and be flexible enough to be extendable for other problems. These extensions should be simple to make without an extensive knowledge of the internal structure of the compiler.

This report surveys the higher level languages presently implemented as simulators on commercially available computers. Examples of automatic patching programs are given to show how these relate to the simulators and extensions and modifications of these systems are discussed.

A class of simulators using lower level languages are described and suggestions are made on how these might be useful.

One of the most important aspects of any software for a hybrid computer compiler is the operating system under which the compiler is run. This report points out the importance of considering the relationship between the operating system and the compiler.

Finally, suggestions are made for system specifications to (1) allow portability of user programs between various installations; (2) allow use of the AHCS in a multi-user environment.

CHAPTER I

Language Specification

Efforts to develop digital computer languages to duplicate the functions of the analog computer have been recorded for twenty years. Rather than give a historical treatise and these developments, we will briefly document here the events leading to the languages currently implemented and commercially available.

Continuous simulation languages can be broken into two categories:

1. Equation-oriented
2. Block-oriented

A proliferation of block-oriented languages followed Selfridge's[1] work at the same time that procedural languages such as Fortran were being developed for general purpose computing. In 1959 Stein, Rose, and Parker[2] developed ASTRAL for the IBM 704 which compiled the user program into FORTRAN, sorted simulation statements and allowed limited use of FORTRAN arithmetic statements. This concept provided a basis for current equation-oriented simulation languages.

The block-oriented MIDAS[3] and the equation-oriented MIMIC[4] incorporated many of these features for the 7090 series of machines with MIMIC translating directly to executable machine code.

DSL/90[5] was developed independently at IBM for the IBM 7090 and allowed extended use of FORTRAN thus allowing problems which more nearly represented hybrid computer programs. When IBM updated to the 360's and 1130's computers, it supplied two new continuous simulators, CSMP/360 and CSMP/1130, which were based on DSL. CSMP/1130 was block-oriented and CSMP/360 was equation-oriented.

Because these languages were proliferating, a simulation software committee was formed by the SCI in 1965 to propose a standard language for continuous simulations and the result was the specification of CSSL[8].

The resulting language was based on MIMIC but heavily influenced by DSL/90 and FORTRAN. The CSSL committee did not attempt to recommend implementation methods.

Since 1967 the equation-oriented continuous simulation languages have been implemented for the most large computer systems either by the manufacturer or a user. The implementations vary somewhat but are quite similar in philosophy. Figure 1 demonstrates the steps taken by CSMP/360 in translating and executing a simulation. CSSL-III and SL-1 have similar implementation on the CDC 6000 series and larger Sigma computers respectively.

The program runs under OS/360 and performs the following steps.

1. Sorts source statements unless statements are specified NOSORT sections.
2. Translates source program to a FORTRAN program.
3. Invokes system FORTRAN compiler.
4. Links compiled program with CSMP library routines and other invoked system routines into a load module.
5. Executes linked program under interpretive control for timing, etc.

This process is highly dependent on the operating system. Clearly under an operating system supporting only batch type FORTRAN the simulation must be executed in batch mode. Even with the use of overlays these programs can be large. CSMP takes a minimum of 102K bytes of core.

One of the points that will be emphasized later in this report is that the language itself does not imply an implementation and a great deal of flexibility is provided in a digital simulator to allow for a choice of integration methods, and sophisticated output routines which might not be necessary if the language is to be used for hybrid compilation.

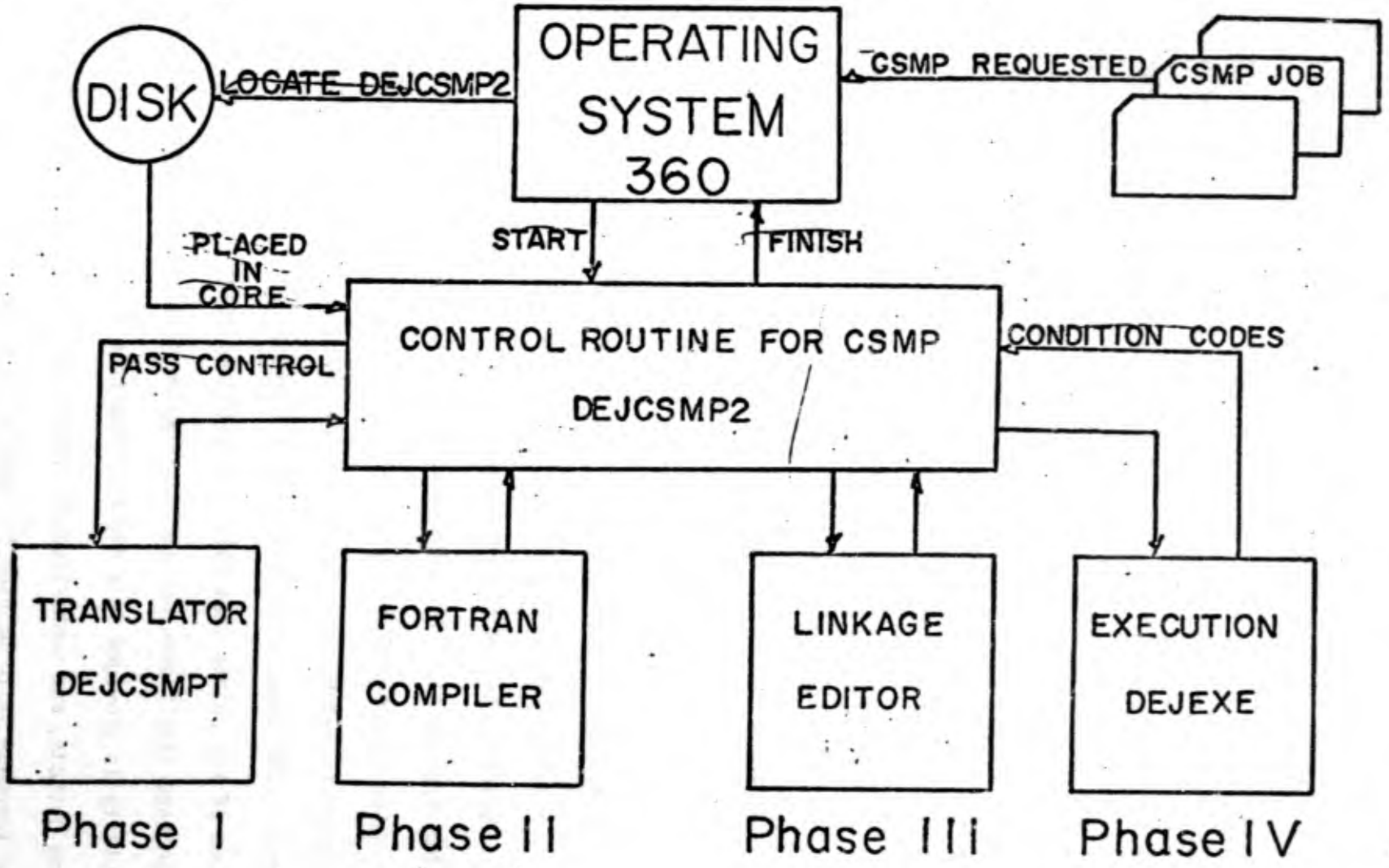


Fig. 1.--Flow Diagram to Illustrate Relationships between OS/360, DEJCSMP2, and the Four Phases of CSMP

Table I shows the currently implemented versions of the languages and the machines on which they are implemented. We will give a brief description of these languages.

(a) MIMIC has been implemented on the CDC 3000 and 6000 series and the Univac 1108. It is still supported on CDC 6000 series. Typical MIMIC statements must be writen in a rigid format with field specified as follows:

Columns	1	2	10	19	73
	Comment Indicator Unless $	Logic Control Variable Field	Result Name Field	Expression Field	Identification

The logic control variations cause the statement which follows to be executed if it is TRUE and ignored when it is FALSE. MIMIC contains standard simulation functions such as INTEGRATOR, INTEGRATOR LIMITER, DERIVATIVE, FIRST ORDER LAG, TRACK AND STORE, FLIP-FLOP, ZERO ORDER HOLD and FUNCTION GENERATION of up to three variables as well as typical arithmetic and trigonometric functions such as ADD, COSINE, LOG and EXPONENTIAL.
The user may write FORTRAN subroutines which may be coupled to the MIMIC program under limited conditions. The MIMIC processor can detect a limited number of errors including algebraic loops. MIMIC does not interact as completely with the operating systems as the implementations of CSMP and CSSL which follow.

TABLE I

CURRENT EQUATION-ORIENTED
CONTINUOUS SIMULATION LANGUAGES

Simulator	Computer
MIMIC	UNIVAC 1108 CDC 6000 series
CSMP	IBM 360/370 under OS or OS compatible systems
CSSL	CDC 6000 UNIVAC 1108 PDP 11/45
SL-1	SIGMA 5 SIGMA 7

(b) CSMP/360 and CSMP-III[6,7]

CSMP/360 and the more recent CSMP-III interact with the operating system as outlined earlier and therefore allow extensive use of FORTRAN including the complete library facilities. CSMP has an expanded set of operator and MACRO from the MIMIC set. User has control of the SORT and NOSORT sections and may define MACROs as well as making unrestricted use of FORTRAN FUNCTION and SUBROUTINE statements. Although the LCV of MIMIC is not defined for CSMP, the language supports MODE CONTROLLED INTEGRATORS, FUNCTION SWITCHES and all IF statements in FORTRAN providing comparable programmed control. The latest version CSMP-III for 360's and 370's has expanded MACROs and double precision arithmetic. CSMP-III also supports the IBM 2250 graphics display and allows on-line access to the input program, the data set and the translate, compile, link-edit and execution phases of the simulation. Diagnostics for both versions are extensive and some debug features are available.

(c) CSSL-III, RSSL

Since its inception, CSSL has been extensively implemented. One version, CSSL-III, was developed by Programming Science, Inc. and is marketed by CDC. Another version, RSSL, was implemented by Raytheon, Bedford, Massachusetts. A subset DARE-IIIB[11] was developed at the University of Arizona.

Versions have also been developed to some level of completion for a variety of machines including PDP 11/45 in conjunction with the automatic patching systems in Europe and for the UNIVAC 1108 at Carnegie-Mellon University. These latter systems will be discussed in the next chapter.

As stated earlier, CSSL is based on MIMIC and does retain the logic control variable. However, CSSL is implemented to interact through the system

with the FORTRAN compiler and link-editor to take advantages of the FORTRAN and system library.

In many ways, CSSL-III and RSSL are similar to CSMP-III with extensive diagnostics and debugging capability. In contrast to MIMIC, it has free form coding, an extensive operator set and MACRO capability.

(d) SL-1[12]

SL-1 was developed in 1970 for Xerox Data Corporation and runs on the Sigma 5 and Sigma 7. It is a super set of the language specified in the CSSL report. In addition to allowing standard FORTRAN to be included in user programs, it allows assembly language (SYMBOL) statements to be included as well. This can be particularly useful to an experienced programmer for designing programs which are time-sensitive.

SL-1 allows multiple independent variables which might have application in more sophisticated hybrid programs. SL-1 is extremely flexible and versatile.

(e) DYNAMO[13]

Dynamo was developed for the IBM 360/370 series independently from the CSSL-CSMP-SL1 efforts and is substantially different. In many ways it is more cumbersome to use than the others requiring a rigid format and the specification of equation type to facilitate sorting. As a simulator, it is limited in using only Euler Integration. On the positive side, there is an interactive version of Dynamo running under CP/CMS 360.

BLOCK-ORIENTED ANALOG COMPUTER SIMULATORS

With the advent of fast, inexpensive mini-computers there has been a growth of block-oriented simulators which closely resemble the functions of the

TABLE II

BLOCK-ORIENTED
CONTINUOUS SIMULATION LANGUAGES

Language	Machine
CSMP	IBM 1130 EAI 690
DARE II	PDP 9
ISL	PDP 8-E PDP 15 EAI PACER/640
SIMEX	PDP 9

analog computer. These are summarized in Table II.

These languages are generally implemented in assembly or machine code and therefore limit use of interspersed FORTRAN. The input code is interpreted directly to linked arrays from which the simulation is executed. This approach does allow a high degree of interactive control since control and modification can be interpreted at execution time.

Although these are not sufficiently comprehensive for hybrid compilation, their array structures do contain the analog computer interconnection information. For this reason they could provide an excellent target language for the compiler. By doing this the interactive capability of the language could be used to make the on-line modifications to the analog computer.

CONTINUOUS/DISCRETE LANGUAGES

A class of languages somewhat related to those discussed here are those which combine the aspects of CSSL or CSMP with those of discrete languages such as SIMSCRIPT, SIMULA, and GPSS.

Discrete languages have been used extensively for problems in which there are statistical variations in arrival times such as traffic problems. By combining the strengths of continuous and discrete simulation languages the class of problems which can be handled is greatly increased. A point which should be considered is that the class of problem which can be described by such languages is naturally suited for hybrid computation and this represents a relatively untapped hybrid user base.

Three systems under development or in use are GASP IV[20] at Purdue, GSL[19] at Case-Western and PROSE[21] from Solveware in San Pedro, California.

CHAPTER II

Software for Automatic Patching

As in the case of the simulation languages, there were many developments in the field of software for the generation of patching information for an analog computer. Some early efforts date back to 1961, but again rather than give a historical review of this work we will give a summary of the approaches to automatic patching which appear most directly applicable to AHCS.

The greatest activity in the development of automatic patching software has been in Europe and this work is documented in Col. Enslow's report.[22]

The systems which are currently available or under development are summarized in Table III along with the host computer on which they are executed and the target computer.

(a) HOI[23]

Perhaps HOI should be discussed first since it is somewhat different from the other systems to be discussed here. HOI is an interpreter operating directly on source statements and therefore is highly interactive but slow. It is primarily designed for use as a checkout system and, as such, does not handle the automatic programming problem.

Although it is possible to link to FORTRAN subroutine it is not particularly suited to be a basis for a Hybrid Compiler directly. It is not unreasonable as a target language for the checkout phase and could conceivably be extended to provide facility for interactive modification of an automatically patched analog subsystem.

(b) APSE[24]

APSE is a language which prepares patching information from the description of the integral differential equations. APSE must be supplied with maximum

TABLE III

SOFTWARE FOR PATCHING SYSTEMS

		HOST COMPUTER	TARGET COMPUTER
APSE	CSMP (Like)	IBM 360/370 UNIVAC 1108 CDC 6000	231R 681 8800
HOI	INTERPRETER (Special)	PACER	680 7800 8800
ACTRAN	CSMP	IBM 370	8944
HIFIPS & HL	Superset CSSL, SL-1	XDS CDC 3000 PDP-10	AD4/PDP-11 AD4/Sigma 3
HAL	Subset CSSL	UNIVAC 1108	680
PATCH	CSMP	IBM 360	680/640

values in order to do the scaling. It is limited to the continuous portion of the analog subsystem.

(c) HAL[28]

HAL is a hybrid computer programming system developed at Carnegie-Mellon by Mark Franklin under Jon Strauss. While not completely developed, HAL demonstrates several interesting points.

The high level language is a limited subset of CSSL called Hybrid Source Language HSL. In addition, the programmer may use HAL, Hybrid Assembly Language and HML, Hybrid Machine Language to exercise more control of his patching. The target is an EAI 680 with logic but the system, even with its MACRO capability, does not appear to be easily extendable to a full hybrid compiler.

(d) ACTRAN, HIFIPS, HCSl[25, 26, 27]

Basically, ACTRAN and HIFIPS are similar systems developed independently in England and the Netherlands, respectively. Recently these efforts have combined as HCSl. The final product will be a true hybrid compiler with good diagnostics. Although there is some discussion of providing a more extensive language than CSSL-III or CSMP-III, it is not clear exactly what the source language will be. Another factor which tends to slow down the progress is the determination to develop a system to run on several host computers and compile to a variety of different systems.

Basically the system is an extended simulator which provides scaling data to a patching/compiling program. Static testing and online editing capabilities will make a reasonably complete hybrid compiling system.

(e) PATCH[29, 30]

PATCH was developed at Washington State University to demonstrate the extendibility of a high level simulator such as CSMP to hybrid compilation.

There are two steps to the development of PATCH.

1. Modifications to the existing simulator to pass the scaling and structure information to the hybrid compiler.
2. Design of the compiler program.

At present the compiler can patch all analog and logic components of the EAI 680 from a subset of CSMP/360. The nature of the CSMP implementation allowed addition of operators which resembled 680 components more closely. The compiler output is a user document from which the patching pot-setting and static testing may be done directly.

The output arrays are complete and can be interfaced through an assignment module to the patching hardware. Generation of load modules for the EAI 640 from the FORTRAN statements of the user's CSMP program has been demonstrated.

CHAPTER III

Software Recommendation for the AHCS

It is difficult if not impossible to make a firm recommendation for a software system for AHCS. Rather, we will try to summarize the factors influencing the choice of a software system and recommend approaches to specifying or accepting specifications for such a system.

I. COMPUTER OR COMPUTERS FOR COMPILATION

One of the first considerations in the development of a compiler is the choice of host computer system. If the compilation is to be executed after a simulation run in order to obtain extreme values, one may look to existing continuous simulators for some insight to the complexity required of the computer system.

A full blown simulator such as CSSL-III or CSMP takes a minimum of 100K bytes with overlays. Some of the refinements, such as choice of integrator package, could be omitted, but it is not clear that a substantial savings would be realized.

As is seen in Figure 1, such a simulator has a high degree of interaction with the operating system. Control must be returned to the processor control after compilation and link-edit phases and the operating system must be able to support this level of activity. It should be pointed out that the disc operating system, DOS, for the smaller IBM 360 models, i.e. Models 40 and 44 cannot support this level of interaction. Other factors which influence the choice of computer are the actual system configuration selected. Several approaches have been recommended, but the vendor reports are not available so that these recommendations are not directed at a specific vendor.

Basically, one might envision a system containing a reasonably large computer to perform the compilation. This computer would then load a small processor which would control the actual switching activities on the analog subsystem and provide run time monitoring. The digital subsystem of the hybrid computer could be either of these processors or an entirely separate one. In any case, it must receive the compiled digital program for the hybrid problem. Static testing could be performed by the set-up processor or the digital subsystem processor.

II. MULTI USER ENVIRONMENT

Although the delivered prototype may support only one hybrid user at a time, the system configuration should not preclude multiple users in some sense.

Some possible operations are:

1. Background-foreground type of operation.
2. Multi-hybrid computer consoles with a single compiler processor.
3. Sharing of hybrid computer facilities on a component requirement basis.

Other configurations can be envisioned for a particular laboratory environment.

(1) Background-Foreground

Background-Foreground mode would be feasible in a configuration in which a digital computer served both as the compiling device and the digital subsystem of the hybrid computer performing time critical tasks. In this configuration the digital computer must be of sufficient size and speed to support the time-critical hybrid code and the largest overlay of the compiler. A nice feature in the operating system would be a spooling system to improve

batch throughput of compilations and to control both compile and compile-go operation.

(2) Multi Hybrid Computers.

The operation here would be somewhat similar to the foreground-background system described. In this case it would be necessary (except for unusual circumstances) to separate the compiling processor from the digital subsystems of the various hybrid computers. However, one might combine the set-up and monitor functions with the time critical computations on a single machine.

One possible configuration would be to run the compiler on a central general purpose computer with remote terminal capability. This computer would load the appropriate set-up processor and digital subsystem and then turn control over to the set-up computer which would perform hardware diagnostics, static testing, and initiate execution. During execution this processor could monitor analog computer components and even perform automatic rescaling. Upon completion of a hybrid computation the central computer could be signaled and supplied with results or any information about abnormal termination.

A further refinement would be to allow the user to interact and modify the code. Should the compiler be interactive, this capability might be sufficient. In the case of a batch compiler there are two possibilities.

(1) To recompile the whole program and reload the setup processor.

(2) Recognize minor modifications equivalent to moving a few wires and modify the set-up processor arrays directly.

The first alternative is costly and time-consuming but would be indicated for massive editing.

The second alternative would place additional requirements on the operating system. In the simpler configuration the central computer is not

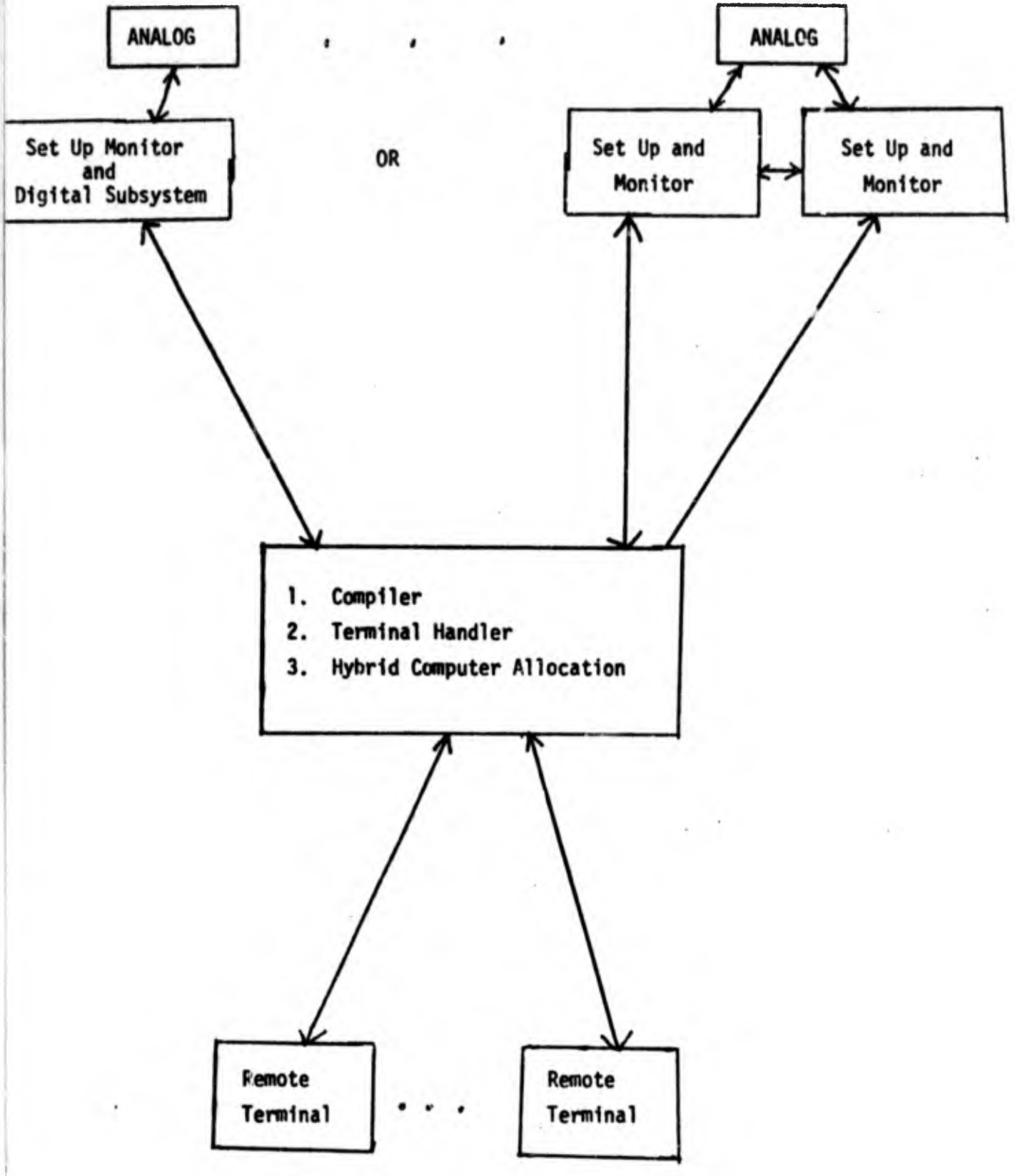


Fig. 2 Multi-User AHCS Configuration I

substantially different from the large processor system found in many laboratories with larger models of IBM 360/370, CDC 6000, or Sigma 5's or 7's. The compiler could be developed as an adjunct to the existing continuous simulator.

The drawbacks are all those found on systems of this type. Namely, relatively slow remote access or expensive fast access and limited remote graphics capability.

A second possibility which is somewhat similar but would provide additional flexibility is now described. A smaller processor (which might be combined with a system set-up process) could become the remote terminal handler. It could then dispatch compilations and output results and other run time diagnostics to the remote user. This operating system could be developed independently of the compiler development. Such an approach provides a great deal of flexibility and allows for future expansions and sophistications.

(3) Component Sharing of a Single Hybrid Computer.

In the case where a large number of problems each using a small complement of system resources are awaiting execution, it might be desirable to share components among them. The second configuration described in section (2) immediately preceding could handle this problem.

The communications processor could receive from the compiler an equipment summary and schedule analog components according to some algorithm based on time and equipment.

III. COMPILER DEVELOPMENT

Perhaps the simplest and fastest approach to implementing a full blown hybrid compiler is to append it to an existing simulator. In this case one can take advantage of the existing translator phase and all the array structures as well as having a direct way of obtaining extreme values for scaling and a run for a dynamic check.

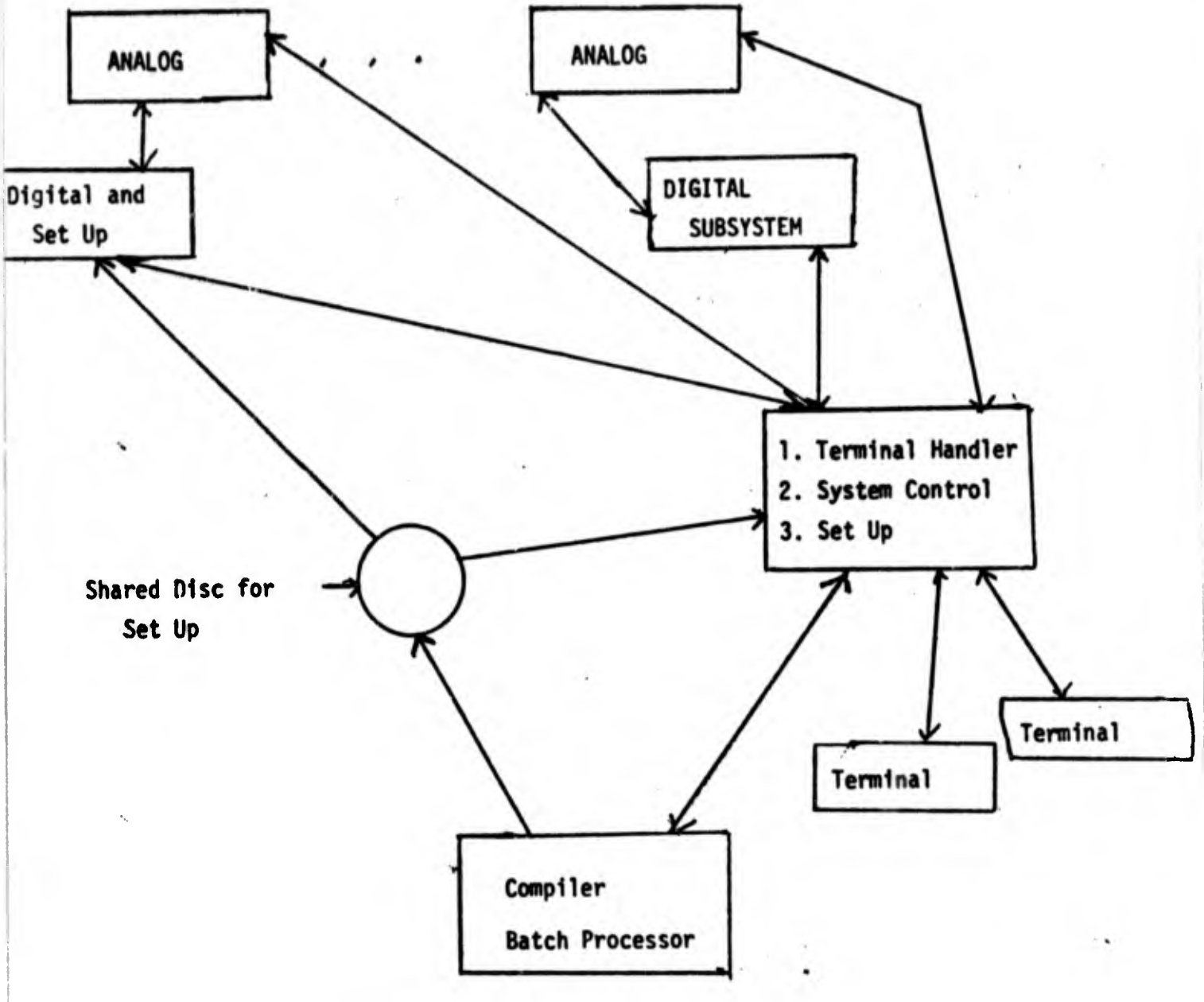


Fig. 3 Multi User AHCS Configuration II

Development of a compiler from scratch will duplicate much of this effort. One problem that can be foreseen is the problem of software being in the public domain. At the time that PATCH was developed in conjunction with CSMP, source programs were made available to us by IBM. It is possible that special arrangements could be made with the vendor of interest. Another possibility is to use RSSL and arrange with Raytheon to take system support of the complete compiler.

Purchase of HCS-1 would again save time and effort.

In summary, a great deal of groundwork has been done on compiler software and efforts should concentrate on improving, expanding and refining these approaches.

IV. CHOICE OF COMPILER LANGUAGE

With the possibility of the development of different Advanced Hybrid Computer Systems for different laboratories it becomes increasingly important to specify a source language to insure portability of source language programs. The most likely candidates at this time appear to be CSMP, CSSL and SL-1.

CSSL is upward compatible to SL-1. CSMP could be forced to be upward compatible as well. An SCI committee is presently studying additions and modifications to the CSSL specifications and is anxious to receive input from any users.

Specification of CSSL or a sub-set or super-set of CSSL would be reasonable for the family of AHCS computer systems.

V. PORTABILITY OF COMPILER

This is probably more of a vendor problem than a user problem. Because of the high dependence on operating system facilities a sophisticated compiler would be machine dependent to some degree. A great number of modules could be

made portable. The output from the compiler must again be determined by the digital processor or processors used for set-up and execution and by the analog computer being patched.

This problem should not limit the flexibility in choice of host computer. In fact the existing laboratory processor should be considered as a serious candidate if the traffic would allow the additional load.

CHAPTER IV

Conclusions

It is difficult to write conclusions without being too conclusive but that is what I shall attempt to do here.

The most important factor in specification of the software is to insure that the operating system will be capable of supporting multiple remote users and providing the flexibility to accommodate anticipated growth of an installation.

Secondly, a great deal of existing software is directly applicable to AHCS and should be taken advantage of rather than devoting a great deal of effort to compiler development. Particular consideration should be given to remote terminal operating systems which can handle multiple users and manage user files efficiently.

A user source language should be specified or adopted. Granted complete portability of user programs seems unrealizable as is evidenced by the example of FORTRAN IV. It might be that complete portability is not even necessary since it limits the possibility of taking advantages of nice features in the hardware and operating system of a particular computer. However, these refinements should be apparent and alternatives should be available to make a portable version of the same program.

Developments are needed to provide execution time modifications to the patching on the analog subsystem. One possibility is to compile to a block-oriented simulator such as ISL, SIMEX, or DARE II. These languages have execution arrays which closely resemble a patched analog computer and do allow a great deal of interaction in terms of modifying problem structures and

monitoring variables. Preliminary studies show that these languages would provide an efficient economical approach to the diagnostic, set up and monitor phase of the problem.

The prospects for software developments are bright. A great deal of software development shows that all aspects of the software system are feasible at this time. It remains for the prospective lab manager to anticipate the needs for the next few years and specify the system accordingly.

REFERENCES

1. Selfridge, R. G., "Coding a General Purpose Digital Computer to Operate as a Differential Analyzer", Proc. Western Joint Computer Conference, 1955, pp. 82-84.

2. Stein, M.L., J. Rose, and D. B. Parker, "A Compiler with an Analog-oriented Input Language", Proceedings Western Computer Conference, 1959, pp. 92-102.

3. Petersen, H.E., F. J. Sansom, R. T. Harnett and L. M. Warshawsky, "MIDAS - How It Works and How Its Worked", Proceedings Joint Computer Conference, 1964, pp. 313-324.

4. Sansom, F.J., "MIMIC Programming Manual", Technical Report SEG-TR-67-31, Wright-Patterson Air Force Base, Ohio, July, 1967.

5. Syn, W. M. and D. G. Wyman, "DSL/90 Digital Simulation", IBM Systems Division, San Jose, R02.355.

6. System/360 Continuous System Modeling Program (CSMP), User's Manual, 360A-CX-16X, IBM Manual GH 20-0367, White Plains, New York, 1968.

7. Continuous System Modeling Program III (CSMP-III) and Graphic Feature, General Information Manual, IBM Manual GH 19-7000, White Plains, New York, 1971.

8. "The SCI Continuous System Simulation Language (CSSL)", Simulation, Vol. 9, No. 6, Dec. 1967, pp. 281-303.

9. Goltz, J. R., "The Dare I Continuous System Simulation System", EES Series Report 37, University of Arizona, 1972, pp. 41-46.

10. Control Data 6000 Computer System CSSL3, User's Guide, Version 1, Pub. No. 17304400, 1973.

11. Trevor, A. B. and J. V. Wait, "Dare-IIIB - A CSSL Type Batch-Mode Simulation Language for CDC 6000 Series Computers".

12. SL-1 Reference Manual, Xerox Data Systems, No. 90-16-76B, El Segundo, California, February 1972.

13. Pugh, A. L., "Dynamo II User's Manual, MIT Press, Cambridge, Mass., 1970.

14. Benham, R. D., "Interactive Simulation Language-8 (ISL-8), Simulation, Volume 16, 1971.

15. 1130 Continuous System Modeling Program, IBM Manual H 20-0282, White Plains, New York,

16. Liebert, T. A., "Dare II: Fast On-Line Digital Simulation on a Small Computer", EES Report 37, University of Arizona, 1972, pp. 47-51.

17. Worth, G.A., "SIMEX - Simulation Executive with Automatic Scaling", M.S. Thesis, Washington State University, 1973.

18. Anderson, W. R., "Continuous System Modeling Program for the EAI 690 Simulation System", Report No. RETR 69-22, Redstone Arsenal, AL., Nov. 1969.

19. Golden, D. G., J. D. Schoeffler, "GSL - A Combined Continuous and Discrete Simulation Language", Simulation, Vol. 20, No. 1, January 1973.

20. Hurst, N. R., "GASP IV: A Combined Continuous/Discrete FORTRAN Base Simulator Language", Ph.D. Thesis, Prudue University, 1973.

21. Thames, J. M., "PROSE - A Problem Level Programming System", Solveware Associates, San Pedro, California, 1973.

22. Enslow, P. H., "Hybrid Computer Technology in Europe", ERO Technical Report, No. ERO1-74, February 1974.

23. HOI Reference Handbook, Publ. No. 00 827.0021-3, Electronic Associates, Inc., West Long Branch, New Jersey, September 1971.

24. The APSE Compiler, Scientific Computation Department Report, Electronic Associates, Inc., West Long Branch, New Jersey, 1971.

25. Rook, P. G., "Automatic Compilation and Set up of Simulation on a Hybrid Computer Using ACTRAN", Proc. 1972 S.C.S.C., San Diego, California, June 1972.

26. Elzas, M. S., "Automated Hybrid Computers Have a Credible Future", Proc. 1973 S.C.S.C., Montreal, Canada, July 1973.

27. HL1 or Toward a Unique Language for All Continuous System Simulation (An Introduction to the HCSl Computer System), Proc. AICA, 7th Conf. on Hybrid Simulation, Prague, August 1973, pp. 65-70.

28. Franklin, M. A., "A Hybrid Computer Programming System", Ph.D. Thesis, Carnegie-Mellon University, 1970.

29. Rigas, H. B. and D. J. Coombs, "PATCH, Analog Computer Patching From a Digital Simulation Language", IEEE Transactions on Computers, Vol. 20, No. 10, October, 1971.

30. Rigas, H. B. and D. J. Coombs, "Modifications to a Digital Simulation Program to Facilitate Automatic Patching", Simulation, Vol. 19, No. 4, October 1971.