1974

A Non-Procedural High-Level Language for Automated Design of Application Systems

J. F. Nunamaker

Report Number:
74-127

A NON-PROCEDURAL HIGH-LEVEL LANGUAGE FOR AUTOMATED DESIGN OF APPLICATION SYSTEMS

by:

J. F. Nunamaker, Jr., Thomas Ho, and Benn Konsynski

Computer Sciences Department
Purdue University
West Lafayette, Indiana 47906

CSD-TR 127

A NON-PROCEDURAL HIGH-LEVEL LANGUAGE FOR AUTOMATED DESIGN OF APPLICATIONS SYSTEMS

ABSTRACT

This paper focuses on the use of high-level non-procedural languages for stating system requirements in computer-aided design of large-scale information systems. Necessary and desirable features of such a language are considered along with the resolution to a problem definition technique composed of two requirement statement languages and their analyzers as they relate to the information system design process. Desirable features of such a high-level language include the following:

- facilitates Machine Independent problem statement
- machine analyzable (for completeness and design)
- ability to provide complete information for design and optimization process
- provides non-procedural representation oriented toward non-programmers.

As no single language in present use has proven adequate for satisfaction of all of the above features, a problem definition technique was evolved from ADS (Accurately Defined Systems), SSL (SODA Statement Language), and PSL II (Problem Statement Language). Such an aggregate technique has proved adequate for present needs in satisfying the above desirable features.

The statement of the design problem must be reflected in the requirements statement language and then analyzed for completeness by automated analysis techniques before system design and optimization can begin.

The procedures and programs described are presently being incorporated into a framework that facilitates man-machine interaction for problem definition and information systems design.

A NON-PROCEDURAL HIGH-LEVEL LANGUAGE FOR AUTOMATED DESIGN OF APPLICATIONS SYSTEMS

COMPUTER AIDS FOR AUTOMATING THE SYSTEMS BUILDING PROCESS

The widespread expansion of computer applications coupled with the less spectacular growth in sources of programming manpower has created a critical situation motivating the development of tools for automating the production of software. Similar in concept to the compiler-compiler, an automated systems building tool generates software for a range of applications far wider than the compilation of high-level programming languages.

The activities performed by computer aids for systems building include:

1. Procedures for stating processing requirements.
2. Automatic analysis of processing requirements.
3. The design of program structure; i.e., determining how many modules must be generated and the size of each module.
4. The design of logical file structures and logical data base.
5. Performance evaluation of hardware and software.
6. The allocation of files to storage devices.
7. The specification of storage structures for each file.

The emphasis of this paper is on steps 1 and 2 above, while steps 3 and 4 are discussed as they influence problem statement procedures.

Processing requirements are stated in a Requirements Statement Language (RSL) or Problem Statement Language (PSL) to permit the statement of requirements for an information system without stating the procedures that will be used to implement the system. The effective use of an RSL is aided by a Requirements Statement Analyzer (RSA), a program that verifies an RSL statement and that performs logical analysis. Finally, an RSA produces a coded statement to be used by additional software components that perform the physical systems design and that automatically produce source language statements implementing the information system described by the RSL statement.

Steps 3 and 4 deal with the logical system design process [1]. As the logical design progresses, a physical design must

emerge in line with evaluation of software and selected hardware system components. Steps 5, 6, and 7 entail hardware design development, data organization, and system evaluation [2]. Previous design procedures have relied heavily on manual processes for generation of designs and use of simulation for evaluation and refinement [3]. A methodology called SODA (Systems Optimization and Design Algorithm) has been derived for the total design process from non-procedural problem statement through software design and hardware selection to final implementation and performance evaluation [4].

EXPERIENCE WITH REQUIREMENTS STATEMENT LANGUAGES

As a proposed solution to a recognized need, the RSL concept has been discussed and is now receiving increased interest in the computing community. Early references include McGee [5] and Pridmore [6]. Recent references include Sammet [7, p. 609], Benjamin [8, p. 642], and Merten and Teichroew [9]. Teichroew [10] surveys seven proposed languages and presents a set of detailed specifications for an ideal RSL.

The seven techniques discussed by Teichroew view the problem in essentially the same way. They describe how to produce outputs from inputs. All seven techniques provide some method for describing data relationships as the user views them. They provide some facility for stating the requirements of the problem. Several provide some facility for stating other data such as time and volume.

Young and Kent [11] represent the earlier work. Information Algebra is the work of the CODASYL Development Committee [12]. Two other efforts have been reported by Langefors [13 and 14] and Lombardi [15]. Accurately Defined Systems (ADS) is a product of the National Cash Register Company [16] and is described by Lynch [17]. The Time Automated Grid (TAG) system, a product of IBM, was developed by Myers [18] and is described by Kelly [19, Chap. 8]. Finally, Systematics is the work of Grindley [20].

ADS and TAG use a practical, straightforward approach without attempting to develop any "theory" of data processing. ADS or TAG consists of a systematic way of recording the information that an

analyst would gather. ADS or TAG could be used by any experienced analyst with very little instruction.

Young and Kent and Information Algebra represent a problem definition approach that is more concerned with developing a theory and use a terminology and develop a notation that is not at all natural to most analysts. Lombardi's approach requires the completion of the system design before it can be used and resembles a non-procedural programming language rather than an RSL.

However, Lombardi's work is relevant because it presents a non-procedural technique for stating requirements once the file processing runs have been determined. Langefors' technique uses the concept of precedence relationships among processes and files without indicating how these relationships are obtained and is relevant to the analysis of a problem statement rather than to the design of a system. However, it does suggest a number of desirable features of a problem statement technique. Using a specialized form of mathematics, Systematics provides facilities for stating alternative actions under various conditions, for defining non-quantitative information items, and for classifying information items into a hierarchy.

Despite the availability of these RSL techniques, their use has not been extensive. To the best of our knowledge, the languages of Young and Kent and of Lombardi have not been used except in an experimental way and the development of Systematics has been discontinued after a field trial. Information Algebra has been used only once by Katz and McGee [21]. It appears that the development and use of TAG has been discontinued by IBM. ADS appears to be gaining in user acceptance. The U. S. Navy [2], in the process of designing a Financial System, and a number of other firms [22] have used ADS as a problem statement technique.

This current work is the result of an evolutionary process involving several different RSL's. The first development SSL/I (SODA Statement Language/I) is the work of Nunamaker [4]. Extension of SSL/I resulted in the development of PSL/I (Problem Statement Language/I) described by Koch, Krohn, McGrew, and Sibley [23]. Experience with PSL/I indicated its shortcomings and led to PSL/II possessing improvements suggested by Hershey, Rataj, and Teichroew

[24]. Simultaneous with the development of PSL/II, experience with ADS demonstrated the value of a forms-oriented RSL for ease of problem definition. Hence, this report focuses on the description of SSL/II, an RSL encompassing the forms orientation of ADS and the power of expression of PSL/II. The evolution of SSL/II is illustrated in Figure 1.

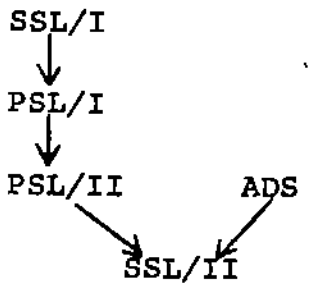

Figure 1. Evolution of SSL/II.

Overview Of Three Requirements Statement Languages

Past experience with problem statement techniques has indicated that no existing problem statement technique is adequate for the complete expression of user requirements relevant to all aspects of systems design and optimization. This deficiency motivated the initial development of SSL/I, the subsequent development of PSL/II, and examination of ADS for desirable features.

ADS is forms-oriented, thereby making it easy to use while still being capable of specifying much of the basic problem definition. SSL/I possesses additional capabilities, particularly in the specification of operational requirements consisting of information on volumes, frequency of output, and timing of input and output. Finally, PSL/II exhibits more powerful generalized facilities for data description, processing requirements, and operational requirements.

SODA Statement Language/I

An SSL/I problem statement is composed of a collection of Problem Statement Units (PSU). A PSU consists of three components: data description, processing requirements, and operational requirements.

The data description is defined by Elementary Data Sets and Data Sets. An Elementary Data Set consists of a Data Name, Data Value, Descriptor Name, and Descriptor Value. An example of an Elementary Data Set is the sales of model X in the north region:

Data Name	Descriptor Value	Descriptor Name	Data Value
SALES_MODEL_X (in the)	NORTH	REGION (is)	500.

A Data Set is the set of all Elementary Data Sets with the same Data Name. An example of a Data Set is the sales of model X in all regions of the country. There are four types of Data Sets: input, storage, terminal (reports), and computed (output of a Process).

Processing requirements consist of computational formulas described by four kinds of Processes: COMPUTE, SUM, IF, and GROUP (grouping of Data Sets that appear together on a report). Time requirements are specified by stating absolute time deadlines. Statement of time requirements for reports is expressed via a Need Vector indicating the time periods during which report production is required. Data set volumes are computed from the volumes specified for each Elementary Data Set.

An SSL/I problem statement exhibits the following structure:

```
Problem Statement Name
   List of Identifiers
   List of Descriptors
      Descriptor Name
      The Number of Descriptor Values for each
         Descriptor Name
   List of Data Sets
      Data Name
      Volume of Data Set
      Type of Data Set
   List of PSU
      Contents of each PSU
         PSU Number
         PSU Name
         Need Vector
         List of Processes
         END of PSU
END OF PROBLEM STATEMENT.
```

SODA Statement Analyzer

SODA Statement Analyzer (SSA) accepts the requirements stated in SSL/I, analyzes them, and provides the problem definer

with diagnostics for debugging his problem statement. SSA also produces a number of networks which record the interrelationships of Processes and data and passes the networks on to the SODA program concerned with the generation of alternative designs.

Each type of input and output is specified in terms of the data involved and the transformation needed to produce output from input and stored data. Time and volume requirements are also stated. SSA analyzes the statement of the problem to determine whether the required output can be produced from the available inputs. The problem statement stored in machine-readable form is processed by SSA which:

1. Checks for consistency in the problem statement and checks syntax in accordance with SSL; i.e., verifies that the problem statement satisfies SSL rules and is consistent, unambiguous, and complete.
2. Prepares summary analyses and error comments to aid the problem definer in correcting, modifying, and extending his problem statement.
3. Prepares data to pass the problem statement on to the SODA program concerned with generation of alternative designs.
4. Prepares a number of matrices that express the interrelationships of Processes and data.

Problem Statement Language/II

To fulfill the needs outlined in this report, the ISDOS (Information Systems Design and Optimization System) Project at the University of Michigan has designed PSL/II (Problem Statement Language/II), a prototype Requirements Statement Language, and is currently implementing PSA (Problem Statement Analyzer), a Requirements Statement Analyzer to analyze PSL/II statements [25].

A basic development of ISDOS has been PSL/II [24], a language to communicate the needs of the user to the ISDOS software. A PSL/II statement specifies the time and volume characteristics which govern the production of outputs and the acceptance of inputs, and the formulas to be used to compute the values of data elements in the outputs. PSL/II is distinguished by the variety of facilities it makes available for problem statement.

PSL/II is a free-form narrative language with English-like statements conforming to specific syntax rules. A PSL/II problem

statement consists of one or more sections to enable modular problem definition. The possible sections include:

1. Real World Entity (RWE) section
2. Problem Definer (PD) section
3. Principal Data Set (PDS) section
4. GROUP section
5. HISTORY SET section
6. DEFINE section
7. PROCESS section
8. FUNCTION section
9. CONDITION section
10. EVENT section.

A Real World Entity section describes some part of the organization for which the information system is being defined. This description is generally in the form of a narrative comment.

A Problem Definer section identifies a problem definer, the location, e.g. post office box, to which messages for the problem definer are to be sent, and the sections of the problem statement for which the problem definer is responsible.

A Principal Data Set section describes an input or output of the information system being defined. The section includes descriptions of:

1. Time of occurrence and volume of the PDS.
2. Content and logical data structure of the PDS.
3. Lists of PROCESS(es) and FUNCTION(s) which involve the PDS.
4. Estimated cost of the PDS.
5. Value of the PDS on some arbitrary scale.

A GROUP section defines the logical structure of an intermediate node in a hierarchical (tree) data structure. In addition, the section identifies the processes and functions in which the group is involved.

A HISTORY SET section defines the logical structure of a grouping of data elements which must be stored by the information system being described. Rules for updating the history set are specified along with the processes and functions in which the history set is involved.

A DEFINE section assigns various attributes to a user defined name. The attributes include:

1. ELEMENT, terminal node in a hierarchical data structure
2. DATA TYPE

3. SYSTEM PARAMETER
4. KEYWORD for information retrieval
5. AUTHORITY for expenditure
6. SOURCE of information
7. SECURITY, access key to insure privacy
8. OPERAND
9. INTERVAL
10. PREDEFINED.

In addition, this section enables:

1. Specification of synonym(s) for the data name being defined.
2. Identification of FUNCTION(s) or PROCESS(es) which either:
   a. Use the data name being defined.
   b. Modify the value of the data name being defined.
   c. Derive the data name being defined.
3. Specification of the conditions which the value of the defined data name must satisfy, e.g. validation rules.
4. Assignment of identifiers to distinguish instances of the defined data name.

A PROCESS section defines a process which is a component of the information system being defined. A PROCESS is a collection of FUNCTION(s) and other PROCESS(es). The largest process is the entire problem itself. Problem definition is performed top-down so that the problem definer first defines the largest processes and subsequently defines the smaller processes of which the larger processes are composed until all constituent processes are defined. For each process, its operand(s) and result(s) are specified. In addition, the relationship of the process to other processes and functions is identified. Finally, information on the occurrence and timing of the process is specified.

A FUNCTION section defines a set of computations for determining the value of an element defined in the problem statement. The function definition is either a decision table or an arithmetic expression. As in the case of a process, the relationship of the function to other functions and processes is identified. Information on occurrence and timing is also specified.

A CONDITION section defines a logical condition as a conditional expression composed of arithmetic expressions, relational operators, and boolean operators. For example, a CONDITION section may be included in a PSL/II problem statement to specify the condition, e.g. gross pay less than or equal to zero, under which a

paycheck is not issued.

Finally, an EVENT section defines an event that must be recognized by the information system being defined. Such an event can then be used in either a PDS, PROCESS, FUNCTION, or CONDITION section to specify its time of occurrence.

PSA accepts inputs in PSL/II and analyzes them for correct syntax. PSA checks the PSL/II statement for completeness and consistency and produces a coded statement of specifications for the other modules of ISDOS.

Accurately Defined Systems

Accurately Defined Systems (ADS) is a product of the National Cash Register Company [16] and is described by Lynch [17]. ADS consists of a set of forms and procedures for systematically recording the information that a systems analyst would gather during compilation of the user requirements for the information system to be implemented. The essential elements of an ADS requirements statement include descriptions of:

1. Inputs to the information system.
2. Historical data stored by the information system.
3. Outputs produced by the information system.
4. Actions required to produce these outputs and the conditions under which each action is performed.

ADS Analyzer

Computer-aided analysis of an ADS statement performs a number of checks and prepares a series of summaries of the statement of user requirements. The simplest kind of check performed involves the validation of ADS source statements to uncover any violations of the syntax rules of ADS problem statement. Rules relating to naming conventions, numbering conventions, information linking, and the like are specified to guide the user during problem definition.

More complex checks of logical consistency and completeness indicate errors in data element definition and in linking of information sources. Major errors of a logical nature include the use of data elements not defined elsewhere in the ADS statement and the redundant definition of data elements with multiple

occurrences in the ADS statement. Less serious errors involve historical data elements for which no update procedures have been specified and definition of data elements not used elsewhere in the ADS statement.

Summary reports produced by computer-aided analysis include a directory of all data element occurrences, indexes to all data elements and processes, matrices indicating the data elements required by each process and the precedence relationships among data elements, and graphical displays of the ADS forms submitted for analysis. The data element directory consists of an alphabetical list of the data elements defined in the ADS statement, the places of occurrence of each element, and the information source of each occurrence. The indexes assign a unique number to each data element and process for identifying row and column positions in the matrices indicating incidence and precedence relationships. The incidence matrix uses process numbers as row indexes and data element numbers as column indexes to identify the data elements used in each computational process. The precedence matrix uses data element numbers as both row and column indexes to indicate, for each data element, the data elements that must be computed before the first data element can be calculated. Finally, the graphical reports display the five kinds of ADS forms in the tabular manner that they would appear in manual use of ADS.

ADS

The ADS requirements statement begins with the definition of all system outputs. Then definition continues with the identification of information that enters the system in order to describe inputs to the system. Finally, the requirements statement is completed with the definition of historical data retained in the system for a period of time and with the specification of computations and accompanying logic that subsequently use the input and historical data to produce the system outputs.

Linking of information elements among the various ADS definitions is accomplished in two ways. First, each element of data is assigned a unique name that is always used whenever that element appears in any ADS definition. Second, each use of a data element in a report, history, or computation definition is linked back to

its information source elsewhere in the ADS description. Hence, all data elements are chained from output to input and each output can ultimately be expressed in terms of inputs to the system. Chaining is accomplished by assigning page and line numbers to all ADS forms so that each use of a data element can be uniquely identified by the form, page, and line on which the element appears.

An example of an ADS requirements statement will demonstrate the effectiveness of the concepts described above. The ADS example describes the requirements of an application for payroll calculation:

The application produces an output report listing social security number, name, and current pay period wages for each employee. Also, the application includes a master file containing the following information in each employee record:

1. Social security number.
2. Name.
3. Wage status.
4. Hourly rate or pay period salary.
5. Year-to-date wages.

Input to the application is a set of time cards containing the pay period date, employee social security number, and number of hours worked during the pay period.

Computations include two types: current wage calculation and year-to-date wage calculation. Current wage calculation is performed for both salaried and hourly paid employees. Hourly calculations are further subdivided into straight-time calculation and overtime calculation. Finally, the logic definition form presents a decision table specifying the conditions under which each computation is performed.

Note the facility for cross-referencing data elements among the various forms. For example, Section III of the report definition form in Figure 2a specifies the source of each element on the report. Similarly, each entry in the history and computation definition forms in Figure 3 includes an indication of the source of the data element specified. Since this example includes only wage calculation and not master file maintenance, the source of all history data elements cannot be specified here. Furthermore, the forms may be incomplete in other respects due to the omission of non-essential details, e.g. report headings, in this example.

In Figure 2a, the Report Definition Form describes the printed output produced by the application. Section I documents the layout of the report by using the symbols identified in the upper right-hand corner to describe the printed fields. The number in parentheses below each field refers to the numbered items in Section III. Section III identifies the source of each data item appearing on the report. Cross-referencing is achieved by specifying H, C, or I for history, computation, or input respectively and by specifying page and line numbers that appear on every form. Section IV shows the sequence in which the output data is listed on the report.

Figure 2b is the Input Definition Form, a description of the input to the source program. Section I describes the format of the input record and is linked to the complete description of each field in Section II. Section II identifies the alphabetic, numeric, or alphanumeric character of each field and its size in number of characters.

The History Definition Form, a description of the master file maintained by the application, appears in Figure 3a. Again, each field is completely described. In addition, the memo entry in line 5 refers to an explanation of the wage status code in the memo list that actually appears on the Input Definition Form.

The Computation and Logic Definition Forms are displayed in Figure 3b. The Computation Definition Form lists the variables to be computed and the factors needed to perform the computations. Again, the source of each factor is specified. The entry in the sign column identifies the arithmetic operation to be performed. Since only binary operators are allowed, temporary variables must be generated for intermediate results and are given mnemonic names here for clarity. The Logic Definition Form represents a decision table that specifies the conditions under which each computation is performed. The computations are listed across the top and linked to the Computation Definition Form while the conditions are specified down the righthand side.

ADS possesses obvious advantages over the traditional narrative requirements statement technique. Narrative statements are ambiguous and often incomplete while ADS provides a standardized

(a)

REPORT DEFINITION for PAYROLL Application

NAME OF REPORT: PAY-REPORT

PREPARED BY

DATE PAGE 1 OF 1

SYMBOL IDENTIFICATION

I REPORT LAYOUT

1 2 3 4 5 6 7

6 θ 9 X X SSS9,999

(1) (2) (3)

(a)

III CROSS REFERENCE TO FIELD NAMES
(WHEN FIELD HEADING INCLUDES MULTIPLE CODES, LIST ALL CODES ON ATTACHMENT)

	NAME	MEMO	✓	SOURCE		
				H C I	PAGE	LINE
1	EMPL-SSN			H	1	1
2	EMPL-NAME			H	1	3
3	WAGES			O	1	1

(a)

IV

MAJOR LEVEL 1 EMPL-SSN (H,01,01)

LEVEL 2

LEVEL 3

(b)

INPUT DEFINITION for PAYROLL Application

WHAT IS SORT KEY AND NORMAL INPUT SEQUENCE: TIME-CARD-SSN

MEDIA TYPE PUNCHED CARD

VOLUME AV. PEAK

NAME OF INPUT RECORD TIME-CARD

PREPARED BY

DATE PAGE 1 OF 1

FOR VARIABLE FORMATS, ENTER I.D. CODE: ψ

NAME OF FORMAT

TIME-CARD-DATE (1)	TIME-CARD SSN (2)	TIME CARD HRS (3)

I INPUT MEDIA LAYOUT

(b)

II COMPLETE DESCRIPTION BELOW OF EACH FIELD IN SECTION I — INPUT MEDIA LAYOUT

LINE	FIELD NAME	SIGN	*	SIZE	DEC	VALIDATION RULES
1	TIME-CARD-DATE		N	6		
2	TIME-CARD-SSN		N	9		
3	TIME-CARD-HRS		N	3		

Figure 2: Report and Input Definition Forms

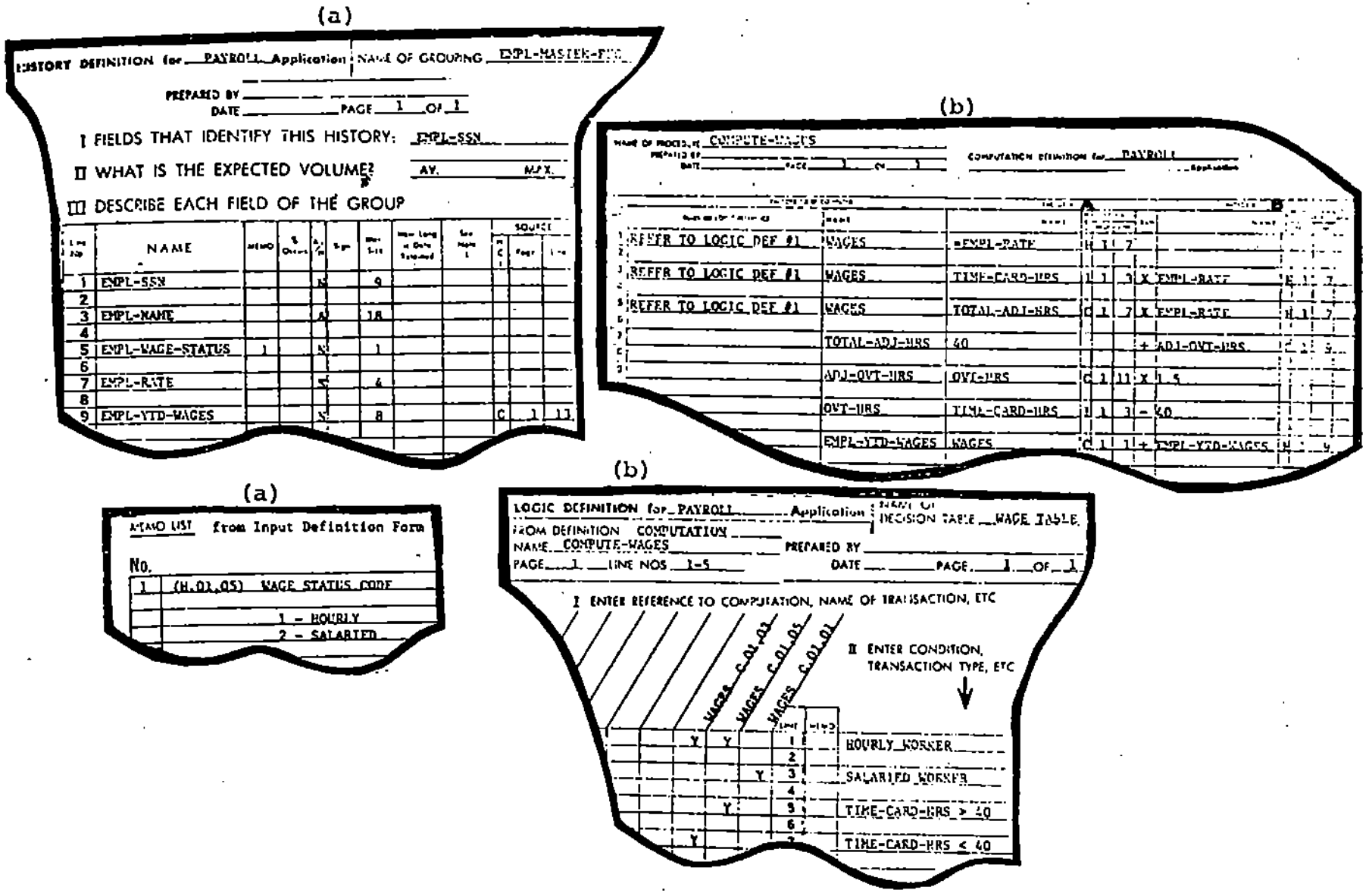

Figure 3: History, Computation and Logic Definition Forms

and systematic approach to system definition. Still, ADS is both exact and precise while remaining hardware independent. ADS promotes effective communication among systems personnel by imposing a discipline that enables the efficient use of human and machine resources. Development time is reduced while software quality is enhanced because the ADS technique enables checking for accuracy, consistency, and completeness of the requirements statement. Above all, dollar savings are realized with the use of ADS for problem definition.

ADS Analyzer

The first module of the Problem Statement Analyzer for ADS (PSA/ADS) performs source deck validation, lists the input cards, creates a file containing all valid card images, and constructs a dictionary table to be used by other PSA/ADS modules. Source deck validation checks compliance with ADS syntax rules and detects errors that include:

1. Specification of an illegal form type, i.e., neither Report, Input, History, Computation, nor Logic.
2. Improper form format.
3. Illegal data element name.
4. Invalid page or line numbering.

For each valid ADS entry, the dictionary table records:

1. Place of occurrence.
    a. Form type.
    b. Page number.
    c. Line number.
2. Data element name.
3. Information source.
    a. Form type.
    b. Page number.
    c. Line number.

Then, the dictionary is sorted, in ascending order, according to the following keys listed in major to minor order:

1. Data element name.
2. Place of occurrence.
    a. Form and entry type.
    b. Page number.
    c. Line number.

The second module of PSA/ADS prints the data element directory and constructs a symbol table containing all data element names in alphabetical order. Obtained from the sorted dictionary table, the data element directory lists the data elements in alphabetical order and provides the following information for each data element:

1. Place(s) of occurrence.
    a. Form type.
    b. Page number.
    c. Line number.
2. Information source(s).
    a. Form type.
    b. Page number.
    c. Line number.

During directory printing, the second module performs logical checks to detect the following errors and warnings:

1. ERROR: NO SOURCE OF INFORMATION.
   A data element has been used, but it has never been defined as an input or as the result of a computation.
2. ERROR: ID IS NOT IN BODY OF FORM.
   A data element has been defined as an identifier, usually for sequencing purposes, of a data grouping that appears on a History or Input Definition Form, but the identifier does not appear as one of the data elements defined in the body of the form.
3. WARNING: NO UPDATE FOR HISTORY.
   A data element has been defined in a History Definition Form, but the element has not been defined as a result of a computation. This situation is an error only if the data element represents cumulative data, e.g., year-to-date total. If the data element represents relatively constant data, e.g., employee address, that is updated from input elements, this situation is not an error.
4. WARNING: NOT USED.
   A data element has been defined as an input or as a result of a computation, but it is not subsequently used as an operand in a computation, as a report or history item, or as a decision variable in a Logic Definition Form.
5. WARNING: REDUNDANT INPUTS.
   A data element appears on more than one Input Definition Form in which the element is not used as an identifier, e.g., for sequencing purposes. Hence, only those input definitions using that data element as an identifier are probably necessary.

6. WARNING: REDUNDANT HISTORIES.
   A data element appears on more than one History Definition Form in which the element is not used as an identifier, e.g., for sequencing purposes. Hence, only those history definitions using that data element as an identifier are probably necessary.

7. WARNING: BOTH INPUT AND COMPUTED.
   A data element has been defined as both an input and the result of a computation, but it does not appear as an operand in a computation. Unless the input data element is being used to verify the computed data element, either the input or computation definition is unnecessary.

8. ERROR: INVALID BACK REFERENCE.
   A data element has been defined with an information source that is not valid. Possible causes include specification of a report definition item as an information source, specification of a non-existent page or line number, and reference to an ADS entry (as an information source) where the desired data element does not exist.

9. ERROR: NO SOURCE OF INFORMATION.
   A data element has been defined for which no information source can be found, i.e., no other definition of that element can be found on any Input, History, or Computation Definition Form.

Also, the second module assigns a unique number to each data element and prints an alphabetical list of the data elements used in the ADS statement. Then, the sorted dictionary table is again sorted, in ascending order, according to the following keys, listed in major to minor order:

1. Form type (numeric).
   a. Report: form type = 1
   b. Input: form type = 2
   c. Computation: form type = 3
   d. Logic: form type = 4
   e. History: form type = 5.
2. Page number.
3. Line number.
4. Entry type (each form consists of different entry types).

The third module of PSA/ADS creates a file containing records of the computational processes defined in the ADS statement, prints a list of the computational processes, and generates matrices displaying the incidence and precedence relationships among the data elements and processes defined in the ADS statement. The third module reads entries from the twice-sorted dictionary table

and for each computation entry, the module writes one or more (depending on the number of operands in the computation) records on the file of computational processes. Each record has the form:

1. Symbol table pointer of the data element that appears as the result of the computation entry.
2. Symbol table pointer of the data element that appears as an operand of the computation for which the first pointer identifies the result.

At the same time, the third module inserts ADS form page delimiters into the card image file produced by the first module for forms printing by the fourth module. The process file is then sorted in ascending order. Since the data elements were placed in the symbol table in alphabetical order by the second module, this sort lists the processes in alphabetical order and the operands in alphabetical order within each process. Then, the third module generates the incidence matrix indicating the data elements that serve as result and as operands for each process. These relationships are easily derived from the result-operand pairs in the sorted process file. Also, an alphabetical list of the processes is generated with the operands of each process listed alphabetically. Again, the sorted process file is sorted in ascending order according to the following keys in major to minor order:

1. Symbol table pointer of operand.
2. Symbol table pointer of result.

Finally, the twice-sorted process file is used to generate the precedence matrix indicating the direct precedents of each process. Data element I is said to be a precedent of data element J if I must be computed before J can be computed. A direct predecent of J is a precedent of J that is not also a precedent of any other precedents of J. To generate the precedence matrix, the module reads each record in the twice-sorted process file and identifies the operand data element indicated in the second field of the record as a direct precedent of the process result data element indicated in the first field of the same record.

Finally, the card image file created by the first module is sorted, in ascending order, according to the following keys in

major to minor order:

1. Form type (numeric, see keys of dictionary sort for legend).
2. Page number.
3. Line number.
4. Entry type.

The fourth and final module reads the sorted card image file and prints the input in a tabular format similar to that of the ADS forms developed by NCR.

Navy Experience With ADS

The SSL/II language being developed is a result of experience gained from working with the United States Navy Material Command Support Activity (NMCSA). The Navy statement of requirements for a financial management system was expressed in ADS by a large Accounting Firm. The ADS statement for the Navy system includes descriptions for 79 reports and for the accompanying history files, computations, and inputs which define 791 data elements. An ADS analyzer, developed at the University of Michigan [26] was used to check the ADS statement of requirements for completeness, consistency and logical accuracy. The ADS analyzer produced information and reports that were used by the SODA Statement Analyzer. SODA was then used to (1) generate preliminary designs of program structure and logical data base structure for the batch application part of the system and (2) to recommend a computer system for the entire financial management system.

The Navy integrated financial management system is a large-scale design and implementation effort for more effective financial management, particularly procurement accounting, within the agency. The systems design effort commenced in May, 1971, and is expected to continue for 4 to 5 years at a cost of 12 million dollars.

A systems design effort of this magnitude has an impact upon many different offices within the complex organization of the agency. Financial managers, the end-users of the system, are scattered among many offices engaged in complicated communication of varied information requirements.

Behavioral Experience With ADS

The first objective of the introduction of ADS into any environment is gaining user acceptance. ADS represents deviation from the established practices and initial resistance to change often occurs. As a result, many questions regarding ADS and its impact upon the organization are raised.

In response to this initial user reaction, an ADS training program is advisable. However, ADS is simple and straightforward so less than one day of intensive training is all that is necessary to adequately prepare individuals to begin using ADS. Then, further training is required only to deal with the specific restrictions imposed upon the use of ADS by the ADS Analyzer software. For example, the Analyzer restricts the length of data element names to forty characters.

The use of a form-oriented procedure such as ADS still requires a significant investment of time and effort to realize the return of a complete and consistent logical systems design. Still, a number of users with ADS experience agree that ADS has saved them considerable time during the specification of logical system design.

This savings is realized by the capability of the ADS Analyzer to provide feedback information to the user. The user should be able to do a better job of specifying his requirements because he receives feedback much sooner in the system design cycle utilizing computer analysis of ADS. Ordinarily, in a completely manual narrative system, ambiguities and omissions in the logical system description are not discovered until physical design or even coding is well underway. By then, many aspects of the system design have been specified so that resolution of difficulties may be impossible.

Physical system design is not the responsibility of the ADS user. Completion of the ADS logical description is followed by the physical system design process that provides the specifications for programming.

Performance of ADS

Experience has demonstrated that ADS is adequate for specification of the logical system. However, an ADS description does not provide sufficient information for optimization of physical system design. Data on system performance requirements was collected to supplement the ADS description in SODA Statement Language. Relevant data includes specification of the frequency of occurrence of each ADS - described input and report and of the volume of each input, report, and history.

Other needed enhancements to computer-aided ADS include facilities for describing data structures and look-up tables and for decision tables expressing processing logic and input validation rules. Finally, additional software for generating report layouts and program test data would add significantly to computer-aided ADS capabilities. Many of these enhancements are to be included in the SODA Statement Analyzer for SSL/II.

ADS Shortcomings

The decision to use ADS as the basis for SSL/II motivates examination of the shortcomings of the current implementation of machine-aided ADS and resolution of these issues before implementation of the system described in this report.

The most obvious need relates to the general orientation of the problem definition technique to machine analysis. The establishment of an effective machine orientation involves diverse issues as straightforward as conventions for naming data elements and as subtle as the manner in which sources of information are referenced.

The most fundamental modification of ADS to enhance machine analysis involves the manner in which occurrences of data elements are referenced in order to specify sources of information. The current implementation only allows a single source of information, i.e. a back reference, to be specified for any data element occurrence. An improved implementation should also enable the specification of multiple references. Hence, references would be used for the qualification of data element occurrences rather than the

mere location of data element occurrences.

Another issue involves the facilities available for description of data structures. Currently, ADS allows only two types of structure: data elements and forms. Although the simplicity of the problem statement technique is an issue of primary concern, the availability of some data structuring capability is essential. Hence, an optional data structuring facility at least capable of describing repeating groups should enable sufficient precision when necessary while still preserving simplicity otherwise. Another related issue involves the use of identifiers. The use of identifiers in ADS is limited to the specification of sort keys in history, input, and report definitions. The notion of an identifier should be broadened to distinguish each occurrence of a data grouping from every other occurrence of the grouping.

Another issue relates to the problem statement facilities for specifying computations and their accompanying logic. The use of the Computation and Logic Definition Forms can best be enhanced by the creation of a single form for specifying decision tables. Such a form will greatly improve machine analysis of logical consistency and completeness of the problem statement. While strong logical connection is desirable, care must be taken to prevent use of the decision table form to "program" rather than to "describe" the system. Another use of the decision table form includes the specification of validation rules for input data elements.

The final issue revolves around the need to provide time and volume information for the arrival of inputs and for the production of histories and reports. Although ADS currently accommodates volume information that is not included in machine analysis, there is no formal method for expressing timing information. Hence, the volume information is relatively useless since there is no way to specify the time period in which the given volume is produced or to specify the time of occurrence for a report.

SSL/II: A FORMS-ORIENTED FRONT-END PROBLEM STATEMENT TECHNIQUE

Motivated by the need for better methods of constructing large software systems, research indicates that problem statement

techniques offer a feasible approach to stating the requirements of an information system without stating the processing procedures that will satisfy those requirements. This approach further motivates the development of software tools for validating the consistency and completeness of the statement of requirements and for optimizing the design of the information system fulfilling those requirements.

Research conducted by the ISDOS Project has resulted in the development of PSL/II. In spite of the powerful facilities for problem statement possessed by PSL/II, examination of PSL/II reveals the apparent need for a forms-oriented front end problem statement technique. Characterized as a free-format technique requiring knowledge of restrictive syntactic rules, PSL/II may require a good deal of training before it can be used by a relatively sophisticated problem definer. Therefore, the development of a forms-oriented front-end problem statement technique seems advisable to enable initial problem definition by a relatively naive user. Then, software can generate a PSL/II representation so that the initial problem statement can be supplemented with the use of the more powerful problem statement facilities available in PSL/II. By translating the forms-oriented front-end problem statement into a data base representation developed for PSL/II, it may be feasible to extract a PSL/II representation of the problem statement from the data base. Then, the software can generate incomplete PSL/II statements for the problem definer to insert the missing information. In this way, a complete PSL/II problem statement can be constructed without requiring user knowledge of the full PSL/II syntax. It is important to note that the data base is the medium enabling use of two complementary problem statement techniques: the forms-oriented technique is used during initial problem statement preparation while PSL/II serves as a vehicle for problem statement completion since PSL/II is viewed as the model for a complete problem statement.

Experience with ADS at the U. S. Navy Material Command Support Activity, Duncan Electric, and other installations supports the choice of ADS as the basis for a front-end forms-oriented

problem statement technique. This paper describes a design of a forms-oriented problem statement technique based on ADS and for development of software for problem statement analysis.

Specifications Of A Software System To Aid Statement Of User Requirements

SSL/II is composed of two sublanguages to provide facilities for expression of user requirements relevant to all aspects of system design and optimization:

1. Phase I is derived from ADS and is augmented with facilities for expressing performance requirements, e.g. I/O volumes and frequencies, and data structures. Phase I is forms-oriented and intended for use while initially preparing a statement of user requirements.
2. Phase II is a free-format representation of user requirements produced by the software for subsequent use in generating specifications for program modules and files.

Three software packages are required for the proper interaction of the Phase I and Phase II representations of user requirements. The first software package analyzes the Phase I statement and prepares diagnostic messages to the user to aid him in preparing a complete and consistent Phase I representation. The second software package produces the Phase II representation of the problem statement in preparation for generating specifications for program modules and files. Phase II possesses expanded capabilities that enable the user to furnish additional details required for complete problem statement. For example, procedures related to file security, e.g. specification of passwords, might be defined during Phase II. Hence, the second software package invokes the query language to form requests to the user. Generally, these requests are in the form of incomplete Phase II statements with blanks to be filled in by the user. Finally, upon satisfaction of all query requests, a final database representation is prepared for use during specifications generation and accompanying management summary reports are generated for user perusal. These reports are graphic descriptions of the information system to be designed and include summaries of the following kinds of information:

1. Size
    a. Number of processes and data elements defined.

b. Estimated size of data base.

2. Data relationships

   a. Incidence relations indicating the data items required by each computational process.
   b. Precedence relations among the data items.

3. Workload and performance

   a. Number of reports produced in each processing cycle.
   b. Estimates of system parameters, e.g. transport volume.

The third software package is a control program which feeds diagnostic messages and query requests to the user and which relays the resulting response from the user to the appropriate software package.

The justification for two phases of problem statement representation results from the need to provide an easy-to-use, human-oriented method for statement of user requirements while still maintaining a rigorous, complete representation for machine analysis. Therefore, Phase I is forms-oriented to guide the user during problem statement preparation while Phase II possesses expanded problem statement facilities to insure a representation suitable for complex computer-aided analysis.

A fourth software package accepts the data base representation generated by the previous packages and proceeds to generate logical design specifications for program modules and files. The logical flow of the software described herein is illustrated in Figure 4.

A Data Structure Facility For A Forms-oriented Front-end Problem Statement Technique

Experience with ADS indicated the need for a facility capable of describing a wide variety of logical structures in data description. However, it is important to realize that such a facility must enable the problem definer to specify the logical relationships among the data items he describes without requiring him to define the specific structures required to represent those relationships. Hence, we describe a hierarchy of data structural units to be made available to the problem definer in a simple relational manner that will enable the interactive design of a

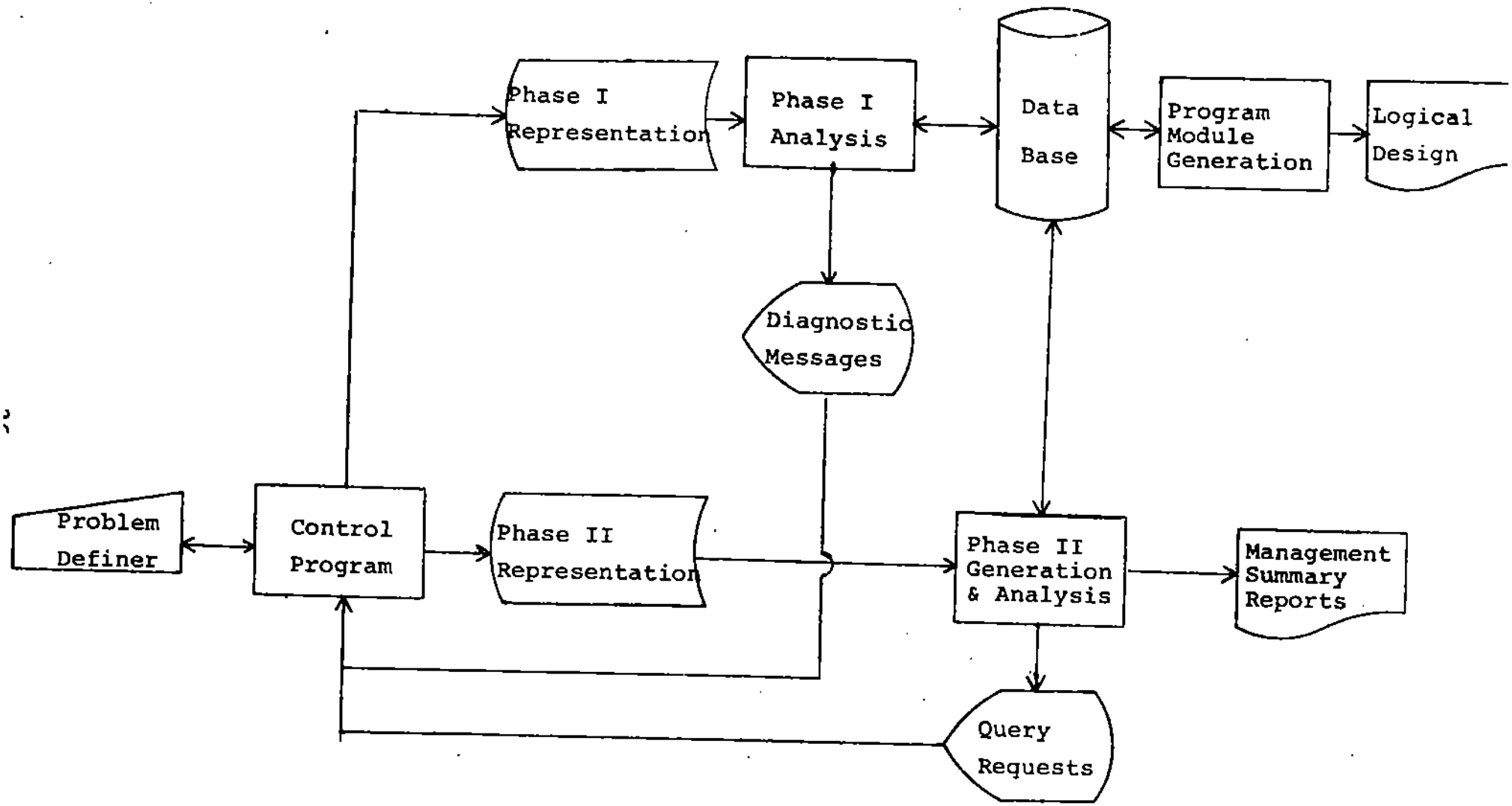

Figure 4: Logical Flow of Requirements Statement Analysis

data base for the application desired.

The Data Structure Class

The types of structures available to the problem definer and the manner in which structures of each type are constructed from other structures describe the data structure class of the problem statement technique. The available structure types and their component structures include:

Structure	Component Structure
item	none
group	item, group
group relation	group
record	group, group relation
file	record, group relation
data base	file

Item

The elementary data structure is the item. The item is the smallest structural unit from which all available structure types are ultimately constructed.

Group

A group is a collection of items or other groups. A simple group is a collection of items only while a compound group is a collection of both items and groups.

A simple group can be used in two ways. One, it can be defined as a collection of items in order to give the collection a name and other attributes of its own. An example is the group EMPLOYEE composed of the items NAME, SOCIAL-SECURITY-NUMBER, WAGE-STATUS, and RATE. Also, the items CHILD-NAME and AGE form the simple group OFFSPRING. Second, a simple group can be defined as a collection of string-valued items having a "collective value" formed by concatenating the string-valued item components. For example, the items MONTH, DAY, and YEAR form the group DATE-OF-HIRE.

A compound group is a collection of a set of items, called principal items, and a set of groups, called principal groups, with this new collection having a name and other attributes of its

own. For example, if the groups DATE-OF-HIRE and OFFSPRING are added to the simple group EMPLOYEE, the result is a new compound group EMPLOYEE consisting of the items NAME, SOCIAL-SECURITY-NUMBER, WAGE-STATUS, and RATE and the simple groups DATE-OF-HIRE and OFF-SPRING.

A group may be either repeating or non-repeating. A repeating group may have an arbitrary number of occurrences for each occurrence of the compound group containing the repeating group. A non-repeating group has only one occurrence for each occurrence of the containing compound group. For example, OFF-SPRING is a repeating group because the number of children can vary from employee to employee. However, DATE-OF-HIRE is a non-repeating group because each employee has only one hiring date.

Group Relation

A group relation is a mapping between two sets of groups. The groups belonging to the first set are called parent groups and those belonging to the second set are called dependent groups.

The group relation provides a way of relating groups. For example, with a set of parent PERSON group occurrences:

$$\{\text{PERSON(JOHN DOE)}, \text{PERSON(J. SMITH)}\}$$

and with a set of dependent SKILL group occurrences:

$$\{\text{SKILL(1000)}, \text{SKILL(2000)}, \text{SKILL(3000)}, \text{SKILL(4000)}\},$$

a group relation can be created to relate each person to the skill(s) he possesses:

$$\left\{\begin{array}{l}\langle\text{PERSON(JOHN DOE)}, \text{SKILL(3000)}\rangle, \\ \langle\text{PERSON(J. SMITH)}, \text{SKILL(2000)}\rangle, \\ \langle\text{PERSON(J. SMITH)}, \text{SKILL(3000)}\rangle\end{array}\right\}.$$

Also, the group relation provides a way to establish a hierarchic relation between two sets of items. In a hierarchic group relation, each occurrence of a dependent group must be subordinate to one occurrence of a parent group; the dependent group occurrence cannot stand alone. An example of a hierarchic group relation associates a parent group occurrence representing a person with a set of dependent group occurrences representing the academic

degrees he holds:

$$\left\{\begin{array}{l}\text{PERSON(J. DOE), DEGREE(BS,1970,PURDUE)}\\ \text{PERSON(J. DOE), DEGREE(MS,1971,PURDUE)}\end{array}\right\}.$$

Generally, a hierarchic group relation is equivalent to a compound group. However, two differences exist:

1. In a compound group, a principal group may be subordinate to a single set of items only (the principal items), but in a group relation, a dependent group may be subordinate to many sets of items (parent groups).

2. In a compound group, the principal items do not have a collective name; the compound group name refers to the entire collection of principal items and principal groups. In a group relation, each parent group has its own name.

An occurrence of a group relation consists of one or more occurrences of each parent and dependent group, with each parent group occurrence associated with one or more dependent group occurrences. If the group relation is non-hierarchic, each dependent group occurrence may be optionally associated with one or more parent group occurrences. If the group relation is hierarchic, each dependent group occurrence must be associated with one parent group occurrence.

In a manner analogous to compound groups, a dependent group in a group relation may be repeating or non-repeating. A repeating dependent group has a variable number of occurrences for each occurrence of its parent group; a non-repeating dependent group has only one occurrence for each occurrence of its parent group.

Record

A record is a collection of groups and group relations in which one and only one group, the record-defining group, is not subordinate to any other group. The record is used to define the major entities of an application. For a given class of entities, e.g. the employees of a firm, the principal items in the record-defining group correspond to fixed entity attributes common to all entities in the given class. The items in the principal group contained in the record-defining group or in the dependent group

subordinate to the record-defining group correspond to variable entity attributes. Variable entity attributes either have multiple values or are not necessarily common to all entities in the given class.

The record-defining group may not be the dependent group in a hierarchic group relation contained in the record. However, the record-defining group may be the dependent group in a non-hierarchic group relation that relates records in the same file. A later discussion of the file describes inter-record relations.

There are three record types: the group record, the tree record, and the plex record. Each record type is a generalization of the former type so that a special case of each type is identical to the former type.

A group record is a single compound group. The compound group is the record-defining group.

A tree record is a set of hierarchic group relations arranged as a tree so that each group has at most one parent, and that one and only group, the record-defining group, has no parent.

A plex record is a set of group relations in which each group except the record-defining group is the dependent group in a hierarchic group relation. In addition, all groups in a plex record may occur in any number of non-hierarchic group relations.

File

A file is a collection of records. Hence, a file represents a collection of application entities, e.g. employees, projects, or parts. The entities represented by a file may belong to the same class, e.g. employees of a firm, or to different classes, e.g. projects and the parts used in each project.

In the sense that one record of a file can be processed without referencing another record in the same file, the records of a file are independent of one another. However, the records in a file may be explicitly inter-related in a manner apparent to the system. For example, the records in a file may be ordered on the value of the record sequencer, a set of items contained in the record. A file with unrelated records or with records related only by ordering is called an unlinked file. In addition, more general

relations are possible by permitting non-hierarchic group relations between groups in different records or between records themselves when the records are group records. A file containing records participating in these more general explicit relations is called a linked file. Of course, the records in a linked file may also be ordered.

Data Base

A data base is a set of files.

Data Structure Definition

Having described the data structure class of the problem statement technique, we now describe various facilities available for definition of the data structure chosen by the problem definer. The most common facility uses the level-number concept for explicit description of hierarchical data structures. However, the level-number concept appears inadequate for definition of a plex record. Hence, an alternative scheme involving a relational view of data is also described.

Level-number

The level-number concept is presented for definition purposes. It is not intended to imply that a problem definer will describe his data requirements in terms of level-number. The level-number structure will be constructed from the relational model presented in the next section and not by the problem definer. Once the data structure in terms of level-numbers is constructed from the relational model it is clear that files can be automatically generated.

Level-numbers are used to describe the structure of a record. The record-defining group is assigned level number 01; groups and items within the record are assigned higher, but not necessarily consecutive, level-numbers that do not exceed some specified maximum value.

A group consists of all the items and groups following it in the definition until a level-number less than or equal to the

level-number of the first group is encountered. All structures, i.e. items or groups, that form a level within the same group must have the same level-number. Whenever a name of a structure needs a level-number lower than the level-number of the name immediately preceding it in the definition, the level-number must be selected from the level-numbers of the structures that include the preceding name.

To demonstrate the use of level-numbers, a group record consisting of the single compound group EMPLOYEE described earlier is defined:

```
01 EMPLOYEE
   05 NAME
   05 SOCIAL-SECURITY-NUMBER
   05 WAGE-STATUS
   05 RATE
   05 DATE-OF-HIRE (non-repeating group)
      10 MONTH
      10 DAY
      10 YEAR
   05 OFFSPRING (repeating group)
      10 CHILD-NAME
      10 AGE
```

However, with a plex record involving any number of non-hierarchic group relations, level-numbers alone are inadequate for data structure definition. Especially when a single dependent group is subordinate to a single parent group by two distinct non-hierarchic group relations, some sort of explicit mechanism; e.g. sets, owners, and members as proposed by CODASYL [28]; is necessary to supplement level-numbers. Such a mechanism is believed to be too sophisticated for use by the relatively naive user for which this forms-oriented language is intended. Even level-numbers themselves may be unsuitable for a naive user!

Hence, a relational view of data using only tabular data structures for representing data relationships may be most suitable for a forms-oriented problem statement technique intended for relatively naive users. A table is most certainly a simple, easily understood data structure. Still, a table is adequate for representing all the data structures described earlier.

A Relational Model Of Data

A relational model of data for large data bases is described

by Codd [29 & 30]. The relational model uses tables for representing the logical data base structure.

A table is a rectangular array with the following properties:

P1: a table is column-homogeneous, i.e. all the items in any single column are of the same type, but items in different columns are not necessarily of the same type.

P2: each item in a table is either a number or a character string.

P3: all rows of a table are distinct.

P4: the ordering of rows in a table is immaterial.

P5: the columns in a table are assigned distinct names and the ordering of columns in a table is immaterial.

As a result, a table represents a relation of degree n, where n is the number of columns in the table. An example of a relation of degree 3 is the relation COMPONENT. The triple (x,y,z) belongs to this relation if the part with part number x is a component of the part with part number y and if z units of part x are needed to construct one unit of part y:

COMPONENT(SUB-PART-NO, SUP-PART-NO, QUANTITY)

2010	6020	4
2015	6020	2
2025	6020	1
3010	6030	3
3025	6030	5

We now demonstrate that the relational model possesses the capability to represent the three types of records described earlier. Hence, in addition to its comprehensibility, the relational model also possesses flexibility.

Tabular Representation Of Group Records

Consider the example of a group record called EMPLOYEE described earlier. The elimination of the principal groups DATE-OF-HIRE (non-repeating) and OFFSPRING (repeating) is accomplished with three separate relations. These three relations convey all the information contained in the group record because the item SOCIAL-SECURITY-NUMBER uniquely identifies each EMPLOYEE and the item CHILD-NAME uniquely identifies the children of each EMPLOYEE:

```
EMPLOYEE(NAME,SOCIAL-SECURITY-NUMBER,WAGE-STATUS,RATE)
DATE-OF-HIRE(SOCIAL-SECURITY-NUMBER,MONTH,DAY,YEAR)
OFFSPRING(SOCIAL-SECURITY-NUMBER,CHILD-NAME,AGE).
```

Tabular Representation Of Tree Records

Consider the example of a tree record called PERSON:

```
01 PERSON (parent of SKILL group and CHILD group)
   05 NAME
   05 NUMBER
   05 SALARY
   05 SKILL
      10 CODE
      10 TITLE
   05 CHILD (parent of PET group)
      10 CHILD-NAME
      10 AGE
      10 PET
         15 TYPE
         15 PET-NAME
```

Although identical to a group record, the tree record PERSON differs from the group record in addressability: an occurrence of the group record must be retrieved as a complete unit, but certain portions, e.g. the SKILL group, of an occurrence of the tree record can be retrieved without retrieving the entire tree record occurrence. Assuming that the following items uniquely identify the corresponding groups:

Item	Group
NUMBER	PERSON
CODE	SKILL
CHILD-NAME	CHILD
PET-NAME	PET

elimination of the group structures yields the following relations:

```
PERSON(NAME,NUMBER,SALARY)
SKILL(NUMBER,CODE,TITLE)
CHILD(NUMBER,CHILD-NAME,AGE)
PET(NUMBER,CHILD-NAME,PET-NAME,TYPE).
```

Tabular Representation Of Plex Records

Consider an example of a plex record describing relationships between suppliers and parts. The parent group SUPPLIER consists of two items: SUPPLIER-NO and SUPPLIER-DESC. The dependent group PART consists of three items: PART-NO, PART-DESC, and QUANTITY. Two relationships between SUPPLIER x and PART y exist:

the CANDIDATE relationship holds if x is capable of supplying y; the ACTUAL relationship holds if x actually supplies y. Since the dependent group PART is subordinate to the parent group SUPPLIER by two distinct non-hierarchic group relations, level-numbers alone are inadequate for representing both relationships. Assuming that SUPPLIER-NO uniquely identifies each SUPPLIER and that PART-NO uniquely identifies each PART, four relations represent both groups and the two relationships between the groups:

SUPPLIER(SUPPLIER-NO, SUPPLIER-DESC)
PART(PART-NO, PART-DESC, QUANTITY)
CANDIDATE(SUPPLIER-NO, PART-NO)
ACTUAL(SUPPLIER-NO, PART-NO).

Evaluation Of The Relational Model

With the relational model, data description is performed in a bottom-up fashion in contrast to the top-down strategy of process definition. User specification of the relational properties of the data enables the software to construct the data base by aggregating the tabular structures. User specification of the processes defining the information system is accomplished via the decomposition of the problem into its component processes. Therefore, it appears that top-down decomposition is the domain of the man while bottom-up aggregation is the domain of the machine as man and machine co-operate in the design of application systems.

PROCESS GENERATION AND PROGRAM MODULE SPECIFICATIONS FROM SSL/II DEFINITION

The SSL/II problem statement contains the basic information required to generate program module specifications from processes that may be grouped into program modules to eliminate unnecessary transport of data from history files to program modules. For example, if it is determined that two processes require the same inputs and occur in the same processing cycle, e.g. daily, then the two processes become candidates for grouping into a single program module.

SODA Generator of Alternatives (SGA) performs process generation by compiling four comprehensive summaries for each SSL/II-described report:

1. Input summary.
2. History input summary.
3. Computation summary.
4. History output summary.

Since the source of each report item is specified in the SSL/II statement, all sources that are either input items or history items are included in the input and history input summaries, respectively. For report items whose sources are computation items, the input and history input items that are used as operand factors in the computations are placed into the input and history input summaries since the sources of all computation operand factors are specified. Also, the computations required to produce the report items are placed into the computation summary. Finally, the history output summary is compiled by listing all history items whose sources are items listed in either the input, history input, or computation summaries. Therefore, the history output summary indicates those history items that might be updated by the elementary module being specified.

After generating a process for each SSL/II-specified report, SGA searches for candidates for program module grouping in two ways. First, if some process requires history inputs either identical to or forming a subset of the history inputs required by another process, the two processes are identified as candidates for grouping. If the two candidates for grouping occur in the same processing cycle, grouping into a single program module is recommended by SGA.

Similarly, if two processes update the same history outputs and occur in the same processing cycle, grouping into a single program module is recommended.

Second, if some process produces history outputs either identical to or forming a subset of the history inputs required by another process, the two processes are identified as candidates for grouping. Again, if the two candidates for grouping occur in the same processing cycle, grouping into a single program module is recommended by SGA.

The four summaries and other specifications produced for each program module become the basis for code generation in fulfillment of the requirements expressed in the original SSL statements. Both human-readable and machine-readable representations are produced to enable code generation by either manual or automatic means according to the choice of the system implementers. A pro-

totype version of SGA has been developed for use with the ADS description of the Navy information system mentioned earlier.

Program Module Grouping For The Navy Example

For the Navy information system described earlier, SGA generated 62 program modules to produce the 79 ADS-specified reports. For each program module, SGA provides the following information to the SODA Performance Evaluator (SPE):

- Brief program module title.
- Frequency of occurrence.
- Program module size, in K bytes.
- History files required for processing.
- File device type.
- Size, in bytes, of each history record input.
- Number of history records input for processing.
- Volume, in number of lines, of printed output.

For each program module, module size and number of arithmetic operations are derived from the quantity and complexity, e.g. alternative logic paths, of computations in the summary produced by SGA. Volume and size of history records input are derived from the history input summary produced by SGA. SGA performs summary analysis on all ADS-specified inputs required to produce each history item. User-provided data on input requirements was then used to derive the volume of the history item under scrutiny. The size of the history item is provided in the ADS description. Finally, twenty record groups were generated with each group containing history items that are used together in a fashion that implies logical connectivity. Each group of records forms the basis for defining history file structures. An overview of the program module specifications for fiscal reporting tasks is presented in Table 1: Batch Program Module Workload Summary.

Note that process grouping into modules and history record grouping into files were performed in a manner that spreads the workload equally among the modules to the greatest extent possible. Workload sharing is made possible by minimizing the variance in the number of computations in each module and by minimizing the variance in the number of records in each file grouping.

BATCH PROGRAM MODULE WORKLOAD SUMMARY

Application/ Program ID	Task Type	Freq/ Month	Memory Required (K bytes)	Language	File ID	Medium Code	Avg. Record Length (char.)	Record Volume	Avg. Output Length (Lines)
A. Fiscal Reporting									
1. Program Budget Status	Print	1	150	COBOL	H1 H4 H5 H11	Disk	861	2200	340
2. Appn. Status by FY and Acct.	Print	1	50	COBOL	H1 H5 H11	Disk	243	2200	23000
3. Report on Reimbursables	Print	1	35	COBOL	H1 H4	Disk	232	225	24000
4. Report on Obligations	Print	1	100	COBOL	H1 H4 H5	Disk	437	2000	1000
5. Analysis of Appropriations and Fund Balances	Print	1/year June	50	COBOL	H1 H4 H5 H11	Disk	476	2200	800
6. Line Item Report	Print	1	35	COBOL	H1 H4	Disk	232	225	24000
7. Summary Line Item Report	Print	1	50	COBOL	H1 H5 H11	Disk	263	2200	4700
8. Procurement Program Progress Report	Print	1	35	COBOL	H1 H5 H11	Disk	268	2200	4000
9. Worksheet	Print	2/year June, Dec.	25	COBOL	H1 H5	Disk	141	2000	4000

Table 1: Batch Program Module Workload Summary

GENERATION OF CODE FROM ADS AND SSL/II

Figure 5 illustrates a COBOL program that conceptually might be generated by SODA to fulfill the requirements described in the ADS statement of Figures 2 and 3. The program reads TIME-CARD-FILE, an input file of time cards, and performs the ADS-specified computations and logic to update EMPL-MASTER-FILE-IN and to produce PAY-REPORT.

The ADS description primarily provides information for generating the DATA DIVISION (part A of Figure 5) and the COMPUTE-WAGES paragraph (part C of Figure 5) of the PROCEDURE DIVISION. The remainder (part B of Figure 5) of the PROCEDURE DIVISION contains the procedures and processing logic needed for the application of the ADS logical definition to the physical implementation of the ADS-specified report generation and history file maintenance. Automatic production of this code necessary for physical implementation can be fulfilled in various ways. One software company has incorporated an additional form called an execution definition into its use of an ADS description for code generation. The execution definition form details the processing logic necessary for driving the execution of the logic and computations described in the ADS forms. Another approach to code generation might involve the incorporation of code skeletons for common data processing functions, e.g. transaction processing for master files. Then, the code skeleton is completed during code generation by providing the missing record sequencing identifiers and program termination conditions. Finally, automatic generation of code for report generation features such as positioning of heading and output lines might be accomplished by incorporating another feature into the computer-aided SSL/II report definition form for specification of report layouts and headings as in the original manual ADS system.

Still, computer-aided analysis involves much more than the rudimentary approach to code generation previously described. Current approaches to code generation from a non-procedural requirements statement merely translate logical descriptions into highly inefficient code, regardless of the quality of the original logical description. For example, deficiencies include the restriction of

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PAYROLL-CALCULATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  6500.
OBJECT-COMPUTER.  6500.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT TIME-CARD-FILE ASSIGN TO INPUT.
   SELECT PAY-REPORT ASSIGN TO OUTPUT.
   SELECT EMPL-MASTER-FILE-IN ASSIGN TO TAPE01.
   SELECT EMPL-MASTER-FILE-OUT ASSIGN TO TAPE02.

DATA DIVISION.
FILE SECTION.
FD TIME-CARD-FILE
   DATA RECORD IS TIME-CARD
   LABEL RECORDS OMITTED
   RECORD CONTAINS 18 CHARACTERS.
01 TIME-CARD.
   02 TIME-CARD-DATE                PICTURE 9(6).
   02 TIME-CARD-SSN                 PICTURE 9(9).
   02 TIME-CARD-HRS                 PICTURE 9(3).
FD PAY-REPORT
   DATA RECORD IS PAY-REPORT-REC
   LABEL RECORDS OMITTED
   RECORD CONTAINS 136 CHARACTERS.
01 PAY-REPORT-REC.
   02 FILLER                        PICTURE X(9).
   02 PAY-REPORT-SSN                PICTURE 9(9).
   02 FILLER                        PICTURE X(10).
   02 PAY-REPORT-NAME               PICTURE X(18).
   02 FILLER                        PICTURE X(10).
   02 PAY-REPORT-WAGES              PICTURE $$$9.99.
   02 FILLER                        PICTURE X(73).
FD EMPL-MASTER-FILE-IN
   DATA RECORD IS EMPL-MASTER-REC-IN
   LABEL RECORDS OMITTED
   RECORD CONTAINS 40 CHARACTERS.
01 EMPL-MASTER-REC-IN.
   02 EMPL-SSN-IN                   PICTURE 9(9).
   02 EMPL-NAME-IN                  PICTURE X(18).
   02 EMPL-WAGE-STATUS-IN           PICTURE 9.
      88 HOURLY    VALUE 1.
      88 SALARIED VALUE 2.
   02 EMPL-RATE-IN                  PICTURE 99V99.
   02 EMPL-YTD-WAGES-IN             PICTURE 9(6)V99.
FD EMPL-MASTER-FILE-OUT
   DATA RECORD IS EMPL-MASTER-REC-OUT
   LABEL RECORDS OMITTED
   RECORD CONTAINS 40 CHARACTERS.
01 EMPL-MASTER-REC-OUT.
   02 EMPL-SSN-OUT                  PICTURE 9(9).
   02 EMPL-NAME-OUT                 PICTURE X(18).
   02 EMPL-WAGE-STATUS-OUT          PICTURE 9.
   02 EMPL-RATE-OUT                 PICTURE 99V99.
   02 EMPL-YTD-WAGES-OUT            PICTURE 9(6)V99.
WORKING-STORAGE SECTION.
01 WAGES                            PICTURE 9(4)V99.

PROCEDURE DIVISION.
OPEN-FILES.
   OPEN INPUT TIME-CARD-FILE, EMPL-MASTER-FILE-IN.
   OPEN OUTPUT PAY-REPORT, EMPL-MASTER-FILE-OUT.
   MOVE SPACES TO PAY-REPORT-REC.
READ-TIME-CARD.
   READ TIME-CARD-FILE AT END GO TO END-TIME-CARD-FILE.
READ-EMPL-MASTER.
   READ EMPL-MASTER-FILE-IN AT END GO TO NO-MATCH.
   IF TIME-CARD-SSN EQUALS EMPL-SSN-IN THEN
      PERFORM PROCESS-TIME-CARD
      GO TO READ-TIME-CARD.
   IF EMPL-SSN-IN LESS TIME-CARD-SSN THEN
      GO TO READ-EMPL-MASTER.
   IF EMPL-SSN-IN GREATER TIME-CARD-SSN THEN
      GO TO NO-MATCH ELSE
         MOVE "SYSTEM ERROR" TO PAY-REPORT-NAME
         WRITE PAY-REPORT-REC
         GO TO CLOSE-FILES.
PROCESS-TIME-CARD.
   MOVE EMPL-MASTER-REC-IN TO EMPL-MASTER-REC-OUT.
   MOVE EMPL-SSN-IN TO PAY-REPORT-SSN.
   MOVE EMPL-NAME-IN TO PAY-REPORT-NAME.
   PERFORM COMPUTE-WAGES THRU COMPUTE-WAGES-EXIT.
   ADD WAGES TO EMPL-YTD-WAGES-IN GIVING EMPL-YTD-WAGES-OUT.
   MOVE WAGES TO PAY-REPORT-WAGES.
   WRITE EMPL-MASTER-REC-OUT.
   WRITE PAY-REPORT-REC.
   MOVE SPACES TO PAY-REPORT-REC.
END-TIME-CARD-FILE.
   CLOSE TIME-CARD-FILE.
COPY-EMPL-MASTER.
   READ EMPL-MASTER-FILE-IN INTO EMPL-MASTER-REC-OUT
      AT END GO TO CLOSE-MASTER.
   GO TO COPY-EMPL-MASTER.
NO-MATCH.
   MOVE "NO MATCH TIME CARD" TO PAY-REPORT-NAME.
   WRITE PAY-REPORT-REC.
CLOSE-FILES.
   CLOSE TIME-CARD-FILE.
CLOSE-MASTER.
   CLOSE EMPL-MASTER-FILE-IN, EMPL-MASTER-FILE-OUT, PAY-REPORT.
   STOP RUN.

COMPUTE-WAGES.
   IF SALARIED THEN
      MOVE EMPL-RATE-IN TO WAGES
      GO TO COMPUTE-WAGES-EXIT.
   IF HOURLY THEN
      IF TIME-CARD-HRS LESS OR EQUAL 40 THEN
         COMPUTE WAGES = TIME-CARD-HRS * EMPL-RATE-IN
         GO TO COMPUTE-WAGES-EXIT ELSE
            COMPUTE WAGES = (40 + (TIME-CARD-HRS - 40) * 1.5) *
                            EMPL-RATE-IN
            GO TO COMPUTE-WAGES-EXIT ELSE
               MOVE "INVALID WAGE STATUS" TO PAY-REPORT-NAME
               MOVE ZERO TO WAGES.
COMPUTE-WAGES-EXIT.
   EXIT.
```

A

B

C

Figure 5: COBOL Program Example

a one-to-one correspondence of reports and program modules with absolutely no consideration for module grouping. In addition, failure to consider volume and frequency of access with regard to the various history data items defined eliminates any possibility for generating optimal file structures. Capabilities of computer-aided analysis should include specification of optimal file designs and grouping of single report generating modules in order to eliminate excessive transport of data from files to programs. These capabilities can only be achieved by extension of forms-oriented programming specification techniques like ADS to true requirements statement techniques providing supplementary volume and timing data to the optimization software. Then, the potential of the computer to aid the problem definer during the system design cycle can be fulfilled.

CONCLUSION

Experience with two problem statement languages ADS and SSL/I in the design of an information system for the U. S. Navy motivated the development of a forms-oriented front-end language called SSL/II to supplement the PSL/II language developed by the ISDOS Project.

ADS analysis and program module generation software are available on the CDC 6500 at Purdue University. Concurrent with the design of forms for SSL/II, implementation of software for SSL/II analysis is currently progressing.

REFERENCES

1. Nunamaker, J. F. Jr.; Nylin, W. C. Jr.; and Konsynski, B. 1972. Processing systems optimization through automatic design and re-organization of program modules. Proc. 4th COINS Conf., December 1972, to be published in 1973 by Academic Press.

2. Nunamaker, J. F. Jr.; Ho, T.; Konsynski, B.; and Singer, C. 1973. Specification and design of an information system using computer-aided analysis. CSD Technical Report, Computer Sciences Department, Purdue University, West Lafayette, Indiana (September 1973).

3. Canning, R. G. 1968. Data processing planning via simulation. EDP Analyzer 6, 4 (April 1968), 1-13.

4. Nunamaker, J. F. Jr. 1971. A methodology for the design and optimization of information processing systems. Proc. AFIPS 1971 SJCC 38, Montvale: AFIPS Press, 283-294.

5. McGee, W. C. 1963. The formulation of data processing problems for computers. In Advances in Computers 4, New York: Academic Press, 1-52.

6. Pridmore, H. D. 1967. The abstract information system concept and the problem of optimum design. Proc. Third Australian Computer Conf., Chippendale, N.S.W., Australia: Australian Trade Publications, 149-170.

7. Sammet, J. E. 1972. Programming languages: history and future. Comm. ACM 15, 7(July 1972), 601-610.

8. Benjamin, R. I. 1972. A generational perspective of information system development. Comm. ACM 15, 7 (July 1972), 640-643.

9. Merten, A. and Teichroew, D. 1972. The impact of problem statement languages on evaluating and improving software performance. Proc. AFIPS 1972 FJCC 41, Montvale, NJ: AFIPS Press, 849-857.

10. Teichroew, D. 1972. A survey of languages for stating requirements for computer-based information systems. Proc. AFIPS 1972 FJCC 41, Montvale, NJ: AFIPS Press, 1203-1224.

11. Young, J. W. Jr. and Kent, H. K. 1958. Abstract formulation of data processing problems. Journal of Industrial Engineering (Nov.-Dec. 1958), 471-479.

12. CODASYL Development Committee. 1962. An information algebra phase I report. Comm. ACM 5, 4 (April 1962), 190-204.

13. Langefors, B. 1963. Some approaches to the theory of information systems. BIT 3 (1963), 229-254.

14. Langefors, B. 1965. Information system design computations using generalized matrix algebra. BIT 5 (1965), 96-121.

15. Lombardi, L. A. 1964. A general business-oriented language based on decision expressions. Comm. ACM 7, 2 (Feb. 1964), 104-111.

16. National Cash Register Company, 1968. A Study Guide for Accurately Defined Systems. Dayton, Ohio.

17. Lynch, H. J. 1969. ADS: a technique in system documentation. Database 1, 1 (Spring 1969), 6-18.

18. Myers, D. H. 1962. A time automated technique for the design of information systems. IBM Systems Research Institute, New York (1962).

19. Kelly, J. F. 1970. Computerized Management Information Systems. New York: Macmillan.

20. Grindley, C.B.B. 1966. Systematics--a non-programming language for designing and specifying commercial systems for computers. Computer Journal. 9 (August 1966), 124-128.

21. Katz, J. H. and McGee, W. C. 1963. An experiment in non-procedural programming. Proc. AFIPS 1963 FJCC 23, New York: Spartan Books, 1-13.

22. Cattell, R. Duncan Electric Company, Lafayette, Indiana. Personal communication.

23. Koch, R. F.; Krohn, M. J.; McGrew, P. W.; and Sibley, E. H. 1970. PSL Version 2, Release 1: A PSL Language Primer. ISDOS Working Paper No. 33, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan (August 1970).

24. Hershey, E. A.; Rataj, W. J.; and Teichroew, D. 1973. PSL/II Language Specifications Version 1.0. ISDOS Working Paper No. 68, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan (February 1973).

25. Teichroew, D. and Sayani, H. 1971. Automation of system building. Datamation 17, 16 (August 15, 1971), 25-30.

26. Thall, R. 1973. A manual for PSA/ADS: a machine-aided approach to analysis of ADS. ISDOS Working Paper No. 35, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan (February 1973).

27. CODASYL Systems Committee. 1971. Feature Analysis of Generalized Data Base Management Systems. New York: ACM (May 1971).

28. CODASYL Data Base Task Group. 1971. CODASYL Data Base Task Group Report. New York: ACM (April 1971).

29. Codd, E. F. 1970. A relational model of data for large shared data banks. Comm. ACM 13, 6 (June 1970), 377-387.

30. Codd, E. F. 1971. Normalized data base structure: a tutorial. In Proc. 1971 ACM SIGFIDET Workshop: Data Description, Access and Control, New York: ACM (November 1971), 1-17.