# A FORMAL TECHNIQUE

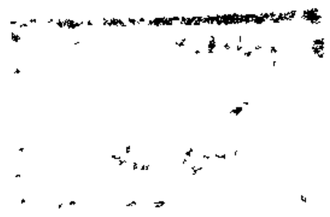
# FOR THE LOGICAL DESIGN

# OF ORGANISATIONAL INFORMATION SYSTEMS

ANSAR AHMAD KHAN M.A, M.PHIL

THESIS SUBMITTED FOR THE DEGREE OF D.PHIL AT THE UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE



APRIL 1984

# PAGE NUMBERING AS ORIGINAL

CONTENTS


CHAPTER 1: INTRODUCTION TO THE THESIS 1

1.1 INTRODUCTION 2

1.2 SUMMARY OF RESEARCH 8

1.3 AN INTRODUCTION TO THE METHODOLOGY (SSDM) 12

1.4 A BRIEF SUMMARY OF THE CHAPTERS OF THE THESIS 17

CHAPTER 2: HISTORICAL SURVEY 19

2.1 INTRODUCTION 20

2.2 DEVELOPMENT UP TO 1975 21

2.3 RECENT TRENDS FROM 1975 25

2.4 FUTURE PROSPECTS 27

2.5 CONCLUSION 29

CHAPTER 3: SURVEY OF METHODOLOGIES 31

3.1 INTRODUCTION 32

3.2 PREVIOUS COMPARATIVE SURVEYS 34

3.3 SYSTEMATICS 44

3.4 STRUCTURED ANALYSIS AND DESIGN 48

3.5 JSD 51

3.6 USE 56

3.7 NIAM 59

3.8 ISAC 62

3.9 CONCLUSION 67

CHAPTER 4: CLASSIFICATION OF METHODOLOGIES INTO CATEGORIES 69

4.1 INTRODUCTION 70

4.2 APPROACHES OBSERVABLE IN THE METHODOLOGIES SURVEYED 72

4.3 OTHER RELEVANT VIEWPOINTS 80

4.4 CONCLUSION 87

CHAPTER 5: SURVEY OF TECHNIQUES 88

5.1 INTRODUCTION 89

5.2 DIAGRAMMATIC REPRESENTATIONS OF FLOW OR PRECEDENCE 90

5.3 NON-DIAGRAMMATIC PROCESS REPRESENTATIONS 98

5.4 DATA REPRESENTATIONS 102

5.4 CONCLUSION 108

CHAPTER 6: REQUIREMENTS DESCRIPTION FOR A SYSTEM DEVELOPMENT METHODOLOGY 109

6.1 INTRODUCTION 110

6.2 PRINCILPLES WHICH SHOULD UNDERLIE A DEVELOPMENT METHODOLOGY 111

6.3 SUBSET OF METHODOLOGY ADDRESSED IN THIS THESIS 123

6.4 CONCLUSION 124

CHAPTER 7: GENERAL CONCEPTUAL MODEL OF THE METHODOLOGY 125

7.1 INTRODUCTION 126

7.2 CONTEXT MODEL 127

7.3 INFORMAL MODELS OF PRODUCT SYSTEM 129

7.4 FORMAL MODEL OF PRODUCT SYSTEM 133

7.5 DEVELOPMENT MODEL 140

7.6 GENERAL MODEL OF SYSTEM EVOLUTION 142

7.7 CONCLUSION 144

CHAPTER 8: APPLICATION OF THE SYSTEM MODEL TO AN EXAMPLE SYSTEM 146

8.1 INTRODUCTION 147

8.2 REQUIREMENTS DESCRIPTION FOR EXAMPLE INDIVIDUAL SYSTEM 147

8.3 SYSTEM MODEL OF EXAMPLE INDIVIDUAL SYSTEM 149

8.4 SAMPLE VERIFICATIONS FROM SYSTEM MODEL OF EXAMPLE SYSTEM155

8.5 SSDL REPRESENTATION OF THE EXAMPLE INDIVIDUAL SYSTEM 160

8.6 COMMENTS ON SSDL 168

8.7 CONCLUSION 170

CHAPTER 9: SOFTWARE TOOLS REQUIREMENT 171

9.1 INTRODUCTION 172

9.2 OVERVIEW OF THE TOOLS 174

9.3 DEVELOPMENT DIALOGUE PROCESSOR 175

9.4 ANALYSER 176

9.5 LOGICAL SIMULATOR 177

9.6 DEVELOPMENT DATABASE DECOMPOSER 178

9.7 CONCLUSION 179

CHAPTER 10: FUTURE AND RELATED WORK 180

10.1 INTRODUCTION 181

10.2 CHARACTERISTIC FEATURES FOR COMPARING METHODOLOGIES 182

10.3 MODELS 184

10.4 SYSTEM SPECIFICATION AND DESIGN LANGUAGE 185

10.5 SOFTWARE TOOLS 186

10.6 SEPARATE CONCERNS 187

10.7 OTHER ISSUES 188

CHAPTER 11: CONCLUSIONS 189

11.1 REVIEW OF PAST WORK 190

11.2 PROPOSALS FOR A NEW APPROACH 192

11.3 FINAL REMARKS 195

REFERENCES 196

APPENDIX A: FEATURES LIST FOR THE PRESENT SURVEY

APPENDIX B: SURVEY OF METHODOLOGIES

LIST OF FIGURES


Figure 3-1: output/recipient table 44

Figure 3-2: output and conditions 44

Figure 3-3: decision table 44

Figure 3-4: output items coding 45

Figure 3-5: item identification 45

Figure 3-6: derivation dictionary 45

Figure 3-7: given/derived items and decision table 46

Figure 3-8: primary identification dictionary 46

Figure 3-9: input set description 46

Figure 3-10: input dictionary 46

Figure 3-11: JSD data flow diagram 49

Figure 3-12: relational schema 49

figure 3-13: entity action list 52

Figure 3-14: entity structure diagram 52

Figure 3-15: specification and processes 53

Figure 3-16: structure text of function 54

Figure 3-17: system specification and schedular 55

Figure 3-18: USE data flow diagram 56

Figure 3-19: transition diagram 56

Figure 3-20: transition diagram encoding 56

Figure 3-21: database normalised relations 56

Figure 3-22: function specification 57

Figure 3-23: DBMS - Troll 57

Figure 3-24: module structure chart 57

Figure 3-25: module specification in PDL 57

Figure 3-26: specification in Plain 58

Figure 3-27: activities model 59

Figure 3-28: activity information list 59

Figure 3-29: high-level function description 59

Figure 3-30: functions relationship 59

Figure 3-31: decomposition of functions 60

Figure 3-32: NIAM information flow diagram 60

Figures 3-33, 3-34: NIAM ISDs 61

Figure 3-35: conceptual grammar in RIDL 61

Figure 3-36: problem table 62

Figure 3-37: list of interest groups 62

Figures 3-38, 3-39, 3-40: A-graphs and text pages 62

Figure 3-41: property table 62

Figure 3-42: table of objectives 62

Figure 3-43: needs for change 62

Figure 3-44: alternative study of change 63

Figure 3-45: formalizability text 63

Figure 3-46: special property table 63

Figure 3-47: prerequisites and requirements 63

Figure 3-48: cost-benefit analysis 63

Figure 3-49: I-graph 64

Figure 3-50: C-graphs 64

Figure 3-51: dictionary of terms 64

Figure 3-52: subsystem functions process list 64

Figure 3-53: process table 64

Figure 3-54: measurable subsystems features 64

Figure 3-55: D-graph 65

Figure 3-56: record type contents 65

Figure 3-57: D-structure diagrams 65

Figure 3-58: program/process description 65

Figure 3-59: control structures 65

Figure 3-60: raw program operation list 65

Figure 3-61: program structure 65

Figure 3-62: work task table 65

Figures 3-63, 3-64: E-graph 66

Figure 3-65: physical layout of input/output 66

Figure 3-66: physical layout of records 66

Figure 3-67: work descriptions 66

Figure 5-1: program flow chart 91

Figure 5-2: structure diagram 91

Figure 5-3: system run chart 92

Figure 5-4 (a), (b): data flow diagram 92

Fifure 5-5: procedure graph 93

Figure 5-6: Jackson structured diagrams 94

Figure 5-7 (a): Petri net graph 95

Figure 5-7 (b) dynamic petri net 95

Figure 5-7 (c): modelling of computer systems 95

Figure 5-8: decision trees 97

Figure 5-9: decision table 99

Figure 5-10: pseudo code 100

Figure 5-11: formal logic 101

Figure 5-12: relational schemas 103

Figures 5-13, 5-14: conceptual schemas 104

Figure 5-15: Bachman diagram 105

Figure 5-16: identification matrix 106

Figure 5-17: data abstraction 106

Figures 5-18, 5-19: manual and automated data dictionaries 107

Figure 7-1: context model 127

Figure 7-2 (a): informal model of product system
with functional structure 129

Figure 7-2 (b): informal model of product system
with subsystem structure 132

Figure 7-3: development model 140

Figure 7-4: General model of system evolution 142

Figure 8-1: Individual system model 149

Figure 9-1: Software tools model 174

ACKNOWLEDGEMENTS

I am indebted to my supervisor, Mr. C. J. Tully for his continuing help and encouragement from my joining the department until the completion of the thesis. I most particularly thank him for his keen interest and insistence that I explore and understand most of the notable existing methodologies, research problems and other relevant literature which served as a knowledge base of concepts for proposing a new methodology. He also continually assisted me in the development of the proposed methodology. Apart from this I am also grateful to him for taking a keen interest in my other problems as a foreign student, which enabled me to continue with and concentrate on the research.

I am grateful to the secretarial staff and all the members of the teaching staff, specially Dr. I. C. Wand and Prof. I. C. Pyle, for their encouragement and very friendly treatment throughout my studies in the department.

I also wish to thank NED University of Engineering and Technology, Karachi, and the Ministry of Education, Pakistan, for granting leave and awarding the scholarship, which together made it possible for me to pursue the research.

Finally I must express my appreciation of all the help given to me in the preparation of this thesis, in particular Mr. R. P. Whittington, a research collegue, who discussed the topics for refinement, and who has given me, together with my wife and children,

every encouragement and assistance in my studies and research.

## ABSTRACT

A complete information system, when conceptually modelled, always comprises a virtual database and a set of derivation processes. It may be decomposed into subsystems; the first level subsystems into which it is decomposed may include intermediate sub-systems (capable of being further decomposed) or elementary sub-systems (not further decomposed) or both. An information system always receives inputs from its environment and provides outputs to it; these comprise its external interface.

The development of an information system involves its initial creation, its application usage, and its evolution. The development process is complex, and its efficiency (in terms of both the quality and the efficiency of resultant systems) has a significant effect on all users of the information system. It is therefore desirable that a complete, consistent, coherent and formal framework be made available for guiding and supporting that class of people who are involved in information system development. Such a framework is termed a methodology, and the class of people as system developers. A methodology permits the unambiguous specification of information systems through formal models and languages. Further to this, a methodology has associated software tools which assist the developer in producing and maintaining documentation, and in verifying and carrying out other operations on the system specifications.

Just as a system developer investigates the particular activities of people in a particular organisation, generalises them and specifies

and designs a target system to be embedded within that organisation, so this thesis investigates the particular activities of system developers, generalises them, and specifies and designs a special kind of target system to be embedded in their (developers,) development system.

The proposals made in the thesis, which together specify such a methodology for information system development, are summarised as follows.

1. A development context which captures the purpose and scope of the methodology and its relationships with other methodologies.

2. A formal conceptual model of the information system development process which encapsulates the worlds inhabited by system developers. The model constitutes a generalisation of these worlds as perceived by the developer, and provides a basis for the capture of information system structures and processing.

3. A system specification and design language SSDL permits the developer to make necessary and sufficient statements about a target system, based on the formal conceptual model. This language enables a developer to specify and design information systems throughout their development stages.

4. A set of software tools which will operate on statements in that language, and assist the developer in producing systems of higher quality and/or in less time.

Although the proposed methodology (SSDM) is under development, the proposals of this thesis are argued to be original and significant. The originality stems from a rigorous conceptualisation of information systems and their development, an exercise characterised by both comprehensiveness and flexibility. The significance of the recommendations is claimed to be their collective provision of a basis for a system of development which offers users information systems of unprecedented effectiveness.

# CHAPTER 1

# INTRODUCTION TO THE THESIS

## CONTENTS

1.1 Introduction

1.2 Summary of research

1.3 An introduction to the methodology (SSDM)

1.4 A brief summary of the chapters of the thesis

## 1.1 INTRODUCTION

The ultimate measure of the success of an information system development methodology is the extent of the general improvement in the resultant target systems. A developer needs better methods to produce target systems which meet the organisation's requirements, are delivered on time and to budget, and are reliable and adaptable. Management are not receiving the information they require and they cannot have changes made within a reasonable time. Systems do not meet their requirements and have errors in them. Predicted trends, as described in MACDONALD (1983) and BODART (1983), may be summarised as follows:

- user demands and dissatisfaction will rise even more, generating an increasing application backlog;

- improvements in technology will be of little relevance (ie. we are solving the wrong problems);

- conventional methodologies are obsolete and will not cope.

In most organisations, however well managed, the admitted backlog is between two to four years and is still growing. ALLOWAY and Quillard (1982) estimated that a hidden backlog of about 168% of that on record exists, because users no longer even voice their requirements.

ALLOWAY (1982) also discovered that user managements are asking for six times as many analysis systems to support decision making, three times as many query systems for flexible inquiry and reporting and

twice as many exception reporting systems as are currently installed. Data processing staff generally implement the kind of systems they have built before, because they feel confident with them.

One important way in which a methodology should help is to speed up system development. In order to control the system development process, most of the existing methodologies concentrate on rigid documentation and the administration of development tasks. They are not capable of adapting to new styles and theories of accelerated development, which particularly emphasise the use of software tools.

The following are the essential objectives which must be met.

- User management must be involved in defining organisational needs and priorities, and also in the subsequent approval and review of systems.

- There must be good communication between end users and system developers.

- The evolution of the information systems of an organisation must be linked firmly with its business goals, objectives and priorities.

- New developments (eg. in computing power, user languages, and communications) must be exploited to bring about more effective systems.

- A complete, comprehensible, coherent, flexible and formal methodology must be available to gain control over information

system development.

Achieving the last of those objectives is critical to the achievement of the others. Among the features suggested in the literature as being important for a system development methodology are the following.

- Maximum machine assistance should be made available to system developers. In particular, all information relating to system development should be maintained in a development database.

- Maximum use should be possible of techniques (eg. prototyping, code generation) to shorten development lead time.

- It should be possible to modify systems with the maximum speed and ease and minimum probability of error.

- There should be the maximum capability for verification at each stage of system evolution.

- Users should be able to check at each stage of system evolution.

- It should be possible to use the methodology for the development of new styles of systems (eg. decision support systems, enquiry systems, expert systems) as well as conventional systems, and to enable several styles to be contained in a single system.

- The methodology should reduce and simplify the developer's work rather than increasing and complicating it.

- The methodology should permit diversity of design styles (eg.

top-down or bottom-up, data-oriented or function-oriented, entity-oriented or event-oriented, etc.).

- The methodology should permit the use of a diversity of individual techniques (such as diagramming and tabular techniques), where this is possible without sacrificing coherence.

- Software tools embodying the concepts of a methodology should constitute an integrated support environment for system developers, users and project management.

Comparing the above requirements with previous proposals for the management of information system development, it is argued that no existing methodology goes far enough in supporting the development process, and, consequently, in serving the user. Even extensions of existing methodologies would be inadequate, because they are based on inadequate models of systems and of the development process.

We have used the term information system to mean a computer-based system which receives information from and transmits information to human beings working in an organisation. There are certain differences between information systems and products such as operating systems, compilers or real time (embedded) systems. Systems of this latter kind interface largely with equipment (such as monitoring or control gear, radar etc). It is increasingly the case, however, that they have characteristics in common with information systems, and it is to be expected that methodologies for their development may share

common features with information systems development methodologies.

Attempts to develop tools, techniques and methodologies, to assist the designer throughout the development life cycle, have proliferated during the past decade. The following statements are believed to be true for such attempts.

- They have been confined either to information systems (interactive or batch), or embedded systems.

- They have covered varying stages of the complete life cycle.

- They have been based on inadequate or non-existent models of the life cycle.

- They have been based on inadequate or non-existent models of the class of systems to which they relate.

- They have been based on varying viewpoints (e.g. programming languages, databases, mathematical modelling, project management, etc.).

- They have in various ways been unfriendly to their users (i.e. system developers).

- They have not contributed to significantly improved correctness or reliability.

- Communication between users and developers has not been significantly improved.

- Developers are uncertain about the amount of testing and checking required; frequently redundant tests/checks are conducted which are costly or totally ignored.

## 1.2 SUMMARY OF RESEARCH

The research undertaken assumes that a system specification and design methodology (SSDM) should operate at three levels to support the developer:

(a) through the provision of a model (or conceptual framework), in terms of the activities involved and their relationships,

(b) through the provision of a language (system specification and design language, or SSDL) to allow the expression of the results of development activities,

(c) through the provision of a set of software tools, which supports the developer in decision making, evaluation, verification and documentation management.

The model determines how one thinks about systems that are to be specified and designed, and the process of specification and design within the complete life cycle. The language enables the designer to record specifications and design decisions made in accordance with the model. The tools enable the designer to manipulate the statements in the language ( to perform, say, checks, decisions, and inferences on them) and thus receive machine assistance which makes the process of specification and design more effective.

The research undertaken falls into two parts which are described as follows.

PART 1 extensive and detailed review of existing work in the field (chapters 2 to 6, appendices)

A new and improved "features list" is presented for comparing methodologies, which has been prepared after surveying several such sets of features. A survey of a large number of methodologies (larger than any other survey) is presented, based on this features list. An attempt has been made to identify a number of approaches which underlie existing methodologies, or which are potentially relevant to future ones. Finally, there is a review of individual techniques relevant to system development, grouped into three simple categories.

PART 2 proposal for a new methodology (chapters 7 to 11)

It is argued that a unified theory in terms of models is necessary. The validity of the models can be established by using them as the basis of a development language and a set of software tools. Although work could be developed in the long term to the point of achieving fully usable products, the primary purpose of this research is to show how a set of models can be developed and described to serve as the basis both for the evaluation or comparison of existing methodologies and for developing new and improved methodologies.

The formality, flexibility and rigour inherent to the proposed models make possible the proposal of a single and powerful system specification and design language (SSDL). This SSDL has capabilities

(both semantic and syntactic) for defining target systems in terms of objects and their properties; domains and restrictions on permissible states of the objects; derivation rules; inputs and outputs.

The availability of such a SSDL makes possible the specification of a set of software tools constituting an integrated support environment for the system developer. The set of software tools specified consists of: a development dialogue processor, an analyser, a logical simulator and a development database decomposer.

The substance of the second part of the research is therefore summarised as follows.

(a) An informal description of requirements for an improved methodology has been presented, which is based on the standard features derived from the survey of methodologies, to serve as the foundation for building the conceptual model of the improved methodology proposed in the thesis.

(b) An overall original conceptual model of the proposed methodology, and an easy, concise and structured notation to describe the model, have been presented.

(c) A new and improved language for system specification and design has been presented on the basis of noting the strengths and weaknesses of such languages during the survey of methodologies. This language has the capability of being accessible to people with a variety of backgrounds and for describing systems of a variety of categories. It offers an

economy to the designer in making statements and ease both in writing and reading.

(d) A functional specification of a set of software tools which will constitute an integrated system development environment has been presented.

## 1.3 AN INTRODUCTION TO THE METHODOLOGY (SSDM)

The research work undertaken proceeds from the survey of existing methodologies, noting their weak and strong points, to suggest an improved computer-based information system development methodology.

The development model of the life cycle proposed in the thesis is based on a recognition of an interplay between specification and design activities, and is described as follows.

1. System development is initiated by a requirements description. It describes what the user (client) wants. Both the user and the developer may be involved in writing this document, but the user must understand it clearly. It will therefore be in natural language, and may be incomplete and contradictory, and contain much material which is not directly relevant to the system developer. This requirements description will be subject to repeated updating throughout the subsequent stages of system development.

2. If a requirements description describes what a user wants, a specification describes what he will get. Specifications will be written by the system developer, in a formal language. It will be machine processable, and subject to automatic checks for completeness and consistency. It is for the developer's subjective judgement to decide to what extent a specification matches the requirements description.

3. Corresponding to the specification of what a system will do, a

design describes how it will be structured to do it.

4. A requirements description is developed at one level only - the level of the complete system - but the specifications and designs are produced not only for the whole system but also for each of the levels of subsystems into which it may be decomposed. Design at one level yields specifications for the next lower level.

5. Specifications can generally be subjected to verification for logical consistency and completeness in two ways: "horizontal" (i.e. internal verification of a single specification), and "vertical" (i.e. verification of a set of specifications at one level against the parent specification at the next higher level).

6. The language (SSDL) in which specifications and designs are expressed is designed to be usable by people of a variety of backgrounds, since system developers vary a great deal in terms of their academic discipline and their past experience.

7. Among the software tools already mentioned is one called a "logical simulator". While this will be discussed at more length later, it is important at this stage to note that its purpose is to provide a feedback channel which will enable users to confirm that the developers have correctly captured their requirements in the formal specifications.

8. The final output of the process of logical specification and

design is a complete and consistent set of specifications for subsequent physical design of (a) the target system database, (b) the set of internal and external interfaces, (c) the programs which are the ultimate embodiment of the system logic. These specifications serve as the starting points for specialised processes of software - assisted design, one of which (for databases) has been dealt with by my research colleague WHITTINGTON R P (1982, 1983).

A great deal of work has been undertaken in the last twenty years which is relevant to the problem area addressed by this thesis. Three approaches seem especially appropriate, and have excercised great influence on the research reported in this thesis.

(1) The relational data model.

(2) Automated data dictionaries.

(3) Systematics, GRINDLEY (1975), which is directed toward logical system design and has an appealing simplicity.

The proposed methodology has the following characteristics.

- It concentrates efforts in the earlier stages of development, and gives much greater opportunity for verifying logical completeness and consistency. As a result errors should be less likely to be introduced, and should be detected earlier.

- It is applicable to a wide range of application systems.

- It is employed from the moment when the formal specification is first drawn up, and consistently thereafter.

- It supports as much variation as possible in the sequence of life cycle events, recognising that individual systems may justify different approaches and individual designers may demand them. There must, notwithstanding, be some clear general life cycle framework.

- It is recognised that a methodology is likely to be incomplete and does not occur in a vacuum, but must have a well defined context within a broader if less precise methodology for system development.

- The conceptual model identifies a minimal set of concepts which are necessary and sufficient to describe the essential features of a system completely and precisely.

- It has the capability to evolve over time in accordance with developing technology and experience.

- It does not prescribe a particular project management system or set of documentation standards.

Because of the broad scope of this subject matter, and the fact that it begins from first principles, the implementation of software tools will require considerable further effort. It is suggested that the work presented is a sufficient contribution to the understanding and development of the field of study in its own right, and indeed that it

offers considerable possibilities for further work of both a theoretical and implementational nature.

## 1.4 A BRIEF SUMMARY OF THE CHAPTERS OF THE THESIS

This thesis is presented in eleven chapters and two appendices. The sequential development is as follows.

- It presents a historical review of evolutionary improvements in the development of methodologies, and a survey of the current trends in the 1980s.

- It reviews published surveys of methodologies, and presents a study of six representative methodologies.

- It presents a classification of broad approaches, or viewpoints, which can be seen to underlie existing methodologies, together with others which could be valuable for future methodologies.

- It presents a survey of techniques to note their suitability for application in a good methodology.

- On the basis of the above description, the need for an improved methodology is argued.

- A conceptual model is proposed on the basis of the preceding arguments.

- It presents an application of the model to an example individual system and shows the transformation of a system schema into a structured matrix which facilitates several types of checkings, analyses, inferences. It also presents comments on "system specification and design language" (SSDL).

- A software toolkit is specified in outline to assist the developer in developing his system.

- Future related research is specified.

- Conclusions are presented.

- There are two appendices. Appendix A defines the feature list used for the comparative survey. Appendix B contains a survey of fortythree methodologies based on the feature list in appendix A.

The above sequence can be divided broadly into two main parts (corresponding to the two main subdivisions of the research work undertaken, as described earlier in this chapter). The first part presents a requirements analysis for a new methodology, and consists of chapters 2, 3, 4, 5, 6 and appendices A and B.

The second part describes the proposed methodology in terms of a conceptual model, a language (SSDL), and a set of software tools; it outlines future work and presents conclusions. It consists of chapters 7, 8, 9, 10, and 11.

# CHAPTER 2

# HISTORICAL SURVEY

## CONTENTS

2.1 Introduction

2.2 Development up to 1975

2.3 Recent trends from 1975

2.4 Future prospects

2.5 Conclusion

## 2.1 INTRODUCTION

The objective of the survey which constitutes the first part of this thesis (chapters 2 to 5) is to lay the basis for the requirements description of an improved and original methodology for developing information systems. This requires an understanding of past proposals and of present predictions of requirements for information systems and for their development.

A detailed summary of the main features of many existing methodologies is presented in the appendices. The survey includes both products in commercial use and projects currently under development in universities and software organisations. A small number of methodologies are studied in greater detail in chapter 3. The aim of this chapter is to give an introductory account of existing methodologies in their historical context.

The survey by COUGER and Knapp (1974) is based on an unsatisfactory historical framework, and includes a good deal of not very relevant material. The survey presented in this chapter attempts to provide a straightforward account partitioned simply into two periods, before and after 1975. Like any historical dividing line, the choice of 1975 is to some extent arbitrary; yet it can be observed to be a fairly clear boundary after which there has been a rapid growth of interest in methodologies and a significant increase in the sophistication of approach.

## 2.2 DEVELOPMENT UP TO 1975

In the early years of computing (roughly corresponding to the so-called "first generation"), the emphasis of universities and early manufacturers was on the invention and improvement of hardware, and the emphasis of the few users (mostly scientific) was on identifying possible applications and on the details of programming. There was no concept of what is now called "system analysis and design", even among the very few early "commercial users".

Although certain graphic and descriptive techniques existed in the fields of work study, O and M, punched cards and tabulating systems, these techniques were hardly if at all used in early commercial applications, because of the understandable preoccupation with the difficult and fascinating task of programming. The most that was done to combat the difficulty of understanding complicated machine code programs was to annotate coding sheets.

As volume production of computers grew, and commercial applications spread, the most common tendency was for users to try to reproduce existing applications on computers, rather than redesigning them. In the USA, where punched card techniques were more widely used than in the UK, this resulted in systems which comprised a large number of small programs. In the UK the applications being replaced were more likely clerical in nature and therefore both less well defined and made up of larger grouping of functions: for these reasons perhaps more attention was given to the design of efficient systems in the UK than in USA. But even so it remained true that the primary emphasis

was in analysing the way in which things were already done, rather than in designing completely new systems. This accounts for the birth of the term "systems analysis" around 1960.

Another factor accounting for the lack of good design was the dependence of users on computer manufacturers, and their acceptance of manufacturers' attempts to offer standardised solutions and to suggest that system development was not a major problem.

In user companies, only the most primitive methods of charting, decision tables, other tabular methods and narrative descriptions were available for the new task of systems analysis. A very few people, working in isolation, attempted to develop theories which they hoped would lead to the design of better systems, either because systems might be described more formally and therefore be better understood, or because some aspects of system performance might be optimised. These efforts included Information Algebra (CODASYL, 1962), Young and Kent Algebra (1958) and Langefors Algebra (1963).

The mid-1960s saw the introduction of IBM's system/360, marking what is often called the third generation of computers. This was typified by a degree of maturity in hardware, and much greater effort (and success) in the provision of system software. Computer manufacturers recognised how much effort users were having to devote to system development, and tried to offer methods in this field which would improve user productivity (in the same way that programmer productivity was being improved by high-level languages) and assist in their sales. Examples of such efforts include ADS, TAG, BISAD and

HIPO. Consultancy firms, together with the management services departments of very large users, soon began to develop so called "life cycle" concepts; these were aimed at improving management control over system development, and applied the traditional "scientific management" approach of breaking a big task down into many smaller, well-defined sub-tasks. This approach, however, was generally perceived as imposing a bureaucratic burden on system development and was not widely accepted.

Manufacturers also recognised the scale and difficulty of the task of developing their own system software and tried to develop in-house support tools (e.g ICL's CADES). Although the problem of developing system software is different from the problem of developing application software, there is some overlap; manufacturers could have tried to adapt their methods to the user community but they did not do so.

The growth of specialist consultancy services and of software houses from the late 1960s opened up a possible alternative to the computer manufacturers as a source of system development techniques; by their nature, however, these companies tended to be involved in "advanced" or "state-of-the-art" applications, and not to be closely involved in what they saw as the more mundane problems of information systems. Computer science in universities also paid little attention to the problems of information systems development, with a few exceptions, e.g. ISDOS (Teichroew, 1977) and CASCADE by Solvberg (INFOTECH 1975); (note that both projects had started much earlier than the publication

dates). These, however, were perhaps over-ambitious and founded on inadequate theory. There were very few attempts to provide theoretically sound approaches; one was Systematics (Grindley, 1966, 1972).

Something which was to prove of great significance was the development from 1973 of the relational model, which provided a coherent and powerful theory of data. While much of the work which has since been done in this field has been too narrowly academic, relational theory has had a deep impact on ways of thinking about information systems and is likely to continue doing so.

## 2.3 RECENT TRENDS FROM 1975

Since about 1975, a number of general trends have been observable in computing which have had, or are likely to have, important influences on system development methodologies. They include: renewed interest in programming languages including particularly languages for logic programming, of which several have been developed; widespread development of operating systems and application packages for micros; the development of primitive programming support environments; the spread of DBMS, in particular those based on the relational model and incorporating query languages; the diversification of technology and its penetration into all areas of applications and sizes of organisation; the spread of word processing and the introduction of primitive office automation; the (largely experimental) introduction of expert systems; a steadily increasing shift in total data processing expenditure from hardware to software; and a similar shift in expenditure from software development to software maintenance, and from in-house software development to packages.

There has also been a growing recognition of a "systems crisis", comparable to the earlier recognition of the "software crisis".

The growth of the world economy in recent years has generated enormous demands for data processing systems and services. In order to maintain orderly economic and technical development of the data processing industry, a number of conditions, including management awareness, a substantial improvement in total data processing quality, reliability and security, increased cooperation between the industry

and the higher education system, and a sound program of standards development, are recognised to be necessary.

There has been a corresponding explosion in systems methodologies. This comes from leading consultancies and software houses, from the advanced state-of-the-art users, and from the academic community (despite the continuing dominance of traditional computer science and the relative weakness of information systems studies). Compared with earlier efforts there is a much greater emphasis on software tools to assist the system developer.

The majority of the methodologies reviewed in this thesis have originated in this period and it is therefore not appropriate to give a long list of them here. They reflect a wide diversity of viewpoints on the part of their developers. On the whole they do not show signs of being based upon a coherent conceptual model of the system development process. Nevertheless such a diversified and pragmatic approach is to be expected at this stage on the growth curve of a new technical development.

This "generation" of methodologies reflects a recognition that system development is evolutionary and incremental in nature, that it is not confined to mainframe computers, and that it must take into account developments such as office automation, expert systems, knowledge bases, decision support systems and end-user system development.

## 2.4 FUTURE PROSPECTS

Although the fifth generation has been under discussion for the last few years, since the concept was introduced by the Japanese, and although some significant programs of research are under way in the leading western countries, the fifth generation cannot yet be said to have arrived. The main characteristics expected of the fifth generation are availability at affordable cost of very great computing power through VLSI; software capability to go much further in emulating human intelligence using new styles of information representation and of programming (for which the developments in VLSI are essential prerequisites); and a quantum leap in the accessibility of computer systems to ordinary people through the engineering of much better interfaces (particularly involving speech handling).

Although the cost of VLSI development is recognised to be high, it is realised that the main problem will lie in the field of software. Great emphasis is therefore being placed on the need for much more effective programming support environments. On the whole, inadequate attention is being paid both to the useful applications of this advanced technology and to the systems-level problems that will be encountered in developing such applications.

There has been little if any discussion of the way in which methodologies will adapt in parallel with the technological changes of the fifth generation. One can express the following hopes.

1. Methodologies will be based on sound models of the system

development process.

2. There will be some convergence of these models and therefore of methodologies.

3. There will be relatively greater emphasis on understanding the information needs of the organisation as opposed to just understanding the problems of developing machine-based systems.

## 2.5 CONCLUSION

In a rather imprecise and unquantified way, one can suggest that software progress has lagged one generation behind hardware, and that systems progress has lagged one generation behind software. This generalisation can be roughly supported in at least the following two ways.

1. Computer hardware was developed out of its immediate precursors (e.g. ENIAC, Mark 1, Colossus, differential analysers) in the first generation; computer software was developed out of its primitive beginnings (loaders) in the second generation; and the first attempts at coherent solutions to the systems development problems occurred in the third generation.

2. Hardware arrived at a stage of relative maturity, after a period of excessive diversity and confusion, and offering a base for subsequent steady evolution, in the third generation; software, after the recognition of the software crisis, reached a similar stage in the fourth generation; and it is to be expected that systems development methodologies and techniques will also achieve maturity and stability in the fifth generation.

It has to be said, however, that information systems users, throughout the whole of the historical period surveyed in this chapter, have suffered (a) from a rate of technological change which has been excessive from their viewpoint, and which has placed them under continual pressure to adapt to external technical factors, (b) from

the dominance of the supplier side in determining broad strategies of use of the technology. These factors, together with (as already noted) the weakness of information systems studies in universities, may largely account for the lag described above; but these factors are equally unlikely to abate in the near future. In particular, one can anticipate that users (and therefore developers of methodologies) will have to accommodate the effects of considerable diversification through the spread of office automation, expert systems, end-user involvement and so on, and that it will require a lot of determined effort to overcome the strong technology-oriented drive already apparent in the fifth generation.

# CHAPTER 3

# SURVEY OF METHODOLOGIES

## CONTENTS


3.1 Introduction

3.2 Previous comparative surveys

3.3 Systematics

3.4 Structured design

3.5 JSD

3.6 USE

3.7 NIAM

3.8 ISAC

3.9 Conclusion

## 3.1 INTRODUCTION

This chapter does two things. First, in section 3.2, a summary is given of a number of published comparative surveys of systems development methodologies, indicating ths scope of each in terms of the number of methodologies surveyed and the features used for comparison. Then, in sections 3.3 to 3.8, six selected methodologies are described. The description is at what might be called a "detailed summary" level - that is, in much more detail than is possible in appendix B of this thesis, but in much less detail than in the published accounts (and omitting their extensive coverage of examples). The detailed summaries are confined to the significant steps of each methodologies, accompanied in each case by figures illustrating the kind of documentary output from each step.

The following reasons led to the selection of the six methodologies for detailed summary.

- They are all fairly (or very) well-known and influential, in the research community or in the practitioner community or both.

- Three of them are in common use (Structured design, JSD, ISAC); the other three are important sources of ideas of varying kinds (Systematics, USE, NIAM).

- Three of them were included in CRIS 1 (USE, NIAM, ISAC); the other three were not (Systematics, Structured design, JSD).

- Three of them are available commercially, appropriately packaged

with books and training courses, etc. (Structured design, JSD, ISAC); the other three are not (systematics, USE, NIAM).

The purpose of the detailed summary of these six methodologies is to give an idea of what is offered by a cross-section of the best of what is currently available and under development, and also to indicate the diversity of approaches and styles adopted.

In the appendices, a much larger number of methodologies is surveyed in less detail. Appendix A describes the features used for this survey, and appendix B is the survey itself.

## 3.2 PREVIOUS COMPARATIVE SURVEYS

Twentyone published surveys are identified, some of which are associated with CRIS 2. CRIS 2 was part of the comparative review of information system design methodologies carried out by WG8.1 of IFIP and reported in OLLE (1983). A very brief summary of each of these surveys follows.

(1) TEICHROEW (1970)

Methodologies surveyed: 6.

Features included: problem statement, life cycle phases covered, objectives.

(2) TEICHROEW (1972)

Methodologies surveyed: 7.

Features included: problem form input, problem form output, data relationships, computational relationships, notation used, other information.

Comment: (1) and (2) are similar. Features lists are brief. Surveys are mostly concerned with system specification and design and with justifying PSL/PSA.

(3) STRUNZ (1973)

Methodologies surveyed: 7.

Features included: analysis of the problem; design, implementation, application area.

Comment: The survey is insufficient and does not reflect all

aspects of a development methodology.

(4) LUDEWIG (1978)

Methodologies surveyed: 14.

Features included: specification of tools, specification of methods, range of aids within the system life cycle, kinds of tools, language used, types of software for which the aids are designed.

Comment: The features mostly cover the classical life cycle, software development tools, and real time software development systems.

(5) BREWER (1979)

Methodologies surveyed: 13

Features included: systems survey, systems evaluation, systems specification, systems programming, systems implementation.

Comment: The features are based on conventional life cycle stages. The survey does not provide any new concepts or useful ideas to the developer.

(6) DoI (1981)

Methodologies surveyed: 21.

Features included: summary, life cycle coverage, notation used, procedures, automated tools, checking, configuration control and maintenance,

experience, applicability and transferability.

Comment: The survey provides an improved set of features depending on the classical system development life cycle. Concentration is mostly on software development (real time systems) and Ada applications. The survey is brief and does not cover all aspects of a complete methodology.

(8) FREEMAN AND WASSERMAN (1982)

Methodologies surveyed: 24.

Features included: identification, general methodology issues, technical aspects, automated support, management aspects, usage aspects, transferability.

Comment: The features have a broader coverage and are not based on a particular system development life cycle. They mostly concentrate on software development, and ignore environment considerations. The survey deals mostly with those aspects which are relevant to the Ada programming language.

(8) TSE T H (1982)

Methodologies surveyed: 6.

Features included: goals, user verification, file design, process design and optimisation, maintenance.

Comment: The survey mainly concentrates on classical ideas of file design, optimisation and maintenance problems, and

ignores many important concepts of an information system development methodology.

(9) ASPROTH AND HAKANSSON (1983)

Methodologies surveyed: 13.

Features included: applicability of method, service measurement, phases of system design covered, role of the end users, condition and results, documentation.

Comment: The survey deals with a limited number of features. It recognises end-user participation, and deals with the efficiency aspects of system design by mathematical notation; but it ignores many important concepts of an information system development methodology.

(10) BODART AND OTHERS (1983)

Methodologies surveyed: 4.

Features included: abstraction problems, decision problems, control problems.

Comment: The survey considers only three features; while they are each potentially very broad, they are in fact considered in a fairly limited way.

(11) BRANDT AND SOLVBERG (1983)

Methodologies surveyed: 13.

Features included: origin and experience, development process,

model, iteration and test, representation means, documentation, user orientation, tools and prospects, comments.

Comment: The survey deals with some important aspects, e.g. models, notations, user participation and tools. Modelling concepts are confined to the E-R and relational models.

(12) FALKENBERG (1983)

Methodologies surveyed: 4.

Features included: brief description of methodology, major principles and concepts, weak points of methodology, suggestions for improvement.

Comment: The survey concentrates mostly on the weak points of methodologies and suggestions for improvement. It lacks the provision of a uniform and precise set of features, and gives poor coverage of modelling and environmental concepts.

(13) GLASSON AND HODGSON (1983)

Methodologies surveyed: 13.

Features included: background study, systems requirements, systems design, systems specification, program design, system implementation, systems maintenance and evaluation.

Comment: The survey offers a fairly detailed treatment strictly in

relation to the classical system development life cycle. It is influenced by Hawryszkiewkis criteria for evaluating design. It is weak in modelling, abstraction and other modern concepts.

(14) IIVARI AND OTHERS (1983)

Methodologies surveyed: 4.

Features included: set of eighty-five questions which are mainly concerned with theoretical interest, measurability and answerability, structurability, neutrality.

Comment: The complicated and lengthy set of eighty-five questions make the survey difficult to understand. The features are based on a sociocybernetic approach. It is weak in modelling concepts.

(15) KUNG (1983)

Methodologies surveyed: 3.

Features included: understandability, expressiveness, processing independence, checkability, changeability.

Comment: The survey considers temporal aspects of modelling in a limited sense. Only a few features, though well structured, are used, and a very small number of methodologies is compared. It is rather general in its approach.

(16) MADDISON (1983)

Methodologies surveyed: 7.

Features included: scope, objectives and deliverable products, philosophy and assumptions, pre-requisites and starting points, life cycle phases, maintenance, application.

Comment: This is a fairly detailed and critical survey, covering a relatively small number of methodologies in depth. Sometimes the analysis becomes inconsistent with the features, perhaps because of the number of different authors.

(17) MALMBORG (1983)

Methodologies surveyed: 9.

Features included: specification of static and dynamic universe of discourse, specification of static and dynamic environment, specification of static and dynamic information systems.

Comment: The survey deals with a limited number of features at a high level of abstraction.

(18) MOULIN (1983)

Methodologies surveyed: 13.

Features included: simplicity of concepts and techniques, usability of methods, completeness, role of users and analysts, software aids, graphic

aids, language for system description, nature of pre-requisites necessary to use the methods surveyed, concepts and techniques.

Comment: The survey provides a fairly good set of features based on the traditional system development life cycle, although the details of the features are not always clear. It concentrates more on philosophical aspects and less on technical aspects.

(19) NISSEN (1983)

Methodologies surveyed: 3.

Features included: specification of some part of the world outside computerised part; knowledge/ignorance, actual or potential, of some part of the world mediated by the computerised parts of information system and knowledge about the access of knowledge by users; design of formal systems to support knowledge of the outside world to become mediated between people; design/implementation and choice of physical systems.

Comment: This gives a fairly detailed treatment of aspects of the universe of discourse and of perceived entities, with some empasis on user participation. There is little emphasis on lower level considerations of system development. It provides some useful concepts and ideas

about system development aspects in relation to the real world.

(20) OLIVE (1983)

Methodologies surveyed: 11.

Features used: levels of abstraction, model description, type of information system.

Comment: Features are mostly based on the concepts of Young and Kent Algebra (1958), Langefors Algebra (1973) and Systematics (Grindley, 1972, 1975). The features are limited and insufficient for a complete survey of modern system development methodologies.

(21) SWIGCHEM AND ESSINK (1983)

Methodologies surveyed: 10.

Features included: scope of the method, levels of abstraction, object system modelling, aspects of information system modelling, decomposition, validation, role patterns, communications and learning, automatic tools.

Comment: The feature list consists of a mixture of classical and structured life cycles and concepts of modelling. The analysis of methodologies using the features is sometimes not clear. The black box matrix technique used for specifying relationships between entities is not convenient - especially for large systems.

From the above brief analysis, we can conclude that there is very great variation in the features lists used in the different surveys, and that individual surveys are based on limited and insufficient sets of features. Any individual set of features is not representative of modern information system development requirements. Therefore, a fuller set of features is required. The set of features proposed for the purposes of this thesis is given in appendix A.

## 3.3 SYSTEMATICS

System specification in Systematics is carried out in the following seven steps. (Note that these steps are a synthesis of the steps explicitly or implicitly stated in the book, which is less than completely clear about the precise sequence in which tasks should be performed.)

Step 1: List the outputs

This step comprises the production of a table showing (a) all the outputs of the proposed system; (b) for each output, its recipients; (c) for each recipient, the use to which the output will be put.

See figure 3-1.

Step 2: Specify the main trigger conditions

This step comprises the production of a table showing (a) all the outputs of the proposed system; (b) for each output, its main trigger condition. Main trigger conditions are system inputs. There may be alternative triggers for a given output. Date and/or time (ie. input from calendar/clock) is allowed as a trigger, and so is the operator activity of loading a program.

See figure 3-2.

Step 3: Specify the subsidiary trigger conditions

This step is carried out wherever the main trigger is not a sufficient condition for the production of an output. The further (subsidiary)

| OUTPUT SET | RECIPIENT | PURPOSE |
|---|---|---|
| 1. Despatch Instruction | Stores | To tell stores to pick stock. |
| | Despatch | To tell despatch department to deliver goods to a particular address. |
| | Customer | To tell customer what goods he is receiving. |
| | Order processing | To tell order processing clerks the current position on the orders so that they can deal with queries. |
| 2. Invoice | Customer | To tell customer what he has received and what he owes so that he may pay. |
| | Order processing | To tell order processing clerks the current position on the order so that they can deal with queries. |
| | Accounts | To keep accounts informed regarding customer's account position so that they can deal with queries. |
| 3. Statement | Customer | To tell customer the amount now due for payment on his account so that he may settle the balance. |
| | Accounts | To keep accounts informed of customer's account balance so that they can deal with queries and identify customer's payments. |

Figure 3-1

| OUTPUT SET | | MAIN CONDITION |
|---|---|---|
| Despatch Instruction | | Customer Order |
| | OR | Production Advice |
| Invoice | | Despatch Advice |
| Statement | | Statement Request |

Figure 3-2

| | | |
|---|---|---|
| For each (main condition): | Customer Order | ✓ |
| Where (further limiting conditions): | Anticipated Account Balance | ⩽ Credit Limit |
| | Free Stock | > 0 |
| Produce | Despatch Instruction | |

Figure 3-3

conditions are shown in a table which is a variation of an extended entry decision table.

See figure 3-3.

Step 4: List the contents of the outputs

This step identifies the data items making up each output. Each item is coded to indicate its use: A (to be acted upon by the output's recipient), I (to identify other items) or N (to provide useful additional information).

Figure 3-4 gives an example of the results of step 4. (It also repeats the information from steps 2 and 3 in a slightly revised form.)

Figure 3-5 gives an example of the results of steps 1 - 5 in a different notation. (It also includes some information, on item identification, which is added in step 6.)

Step 5: List all data items

This step involves partitioning the union of all output data items into (a) given items, (b) derived items. The derivation for each derived item must be specified, in terms of other items which themselves are either given or derived. This derivation analysis is continued untill all derived items have been specified in terms of given items.

Figure 3-6 gives an example of the derivation dictionary partially

**Output Set Description Sheet**

| | | | | |
|---|---|---|---|---|
| Reference | $O_{G_1}$ | | | |
| Output Set Name: *Despatch Instructions* | Date | 6.5.70 | | |
| System: *Order Processing* | Analyst | H.R. | | |

| Trigger : For Each | *Customer Order* | *Production Advice* | | |
|---|---|---|---|---|
| *Anticipated A/c Bal.* | *// Credit limit* | *// Credit Limit* | | // |
| *Outstanding Qty.* | – | > 0 | | ✓ |
| *Free Stock* | > 0 | > 0 | | ✓ |

| DATA SETS | USE | |
|---|---|---|
| DESPATCH No. | I | ✓ |
| CUSTOMER No. | N | ✓ |
| CUSTOMER NAME | A | ✓ |
| DELIVERY ADDRESS | A | ✓ |
| CUSTOMER ORDER No. | N | ✓ |
| ORDER DATE: | N | ✓ |
| PRODUCT No. | I | ✓ |
| QUANTITY TO DESPATCH | A | ✓ |
| OUR ORDER No. | N | ✓ |

Figure 3-4

1. DESPATCH INSTRUCTION

Customer Order: Anticipated a/c Bal.‖ ≤Credit Limit
∧ Free Stock‖ > "0"

∨ Production Advice: Outstanding Qty. ‖ > "0" [ 'ET' | 'Product No.' | Our Order No. ]

giving Our Order No.

∧ Anticipated a/c Bal. [ 'ET' | Customer No. ] [ 'Our Order No.' ]
‖ ≤Credit Limit [ 'ET' | Customer No. ] [ 'Our Order No.' ]
∧ Free Stock‖ > "0"

Despatch No.
Customer No.

Customer Name [ 'ET' | Customer No. ] [ 'Our Order No.' ]

Delivery Address
Customer Order No.
Order Date
Product No.
Qty. to Despatch
Our Order No.

2. INVOICE

Despatch Advice: Outstanding Qty. ‖ < "0"
Invoice No.
Customer No.

Customer Address [ ET | Customer No. ] [ 'Our Order No.' ]

Delivery Address
Customer Order No.
Order Date
'Product No.'
'Qty. Despatched'
Price
Item Total
Discount Total
'Our Order No.'
'Despatch Date'

Customer Name [ ET | Customer No. ] [ 'Our Order No.' ]

Invoice Total
Discount

Figure 3-5

DERIVATION DICTIONARY

System: Order Processing | Page: 1

| CHECK | DERIVED | DERIVATION |
|---|---|---|
| ✓ | Despatch No. | For each Despatch Instruction |
| ✓ | Qty. to Despatch | Free Stock ‖ > Outstanding Qty. / Outstanding Qty. \| > "0" ∧ < Outstanding Qty / Free Stock<br>where:<br>Free Stock ⌊ 'Product No.' / ET ⌊ 'Despatch No.'<br>Outstanding Qty. ⌊ ET / Our Order No. / 'Product No.' ⌊ 'Despatch No.' |
| ✓ | Invoice No. | For each Invoice |
| ✓ | Item Total | Qty. Despatched ⌊ 'Product No.' / Despatch No. ⌊ 'Invoice No.' × Price ⌊ 'Product No.' / ET ⌊ 'Invoice No.' |
| ✓ | Discount Total | Σ Discount |
| ✓ | Invoice Total | Σ Item Total - Discount Total |
| ✓ | Balance | Σ Payments ⌊ Invoice No. ⌊ 'Customer No.' - Σ Invoice Total ⌊ Invoice No. ⌊ 'Customer No.' |
| ✓ | Discount | Item Total × Discount Rate ⌊ 'Product No.' / Despatch No. ⌊ 'Invoice No.' |
| ✓ | Anticipated a/c Balance | Balance - Σ Pipeline Value ⌊ Despatch No. ⌊ Our Order No. ⌊ 'Customer No.' |
| ✓ | Outstanding Qty. | Qty. Ordered - Σ Qty. Despatched ⌊ 'Product No.' / Despatch No. ⌊ 'Our Order No.'<br>- Σ Pipeline Qty. ⌊ 'Product No.' / Despatch No. ⌊ 'Our Order No.' |
| ✓ | Free Stock | Stock - Σ Pipeline Qty. ⌊ 'Product No.' |
| ✓ | Pipeline Qty. | Σ Qty to Despatch ⌊ 'Product No.' / 'Despatch No.' / Despatch Instructions & Despatch Advice |
| ✓ | Stock | Σ Receipts ⌊ Batch No. ⌊ 'Product No.' - Σ Qty. Despatched ⌊ 'Product No.'. |
| ✓ | Discount Rate | Customer Type ‖ "Wholesale" / "Home" / > Quota / "40%" \| "Wholesale" / "Home" / < Quota / "25%" \| "Retail" / "Home" / - / "20%" \| "Trade" / "Home" / - / "15%" \| - / "Export" / - / "25%"<br>Area<br>Qty. Ordered<br>where':<br>Customer Type ⌊ ET / Customer No. ⌊ Our Order No. ⌊ 'Despatch No.'<br>Area ⌊ ET / Customer No. ⌊ Our Order No. ⌊ 'Despatch No.'<br>Qty. Ordered ⌊ 'Product No.' / Our Order No. ⌊ Customer No. ⌊ Our Order No. ⌊ 'Despatch No.' / ET this year<br>Quota ⌊ ET / Customer No. / 'Product No' ⌊ Our Order No. ⌊ 'Despatch No.' |
| ✓ | Pipeline Value | Pipeline Qty. × Price ⌊ 'Product No.' / ET ⌊ 'Despatch No.' × Discount Rate |

Figure 3-6

produced during this step. (It includes entries only for derived items, and includes some information, on item identification, which is added in step 6.)

Figure 3-7 gives a related, but different, example, in which all data items in the system are shown, whether given or derived. (For derived items, cross-references to decision tables are shown where appropriate. For given items, cross-references to inputs are shown; this information is added in step 7.)

Step 6: Specify primary identifiers

In this step, the primary identifiers (equivalent to primary keys in the relational model) are specified in a primary identification dictionary.

See figure 3-8.

Completion of this step allows the completion of documents initiated in steps 4 and 5.

Step 7: Design inputs

All given data items are now grouped into inputs. Some inputs will already have been identified as triggers (step 2); some new inputs will need to be identified. Given items in the dictionary in figure 3-7 can now be cross-referenced to their appropriate inputs.

Figure 3-9 shows one form in which the results of this step are documented. Each item is coded to indicate its use: I (to identify

| DATA SETS | Origin |
|---|---|
| ET | All input groups |
| Despatch No. | For each Desp. Instr /2 |
| Customer No. | 1,5 |
| Customer Name | 5 |
| Delivery Address | 1 |
| Customer Order No. | 1 |
| Order Date | 1 |
| Product No. | 1, 2, 3.5, 6 |
| Qty. to Despatch | Table 1 |
| Our Order No. | 1 |
| Invoice No. | For each Invoice /7 |
| Customer Address | 5 |
| Qty. Despatched | 2 |
| Price | 6 |
| Item Total | Qty. Desp. x Price |
| Discount Total | Σ Discount |
| Despatch Date | 2 |
| Invoice Total | Σ Item Total - Discount Total |
| Balance | Σ Payments - Σ Inv. Total |
| Discount | Item Total x Discount Rate |
| Anticipated A/c Bal | Balance - Σ Pipeline Value |
| Credit Limit | 5 |
| Outstanding Qty. | Qty. Ordered - Σ Qty Despatched - Σ Pipeline Qty. |
| Qty. Ordered | 1 |
| Receipts | 3 |
| Free stock | Stock - Σ Pipeline Qty |
| Pipeline Qty. | Σ Qty. to Despatch |
| Stock | Σ Receipts - Σ Qty Despatched |
| Discount Rate | Table 2 |
| Area | 5 |
| Customer Type | 5 |
| Quota | 5 |
| Payment | 7 |
| Pipeline Value | Pipeline Qty. x Price x Discount Price |
| Batch No. | 3 |
| | |

Decision Table 1

| Free Stock | ≥ Outstanding Quantity | > 0 ∧ < Outstanding Quantity |
|---|---|---|
| Quantity to Despatch | Outstanding Quantity | Free Stock |

Decision Table 2

| Customer Type | 'Wholesale' | 'Wholesale' | 'Retail' | 'Trade' | - |
|---|---|---|---|---|---|
| Area | 'Home' | 'Home' | 'Home' | 'Home' | 'Export' |
| Σ Qty. Ordered | ≥ Quota | < Quota | - | - | |
| Discount Rate | 40% | 25% | 20% | 15% | 25% |

Figure 3-7

PRIMARY IDENTIFICATION DICTIONARY

System: Order Processing | Page: 1

Primary Identifiers

| DATA SETS | ET | Our Order No | Customer No | Product No | Despatch No | Invoice No | Batch No |
|---|---|---|---|---|---|---|---|
| ET | | 1 | | | 2 | 3 | 4 |
| Despatch No. | 1 | 1 | | | | | |
| Customer No. | | 1 | | | | | |
| Customer Name | 1 | | 1 | | | | |
| Delivery Address | | 1 | | | | | |
| Customer Order No. | | 1 | | | | | |
| Order Date | | 1 | | | | | |
| Product No. | | | | | | 1 | |
| Qty. to Despatch | | | | 1 | 1 | | |
| Our Order No. | | | | | 1 | | |
| Invoice No. | | | | | 1 | | |
| Customer Address | 1 | | 1 | | | | |
| Qty. Despatched | | | | 1 | 1 | | |
| Price | 1 | | | 1 | | | |
| Item Total | | | | 1 | | 1 | |
| Discount Total | | | | | | 1 | |
| Despatch Date | | | | | 1 | | |
| Invoice Total | | | | | | 1 | |
| Balance | 1 | | 1 | | | | |
| Discount | | | | 1 | | 1 | |
| Anticipated Account Balance | 1 | | 1 | | | | |
| Credit Limit | 1 | | 1 | | | | |
| Outstanding Qty. | 1 | 1 | | 1 | | | |
| Qty. Ordered | | 1 | | 1 | | | |
| Receipts | | | | | | | 1 |
| Free Stock | 1 | | | 1 | | | |
| Pipeline Qty. | | | | 1 | 1 | | |
| Stock | 1 | | | 1 | | | |
| Discount Rate | | | | 1 | 1 | | |
| Area | 1 | | 1 | | | | |
| Customer Type | 1 | | 1 | | | | |
| Quota | 1 | | 1 | 1 | | | |
| Payment | | | | | | 1 | |
| Pipeline Value | | | | 1 | 1 | | |
| Batch No. | | | | | | | |
| | | | | | | | |

Figure 3-8

## Input Set Description Sheet

| | | | |
|---|---|---|---|
| Input Set Name | Customer Order | Reference | 1G1 |
| System | Order Processing | Date | 19/5/70 |
| | | Analyst | H.B. |

| DATA SETS | |
|---|---|
| CUSTOMER No. | I |
| DELIVERY ADDRESS | N |
| CUSTOMER ORDER No. | N |
| ORDER DATE | N |
| PRODUCT No. | I |
| OUR ORDER No. | I |
| QUANTITY ORDERED | N |

Figure 3-9

## INPUT SET DICTIONARY

| System | Page |
|---|---|
| Order Processing | 1 |

INPUT SETS

| DATA SETS | Order | Despatch Advice | Product'n Advice | Payments | Customer Details | Product Details | Statemt Request |
|---|---|---|---|---|---|---|---|
| ET | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Customer No. | ✓ | | | | ✓c | | |
| Delivery Address | ✓ | | | | | | |
| Customer Order No. | ✓ | | | | | | |
| Order Date | ✓ | | | | | | |
| Product No. | ✓ | ✓ | ✓ | | ✓ | ✓c | |
| Our Order No. | ✓ | | | | | | |
| Qty. Ordered | ✓ | | | | | | |
| Qty. Despatched | | ✓ | | | | | |
| Despatch Date | | ✓ | | | | | |
| Receipts | | | ✓ | | | | |
| Batch No. | | | ✓ | | | | |
| Invoice | | | | ✓ | | | |
| Payments | | | | ✓ | | | |
| Customer Name | | | | | ✓ | | |
| Customer Address | | | | | ✓ | | |
| Credit Limit | | | | | ✓ | | |
| Area | | | | | ✓ | | |
| Customer Type | | | | | ✓ | | |
| Quota | | | | | ✓ | | |
| Price | | | | | | ✓ | |

Figure 3-10

other items) or N (new information).

Figure 3-10 shows an alternative form, in which all given items for the system are cross-referenced to the inputs in which they are given.

A final comment on Systematics is that the developer is responsible for carrying out any checks for consistency and completeness. Its unique and powerful feature is that it proposes new types of checks, but they are quite difficult in practice to comprehend and carry out.

## 3.4 STRUCTURED ANALYSIS AND DESIGN

Under this heading fall a number of approaches which differ in detail but which are related by their common link with Yourdon. The variant described here is Gane and Sarson (1979). It has two features in common with Systematics. First, it concentrates on the task of producing "a logical functional specification, a detailed statement of what the system is to do, which is as free as possible of physical considerations of how it will be implemented". Second, it offers a two-level account of how this task is to be carried out; and it is not easy to reconcile the two accounts. Whereas for Systematics, in the previous section, an attempt was made to merge the two accounts, in this section only the more detailed and clear-cut account will be summarised. It consists of four steps.

Step 1: Draw logical data flow diagrams

Data flow diagrams are used to represent the flow of data between "real-world" entities, processes and data stores. They are first used to document existing systems, and then to specify possible new systems. Automated systems boundaries can be indicated on data flow diagrams. Process boxes in a data flow diagram can be "exploded" to lower-level diagrams.

See figure 3-11.

Step 2: Construct data dictionary

A data dictionary is used to hold information about all objects named

during system development. Data flow diagrams identify both data flows and data stores; they are composed of intermediate data structures (cf Cobol groups), which in turn are composed of atomic data elements. All these are named, and entered in the data dictionary with appropriate information about them. In addition, the data dictionary is used to hold entries for objects other than data objects: "real-world" entities, and processes. More general glossary entries may also be included. A data dictionary may be in either manual or automated form.

See chapter 5 for a further discussion of data dictionaries, and for example figures.

Step 3: Define process logic

Gane and Sarson (1979) offer a variety of tools for use in defining process logic: decision trees, decision tables, structured English, pseudo-code and tight English. They discuss the relationships between these tools, and the strengths and weaknesses of each.

Again, see chapter 5 for further discussion and example figures.

Step 4: Define the contents of data stores

This step provides a logical database schema, in relational third normal form, consistent with Gane and Sarson's purpose of remaining independent of physical considerations. Steps 3 and 4 between them complete the detail logical specification of the new system.

See figure 3-12.

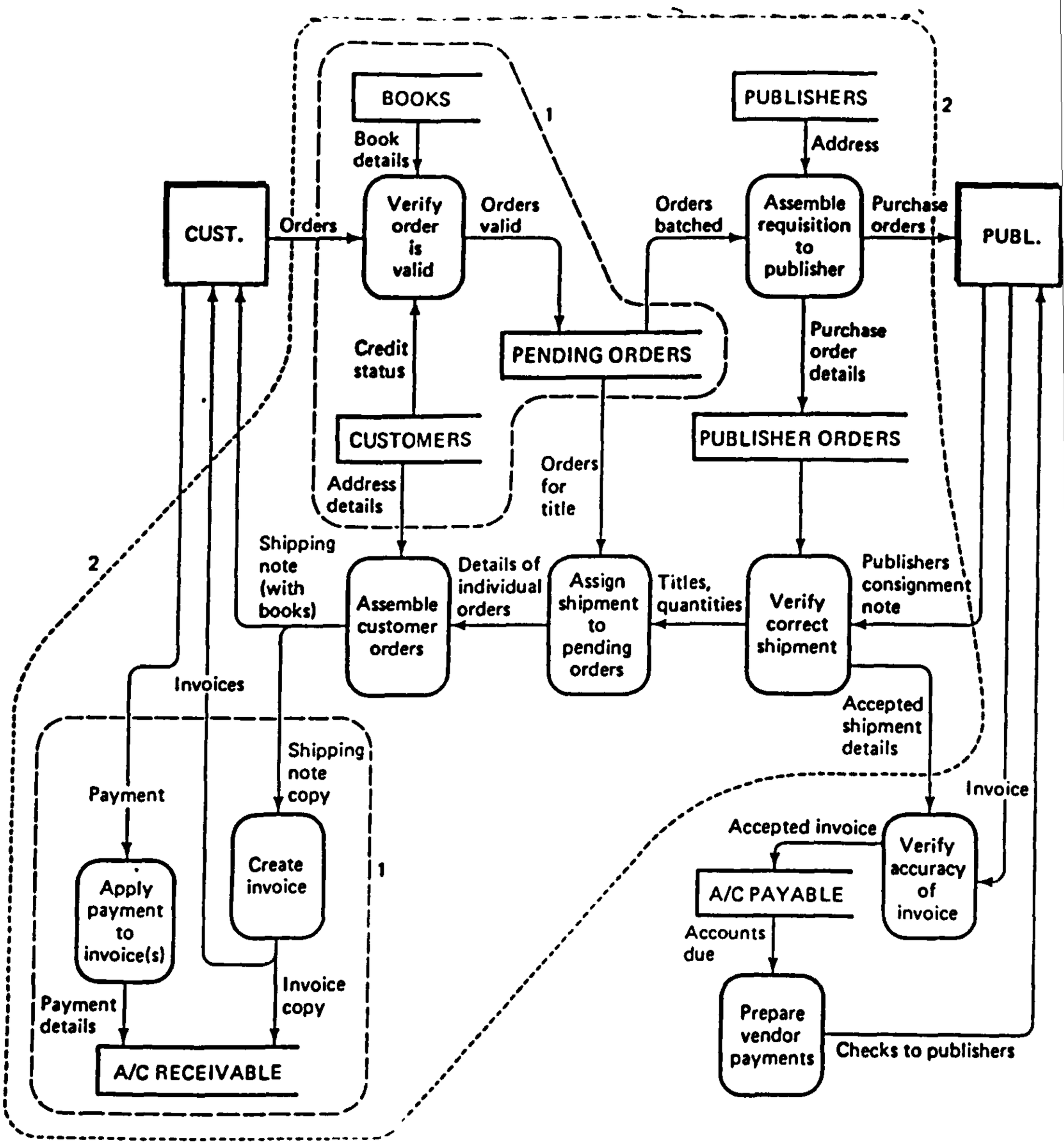



Figure 3-11

| | |
|---|---|
| ORGANIZATION-MASTER | (ORG-ID, ORGANIZATION-NAME, ORGANIZATION-ADDRESS, BILLING-ADDRESS, MAIN-PHONE, DATE-ACCOUNT-OPENED, BALANCE-OUTSTANDING, NUMBER-OF-ORDER-TO-DATE) |
| CONTACTS | (CONTACT-NAME, ORG-ID, CONTACT-PHONE, JOB-TITLE) |
| BOOK-INVENTORY | (ISBN, BOOK-TITLE, AUTHOR, PUBLISHER-NAME, PRICE, QUANTITY-ON-HAND, QUANTITY-ON-ORDER, REORDER-LEVEL) |
| AUTHOR-AFFILIATION | (AUTHOR, ORGANIZATION-AFFILIATION) |
| INVOICES | (INVOICE-NO, ORG-ID, INVOICE-DATE, INVOICE-AMOUNT) |
| PAYMENTS | (CHECK-NO, ORG-ID, PAYMENT-DATE, PAYMENT-AMOUNT) |

Figure 3-12

A final comment on structured analysis and design that it offers more down-to-earth notations than Systematics but, like Systematics it relies entirely on the developer to apply verification for consistency and completeness.

## 3.5 JSD

"A JSD project has three main phases. In the first phase, consisting of the entity action and entity structure steps, an abstract description of the real world is made. In the second phase, consisting of the initial model, function, and system timing steps, the abstract description is realised as a process model, and the currently known functions are specified on the basis of this model. The third phase consists of the implementation step, and converts the specification into a practical set of executable programs matched both to the response requirements of the specification and to the number and power of the available processors. A major checkpoint should occur at the end of each phase. At the end of the first and second phases, the check is concerned to establish the fit between the specification and the user's needs; at the end of the third phase, the check is primarily technical, addressing questions of convenience and efficiency of system execution, and the correctness of the implementation with respect to the specification. The first two phases are focused on the user, on his world, on his view of his world, and on what help he wants from the system. The third phase is technical, and concerned with the computer". (JACKSON 1982.)

The following is a brief description of the six steps identified in the quotation above.

Step 1: Entity action step

The developer identifies "real-world" entity types which are relevant

to the system to be developed (the criterion is that the system will produce or use information about them); for each entity type he identifies actions that it performs/suffers. Entities and actions must exist in the real world (not in the designed system) and must be atomic. The actions for a given entity type must be capable of being ordered in time, and must be capable of being thought of as occurring at a point (rather than over a period) of time.

The result of this step is an initial system model. The entities and actions which are listed constitute a definition of the model boundary.

See figure 3-13.

Step 2: Entity structure step

For each entity type, the actions which have been listed as occurring during its lifetime are now expressed as a sequential process, using the diagramming notation familiar in JSP. If it proves impossible to express an entity's action in this way (ie. if more than one diagram would be necessary to do so), then the entity type must be decomposed into a set of entity types such that the diagramming conventions are adequate. (An example in the book is of a soldier, who pursues two concurrent careers: a promotion career and a training career. The sets of activities for each career need to be shown separately, as attributes of "separate" entity types.)

See figure 3-14.

# The Widget Warehouse Company

## 1 ENTITY AND ACTION LISTS

CUSTOMER: PLACE, AMEND, CANCEL, DELIVER
CLERK: DELAY, ALLOCATE
ORDER: PLACE, AMEND, CANCEL, DELIVER, DELAY, ALLOCATE
PRODUCT: ALLOCATE, DELIVER

## 2 ACTION DESCRIPTIONS

PLACE: convey an order to the company for allocation and delivery. Action of CUSTOMER and ORDER.
Attributes: product-id, quantity, requested date, . . .

AMEND: change the quantity or requested date of an order; product-id cannot be changed. Action of CUSTOMER and ORDER.
Attributes: code (new quantity or new requested date), quantity or date, . . .

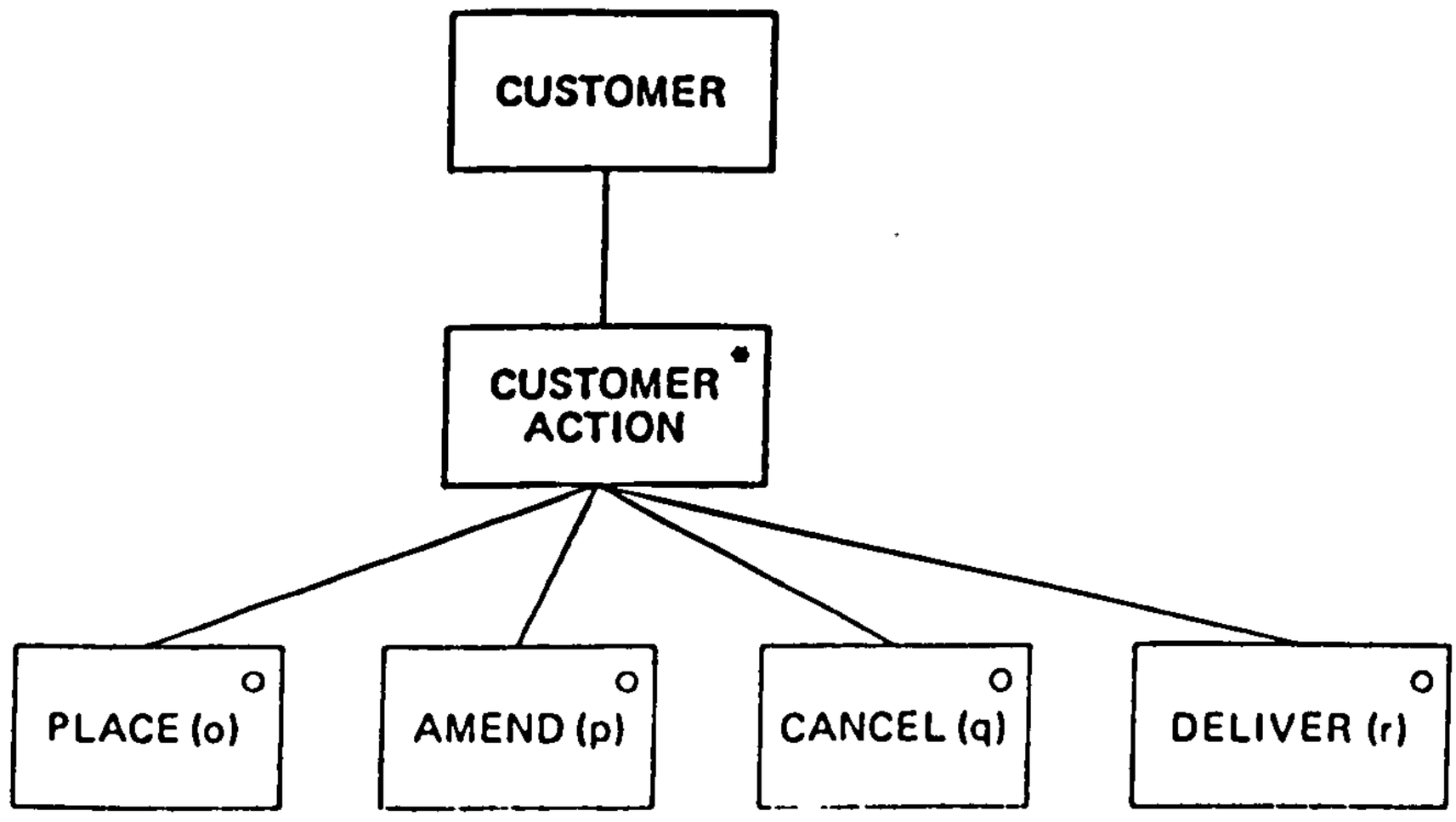
CANCEL: cancel an order. Action of CUSTOMER and ORDER.
Attributes: . . .

DELAY: delay an order because stock is not available for it to be allocated. Action of CLERK and ORDER.
Attributes: . . .

ALLOCATE: allocate product stock to an order. Action of CLERK, ORDER, and PRODUCT.
Attributes: quantity, . . .

DELIVER: deliver ordered product to a customer. Action of CUSTOMER, ORDER, and PRODUCT.
Attributes: date, quantity, . . .

Figure 3-13


CUSTOMER
CUSTOMER ACTION
PLACE (o)
AMEND (p)
CANCEL (q)
DELIVER (r)

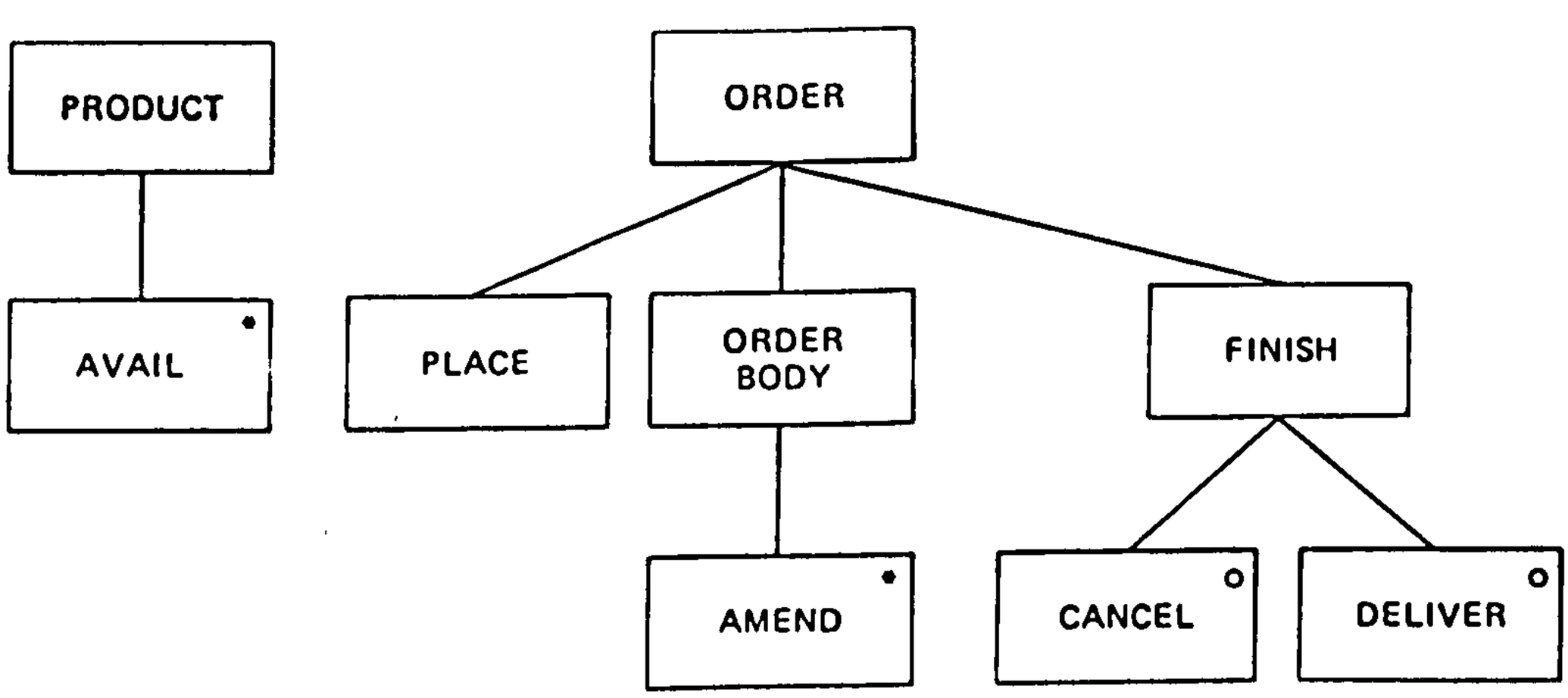

PRODUCT
AVAIL
ORDER
PLACE
ORDER BODY
AMEND
FINISH
CANCEL
DELIVER

Figure 3-14

Step 3: Initial model step

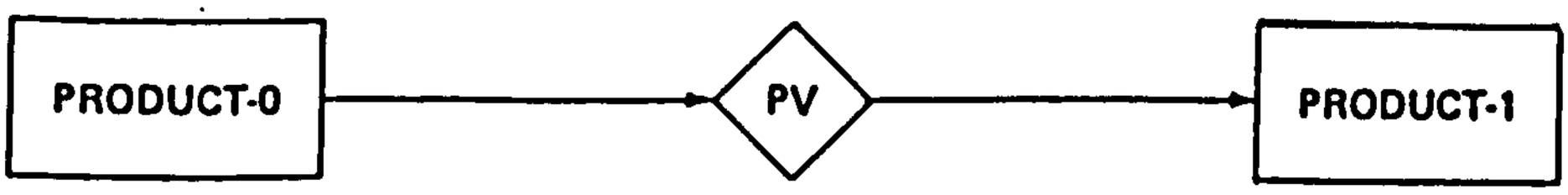
The next step is to produce an initial model of the system to be designed. This comprises a set of processes, each one matching a real-world entity process as modelled in step 2, extended by the provision of a connection between the two so that an action of the real-world process (referred to as a level 0 process) causes relevant information to pass to the system process (level 1 process). These inter-process connections are of two types: data streams, and state vector inspections. They are shown by system specification diagrams.

Each system process can now be expressed in the form a structure text. This is a textual form of the corresponding real-world process structure diagram from step 2, with the addition of operations for data stream or state vector communication.

See figure 3-15.

Step 4: Function step

The initial model is one which simply (passively) tracks events in the real world; it does not do anything of its own accord. The purpose of a designed system is, of course, that it should perform useful functions. Such functions are added to the system model in step 4. They are specified in the form: "When such - and - such a combination of events has occurred in the real world, the system should produce such - and - such outputs". The specification is documented first as an elaboration of the appropriate system specification diagram, showing how the new function is connected to the existing system




(a) System specification diagram showing state vector connection


(b) System specification diagram showing data stream connections.

```
PRODUCT-1 seq
   getsv PV;
   PRODUCT-1-BODY itr
      AVAILABILITY-AT-DATE-I seq
         AVAIL; i := j (where PV = DATEj);
         getsvPV;
         AVDATE-BODY itr while (DATEi)
            getsv PV;
         AVDATE-BODY end
         AVAILABILITY-AT-DATE-I end
      PRODUCT-1-BODY end
   PRODUCT-1 end
```

```
CUSTOMER-1 seq
   read C;
   CUSTOMER-1-BODY itr
      CUSTOMER-ACTION sel (PLACE(i))
         PLACE; write PLACE to CO(i); read C;
      CUSTOMER-ACTION alt (AMEND(j))
         AMEND; write AMEND to CO(j)); read C;
      CUSTOMER-ACTION alt (CANCEL(k))
         CANCEL; write CANCEL to CO(k); read C;
      CUSTOMER-ACTION alt (DELIVER(1))
         DELIVER; write DELIVER to CO(1); read C;
      CUSTOMER-ACTION end
   CUSTOMER-1-BODY end
CUSTOMER-1 end
```

```
ORDER-1 seq
   read CO;
   PLACE; read CO;
   ORDER-1-BODY itr while (AMEND)
      AMEND; read CO;
   ORDER-1-BODY end
   FINISH sel (CANCEL)
      CANCEL; read CO;
   FINISH alt (DELIVER)
      DELIVER; read CO;
   FINISH end
ORDER-1 end
```

(c) Structure text.

Figure 3-15

processes, and second by structure text showing the detailed specification of the function.

See figure 3-16.

Step 5: System timing step

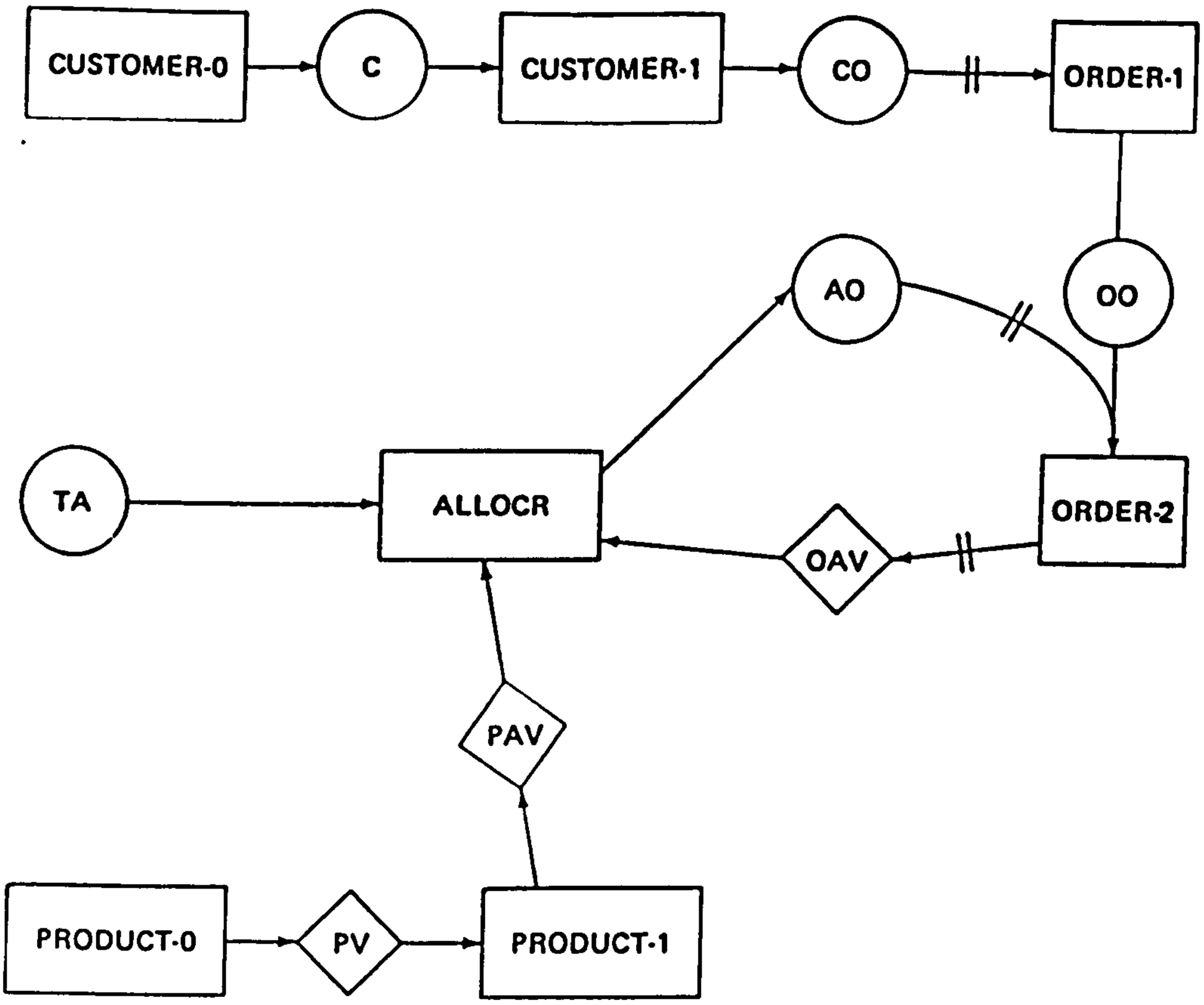
Based on his knowledge of the structure of the system model, the developer now specifies the timing constraints which must be met by the system when implemented. This specification is expressed informally. Constraints may be of various kinds, including the following.

- Response time between an input and its corresponding output.
- Frequency with which the system is updated with respect to the real world.
- Frequency with which state vectors must be inspected.

Step 6:Implementation step

This final step is concerned with producing a system implementation diagram which is a transformation of the set of system specification diagrams. The structure texts produced in earlier steps may be retained for implementation, thus substantiating Jackson's claim that the activity of programming is no longer a separate stage of system development but is dispersed throughout the development activity.

One feature of the earlier steps of JSD that has not been made explicit in this brief description is that the modals assume the




(a) Elaborated system specification diagram.

```
ALLOCR itr
  AGROUP seq
    read TA;
    getsv PAV; available := quantity in PAV;
    AGROUP-BODY seq
      getsv OAV;
      DELAY-GROUP itr while (DELAYED)
        DELAY-ORDER sel (requested ≤ available)
          available := available – requested;
          write ALLOCATE to AO (OAV);
        DELAY-ORDER alt (else)
          write DELAY to AO (OAV);
        DELAY-ORDER end
        getsv OAV;
      DELAY-GROUP end
      NORMAL-GROUP itr while (not end-of-OAVs)
        NORMAL-ORDER sel (requested ≤ available)
          available := available – requested;
          write ALLOCATE to AO (OAV);
        NORMAL-ORDER alt (else)
          write DELAY to AO (OAV);
        NORMAL-ORDER end
        getsv OAV;
      NORMAL-GROUP end
    AGROUP-BODY end
  AGROUP end
ALLOCR end
```

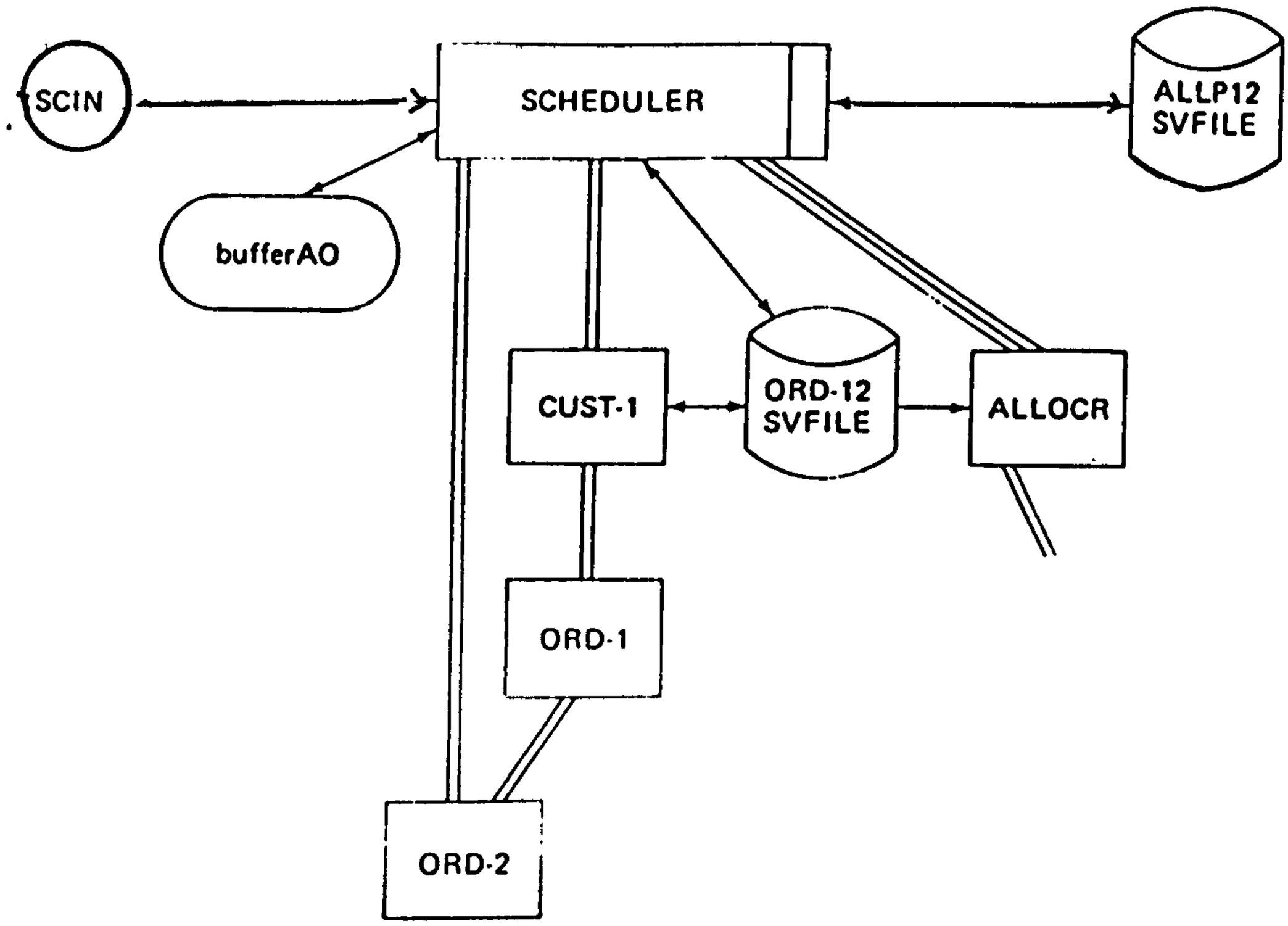
(b) Structure text for function.

Figure 3-16

existence in the system of a separate processor for each real-world entity (not entity type). Thus, for most practical systems, there would be thousands or millions of processors. The essential task of the implementation step is to remove this abstraction, by determining (a) how many real or virtual processors will be used for system running, (b) which processes will be allocated to each processor, (c) how each processor's time will be scheduled among the processes which it is to execute. Corresponding to each processor, therefore, there, will be a set of processes which are controlled by a scheduler. The detail of the scheduler is defined again by means of structure text; the dependent processes are transformed by the technique of inversion as defined in JSP.

See figure 3-17.

Assessment of JSD is made particularly difficult by Jackson's determination to distance himself from all other approaches. There are unique features in JSD (eg. system processes which exactly model real-world processes; assumption of one process per entity), and he deliberately ignores approaches which are commonly thought to be useful (eg. data dictionaries, relational analysis). But he also goes out of his way to dismiss ideas which it is not hard to see are really present in JSD in disguise (eg. stepwise refinement, conceptual modelling). JSD is similar to Systematics and Structured Analysis and Design in its coverage of the life cycle, but is very idiosyncratic in its model and expression.




(a) System implementation diagram.

```
SCHEDULER seq
  list := null; ptr := head of list;
  SCHEDULER-BODY itr
    SCHEDULER-PHASE sel (SCIN empty)
      POSSIBLE-ALLOCR sel (list is null)
      POSSIBLE-ALLOCR alt (list is not null)
        activate ALLOCR (ptr);
        query ALLOCR (ptr);
        POSSIBLE-ALLOCR-DELETE sel (read TA in ALLOCR (ptr))
          remove ALLOCR (ptr) from list;
        POSSIBLE-ALLOCR-DELETE alt (read TOA in ALLOCR (ptr))
        POSSIBLE-ALLOCR-DELETE end
        ptr := next in list;
      POSSIBLE-ALLOCR end
    SCHEDULER-PHASE alt (SCIN not empty)
      read SCIN;
      SCIN-RECORD sel (TAREC)
        query ALLOCR (TAREC-id);
        TARECORD sel (read TOA in ALLOCR (TAREC-id))
          {allocation already in progress: ignore TAREC}
        TARECORD alt (read TA in ALLOCR (TAREC-id))
          activate PROD-1 (TAREC-id);
          add ALLOCR (TAREC-id) to list;
        TARECORD end
      SCIN-RECORD alt (EREC)
        activate ENQ;
      SCIN-RECORD alt (CREC)
        activate CUST-1 (CREC-id);
      SCIN-RECORD alt (TLREC)
        activate LISTER;
      SCIN-RECORD end
    SCHEDULER-PHASE end
  SCHEDULER BODY end
SCHEDULER end
```

Figure 3-17

(b) Structure text for scheduler.

## 3.6 USE

The description of USE upon which the following account is based is given in the CRIS 1 Conference Proceedings. That description is not laid out as a set of steps (these have been inferred from the description), and indeed the impression is gained that the methodology was at that time still in a process of experiment and development.

All the examples relate to the standard CRIS 1 test case.

Step 1: Analysis

A requirements analysis is carried out, using the Structured Systems Analysis (SSA) method, to generate a set of dataflow diagrams (see figure 3-18) and a conceptual database model.

Step 2: User/system dialogue specification

All dialogues between user and system are specified using transition diagrams (see figure 3-19).

Step 3: Run interface prototype

The transition diagrams are encoded (see figure 3-20) and executed using a software tool called TDI (transition diagram interpreter). This step gives feedback to the user at an early stage of system specification.

Step 4: Database specification

The database for the system is specified as a set of normalised relations with accompanying domain definitions (see figure 3-21).

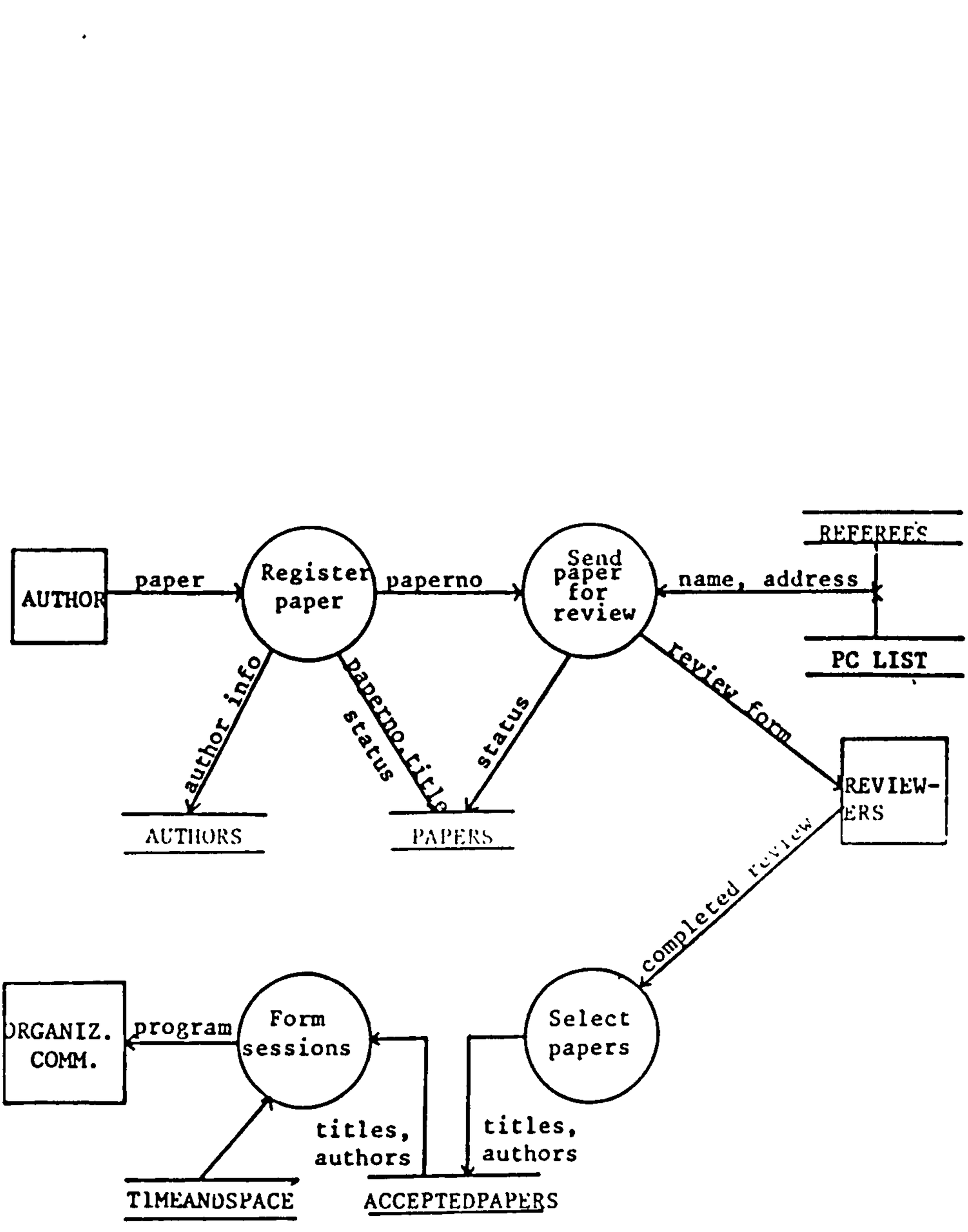
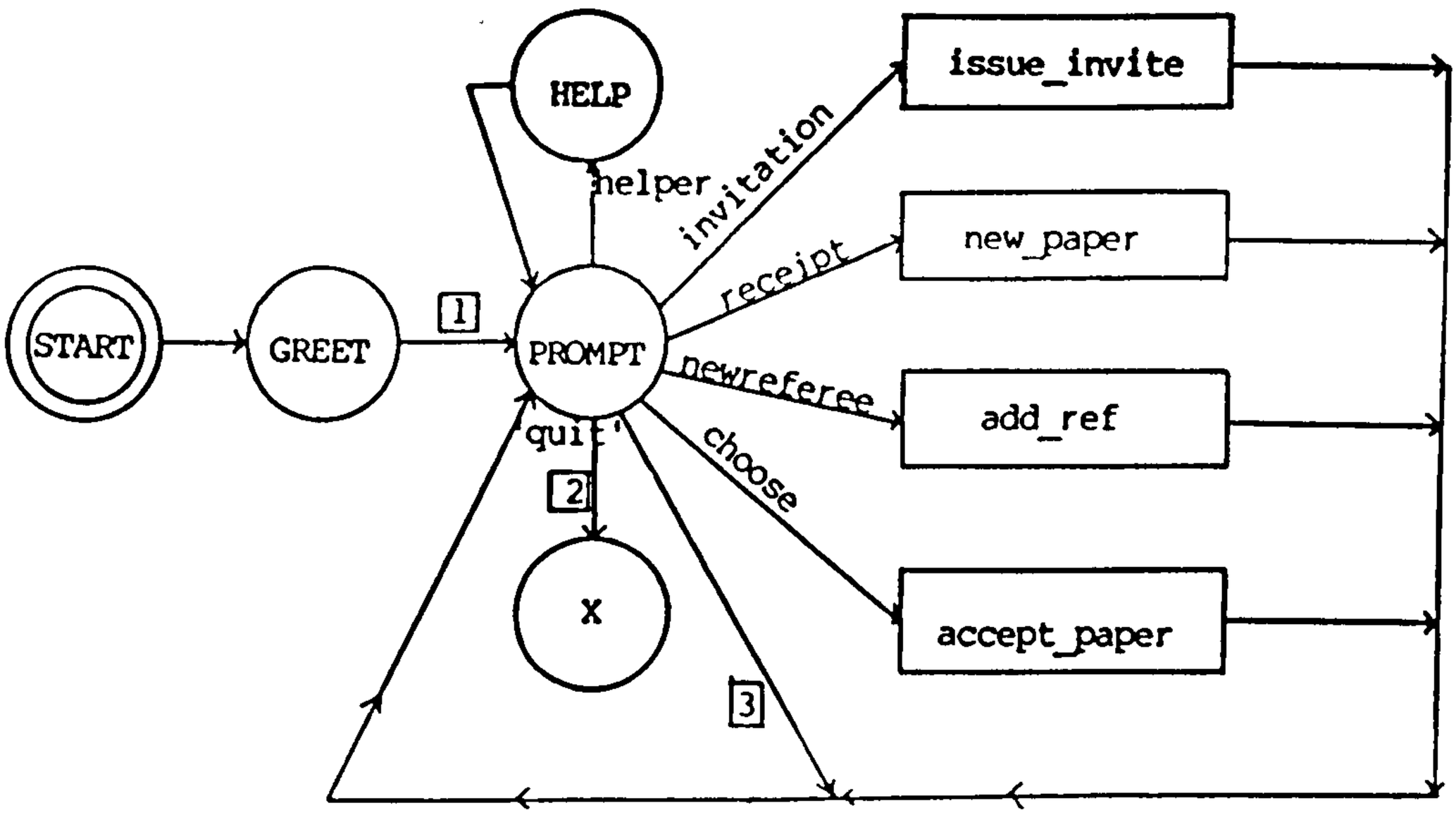

AUTHOR
paper
Register paper
paperno
Send paper for review
REFEREES
name, address
PC LIST
author info
paperno, title
status
status
review form
AUTHORS
PAPERS
REVIEW-ERS
completed review
ORGANIZ. COMM.
program
Form sessions
Select papers
titles, authors
titles, authors
TIMEANDSPACE
ACCEPTEDPAPERS

Figure 3-18



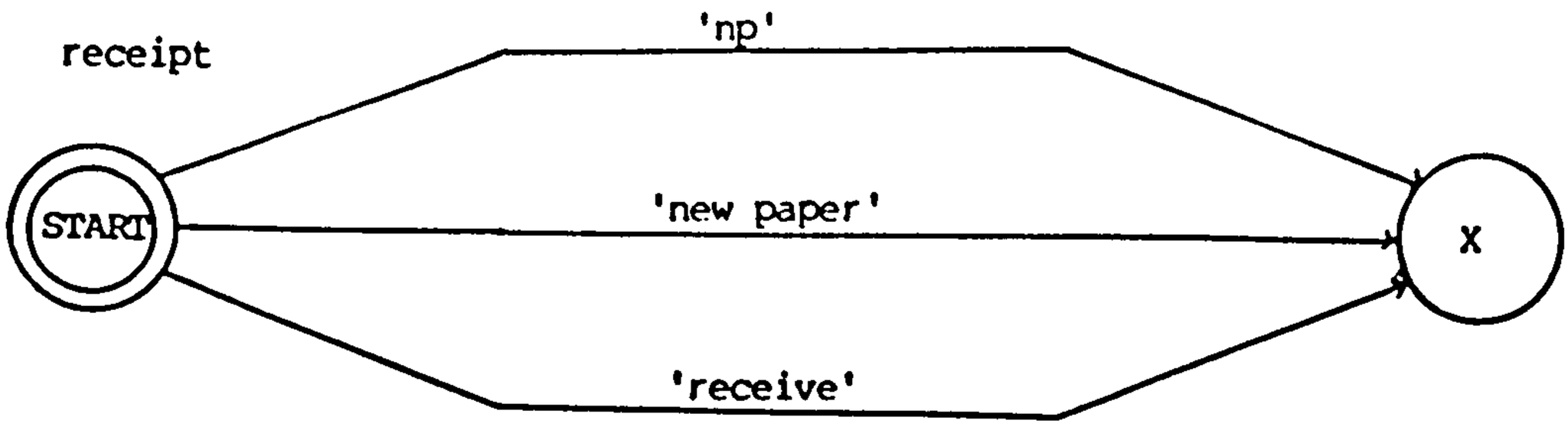

Messages

START = "Welcome to CONMAN"
PROMPT = "$"
HELP = "Valid commands are ..."
X = "Byebye

Actions

1 Open database; import relations needed by PC
2 Close database
3 write 'illegal command'

Transition diagram for "transaction level" of CONMAN for Programme Committee




Transition diagram showing valid inputs to cause transition receipt

Figure 3-19

| Diagram | Start Node |
|---|---|

Figure_5 START

| Node | Message | |
|---|---|---|
| GREET | "Welcome to CONMAN" | |
| PROMPT | "$" | |
| HELP | "Valid commands are ..." | {entire text not shown} |
| X | "Byebye" | |

| Source node | input selector | destination | action | returns |
|---|---|---|---|---|
| START | | GREET | | |
| GREET | | PROMPT | 1 | |
| PROMPT | invitation | <issue_invite> | | |
| PROMPT | receipt | <new_paper> | | |
| PROMPT | newreferee | <add_ref> | | |
| PROMPT | choose | <accept_paper> | | |
| PROMPT | helper | HELP | | |
| PROMPT | 'quit' | X | 2 | |
| PROMPT | | PROMPT | 3 | |
| HELP | | PROMPT | | |

| Action Number | Action |
|---|---|
| 1 | Open database; import relations needed by Programme Committ |
| 2 | Close database |
| 3 | write 'illegal command' |

| Diagram | Start Node |
|---|---|
| receipt | START |

| Node | Message |
|---|---|

| Source node | input selector | destination | action | returns |
|---|---|---|---|---|
| START | 'np' | X | | |
| START | 'new paper' | X | | |
| START | 'receive' | X | | |

| Action Number | Action |
|---|---|

Figure 3-20

```
domain weekday: scalar (Mon, Tue, Wed, Thu, Fri);

domain clock: integer (800..2000);

domain date: integer (0..3112);

domain money: float (0.00..200.00);

domain paperstatus: scalar (received, inreview, accepted, insession, rejected);

domain person: string;


relation accepted_papers [key paperno] of
     paperno: paperrange;
     title: string;
     sessionnum: sessionrange;
end;

relation attendance [key name] of
     name: person;
     amtpaid: money;
end;

relation author_list [key name, paperno] of
     name: person;
     paperno: paperrange;
end;


relation mailing_list [key name] of
     name: person;
     affiliation: string;
     detail_address: string;
     postcode: string;
     city: string;
     country: string;
end;

relation papers [key paperno] of
     paperno: paperrange;
     title: string;
     resp_pc_member: person;
     status: paperstatus;
end;

relation pc_list [key name] of
     name: person;
     papercount: integer (0..10);  {no PC member handles more than 10 papers}
end;

relation priority_list [key name] of
     name: person;
     role: string;
end;

relation referee_list [key name] of
     name: person;
     number_assigned: integer (0..6);  {limit on papers to be refereed}
end;


relation reviewing [key refname, paperno] of
     refname: person;
     paperno: paperrange;
     datesent: date;
     dateofreply: date;
end;

relation sessions [key sessionnumber] of
     sessionnumber: sessionrange;
     title: string;
```

Figure 3-21

Step 5: Operations specification

The functions to be performed by the system are specified in two ways. The first is informally, using narrative text. The second is formally, using a notation with axioms and verification conditions (see figure 3-22).

Step 6: Run functional prototype

The database and operations specifications are then coded using a database management system called Troll (see figure 3-23), and the system can now be run in prototype form using stored data and actual functions.

Step 7: Architectural design

The system is now decomposed to modules (apparently equivalent to programs), each of which is defined in terms of its interfaces and functions. The module structure is shown in a structure chart (see figure 3-24).

Step 8: Detailed design

The logic for each module is specified using a program design language (PDL) (see figure 3-25). Also apparently at this stage detailed databse design is carried out.

Step 9: Programming

Based on detailed specifications from step 8, programs are written in the Plain language. This is a Pascal-based language, with facilities

```
object paper
   abstract image
            <title, authorlist, papernumber, ...>
   abstract invariant
   abstract operations
            receive paper...
                pre
                post
            review paper
            accept paper
            assign paper to session
            reject paper
            accepted?
            change paper title
            change authorship
```

If we consider the operation "assign paper to session", we can identify some pre-conditions on the operation, including:

-- the session name is valid
-- the paper has not already been assigned to another session
-- the session does not have more than some maximum number of papers

Postconditions might specify:

-- the session is noted as containing that paper
-- the paper is noted as having been assigned to a session

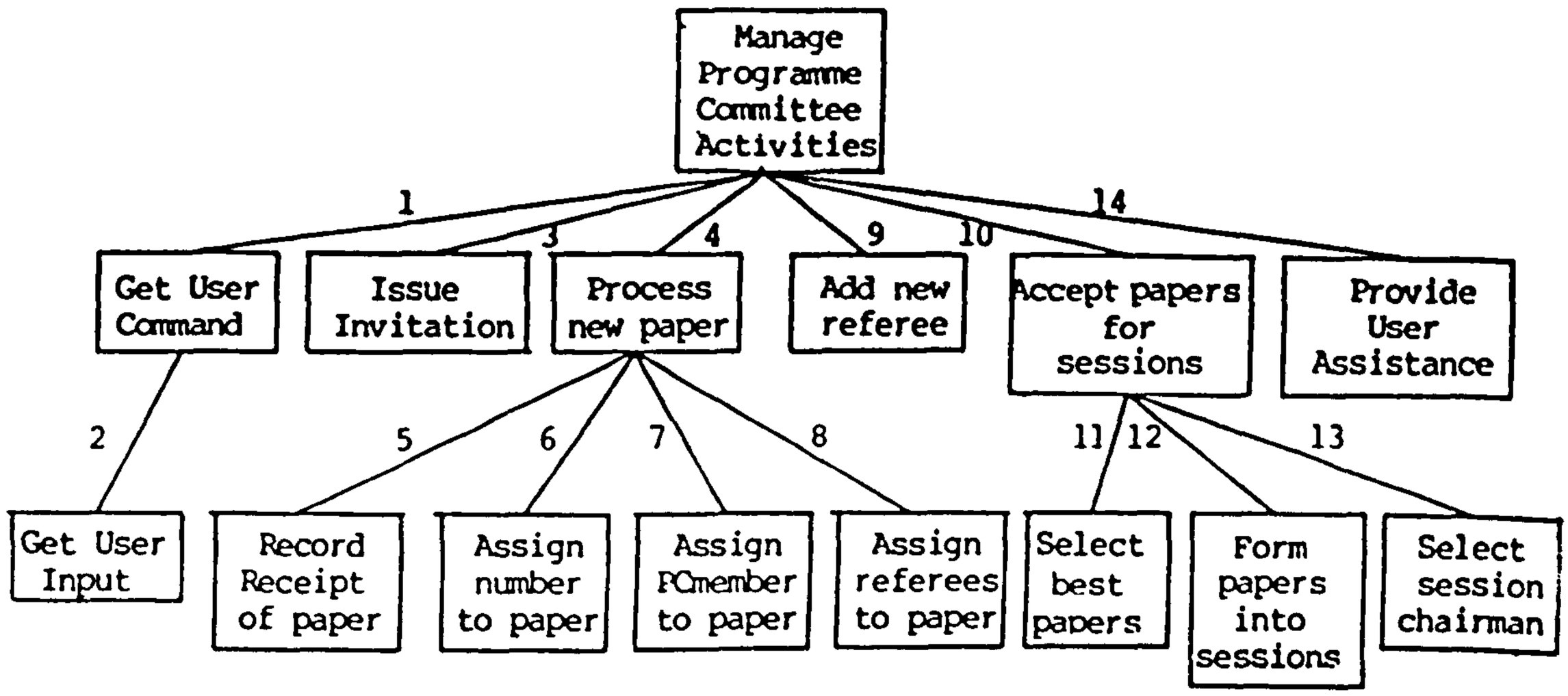
More formally, the above conditions might be specified as:

```
abstract operations
assign_paper_to_session (paper, session)
  pre   valid_session_name (session) & ~assigned (paper)
                    & paper_count (session) < MAXPAPERS
  post  assigned (paper) & contains (session, paper)
```

Figure 3-22

```
open conference;
import referee_list;
import mailing_list;
insert referee_list [<$refname,0>];
   {initially referee has no papers to review}
   {must also obtain information for mailing_list relation}
if ~exists (mailing_list [$refname]) then
     insert mailing_list [<$refname,$refaffil,$refadress,$refpostcode,
                                  $refcity,$refcountry>];
end if;
export referee_list, mailing_list;
quit;
```

Figure 3-23




| | INPUT | OUTPUT |
|---|---|---|
| 1 | | command_id |
| 2 | | input |
| 3 | mailing_list | mailing_list |
| 4 | papers, referee_list, reviewing, pc_list, author_list, mailing_list | papers, referee_list, reviewing, pc_list, author_list, mailing_list |
| 5 | papers, author_list, mailing_list | papers, author_list, mailing_list |
| 6 | papers, author_list | papers, author_list, paper_number |
| 7 | papers, pc_list | papers, pc_list |
| 8 | papers, referee_list, reviewing, paper_number | papers, referee_list, reviewing |
| 9 | referee_list, mailing_list | referee_list, mailing_list |
| 10 | papers, accepted_papers author_list, mailing_list, times, sessions, session_chair | papers, accepted_papers, sessions, session_chair |
| 11 | papers, accepted_papers | papers, accepted_papers |
| 12 | accepted_papers, papers, sessions, times | accepted_papers, papers, sessions |
| 13 | session_chair, sessions | session_chair |
| 14 | message_number | |

Figure 3-24

```
MODULE Assign_Referees
INPUT
     paperno: paperrange;
     papers, referee_list, reviewing: relation;
OUTPUT
     papers, referee_list, reviewing: relation;
        {all three relations modified by this module}
CALLS
CALLED BY
     new_paper
LOCAL DATA
     input: string;  {user input of name(s)}
     countrefs: integer (0..5);   {number of referees assigned}
FUNCTION
     For the given paper number, Assign_Referees prompts the user to assign
     one or more referees for the paper, accepting names until the user types
     an empty line (<cr> only) or until 5 names have been collected.
     The module increments the count of papers assigned to the referee,
     limiting the number of papers to six, and changes the status of the
     paper after the referees have been assigned.
ALGORITHM
     write  'Select referee(s) for paper number ',  paperno;
     write  papers[paperno].title;
     countrefs := 0;
     write 'Name:';
     read input;
     while  input ~= ""  and countrefs < 5
     loop
          if exists (referee_list [input])
          then
               if referee_list.number_assigned < 6
               then
                    referee_list.number_assigned := referee_list.number_assigned + 1;
                    insert reviewing [<input, paperno,100*day+month>];
                    papers.status := inreview;
                    countrefs := countrefs + 1
               else
                    write 'Referee has too many papers.  Try again.';
               end if
     else
               write 'Name not in referee list. Try again.';
               {***Design problem: note that minor misspellings of
                referee names or use of last name only may fail to
                find name in referee_list relation***}
     end if
end loop
     write countrefs, 'referees assigned';
     if countrefs ~= 0
     then write 'Paper ', paperno, ' in review.'
     else signal noneassigned
     end if;
EXCEPTIONS
     noneassigned
END MODULE
```

Figure 3-25

for string handling, pattern matching, exception handling and database management.

See figure 3-26.

Use offers a prototype project support environment. It is conceptually sound, and uses sensible software tools which are interconnected via UNIX. Primitive configuration management capabilities are offered via a tool called MCS (module control system).

```
procedure Assign_Referees (paperno: paperrange);
exception noneassigned;
imports papers, referee_list, reviewing: modified;
var
   input: string;
   countrefs: 0..5;
begin
        .
        .
    {body of Assign_Referees, following detailed design}
        .
        .
end Assign_Referees;
```

Figure 3-26

## 3.7 NIAM

The following are the steps to be followed in applying the NIAM methodology, as synthesised from the available account. Again the examples relate to the CRIS 1 test case.

Step 1: Object system activities

First, all activities to be performed jointly by the "object system" and the information system are shown. (The object system is that part of the total human activity system which supplies information to, and receives information from, the mechanised information system.) The activities are drawn from a prior unformalised stage of requirements description.

See figure 3-27.

Step 2: Information requirements

For each activity identified in step 1, a list of information sets is given. Each information set is an input message stream needed to perform or control the activity.

See figure 3-28.

Step 3: Information system functions

The scope of the information system is now defined by identifying the set of high-level functions which it will perform (see figure 3-29). The relationships between these functions are then shown in the form of an information flow diagram (IFD), which is essentially the same as

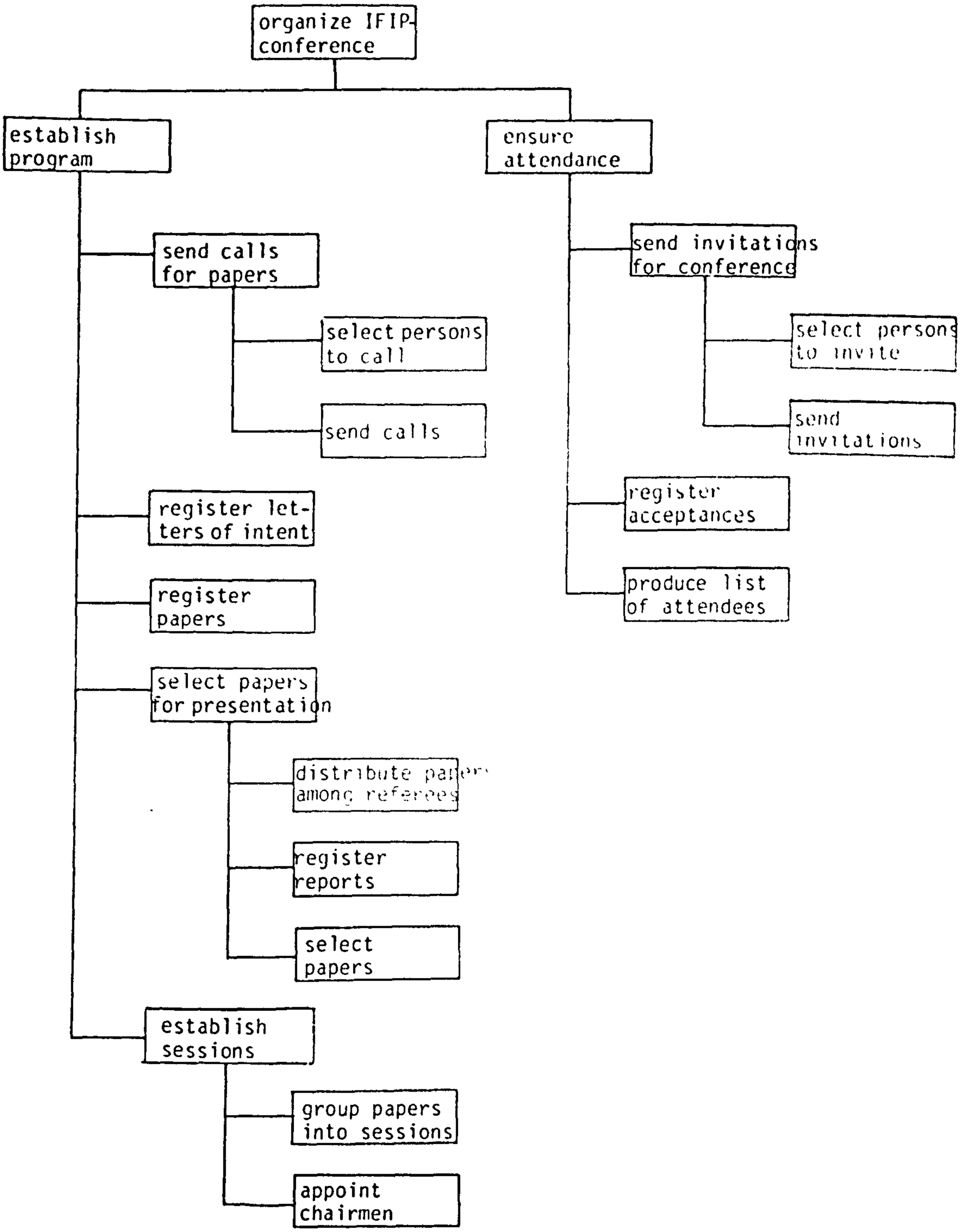

organize IFIP-conference
establish program
send calls for papers
select persons to call
send calls
register letters of intent
register papers
select papers for presentation
distribute papers among referees
register reports
select papers
establish sessions
group papers into sessions
appoint chairmen
ensure attendance
send invitations for conference
select persons to invite
send invitations
register acceptances
produce list of attendees

Figure 3-27

| Activity | Information needed |
|---|---|
| Send calls | - Information on callees<br>- Information on conference<br>- Information on call-layout |
| Distribute papers among referees | - Information on papers<br>- Information on referees<br>- Information on conference |
| Select papers | - Information on reports<br>- Information on papers<br>- Information on conference<br>- Acceptance criteria |

Figure 3-28

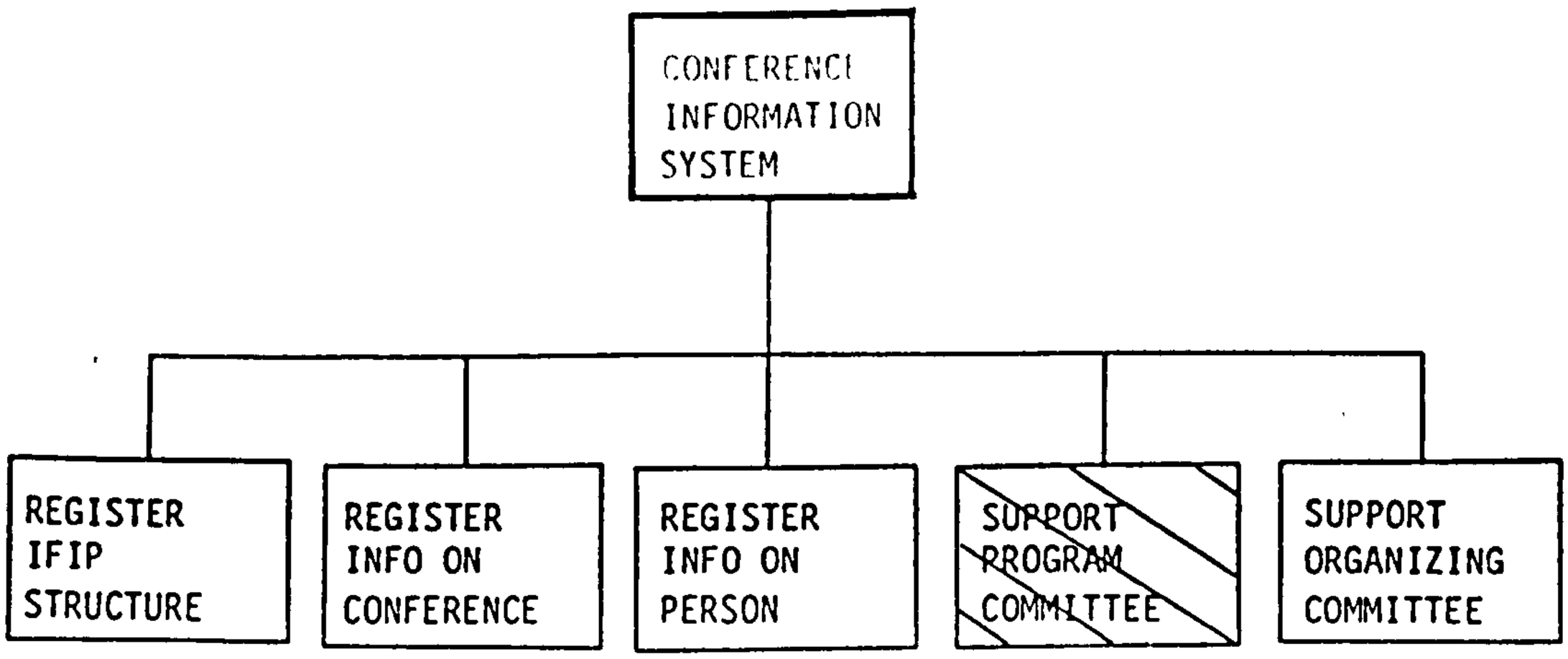



Figure 3-29

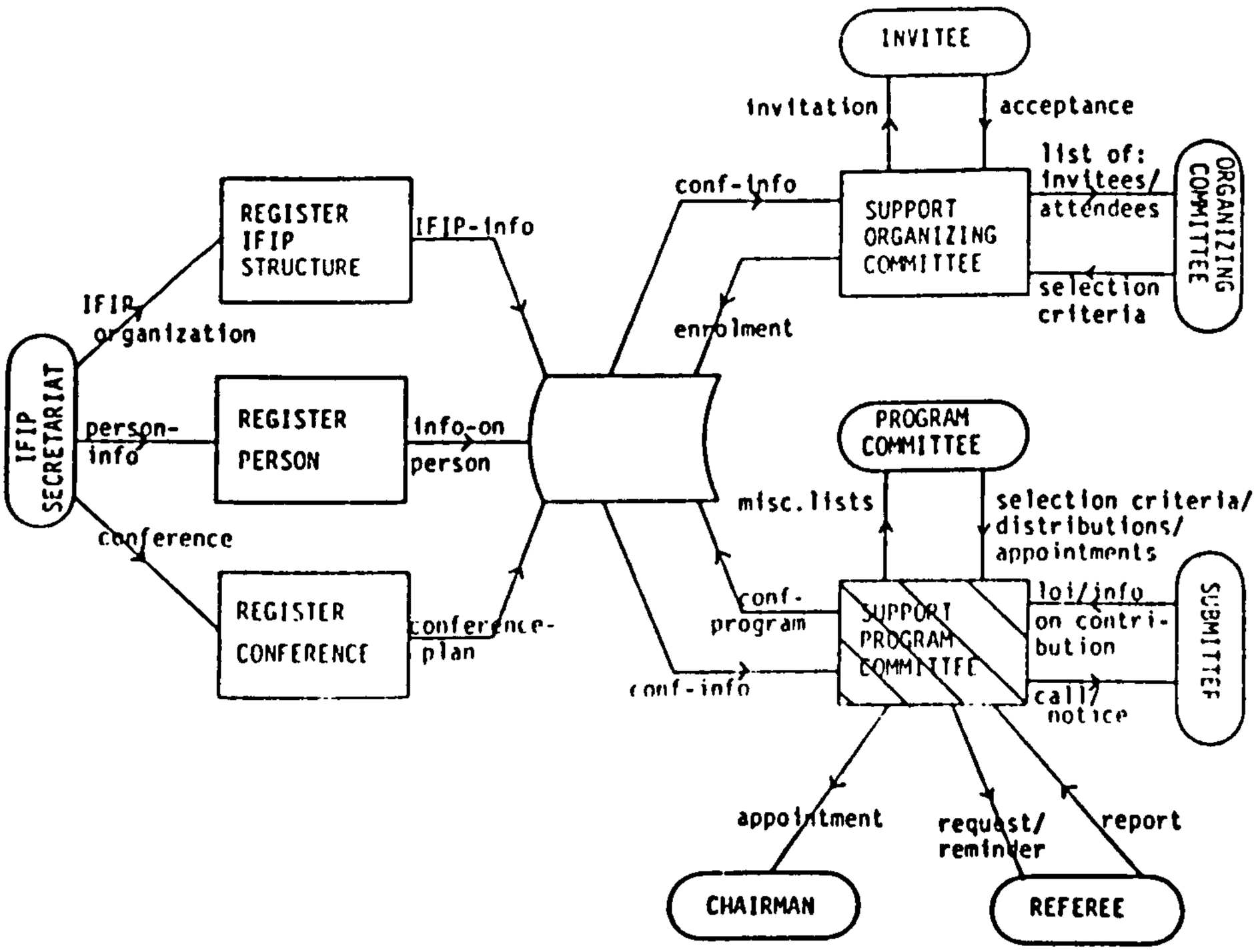



Figure 3-30

a data flow diagram in the structured methods (see figure 3-30).

Step 4: Functional decomposition

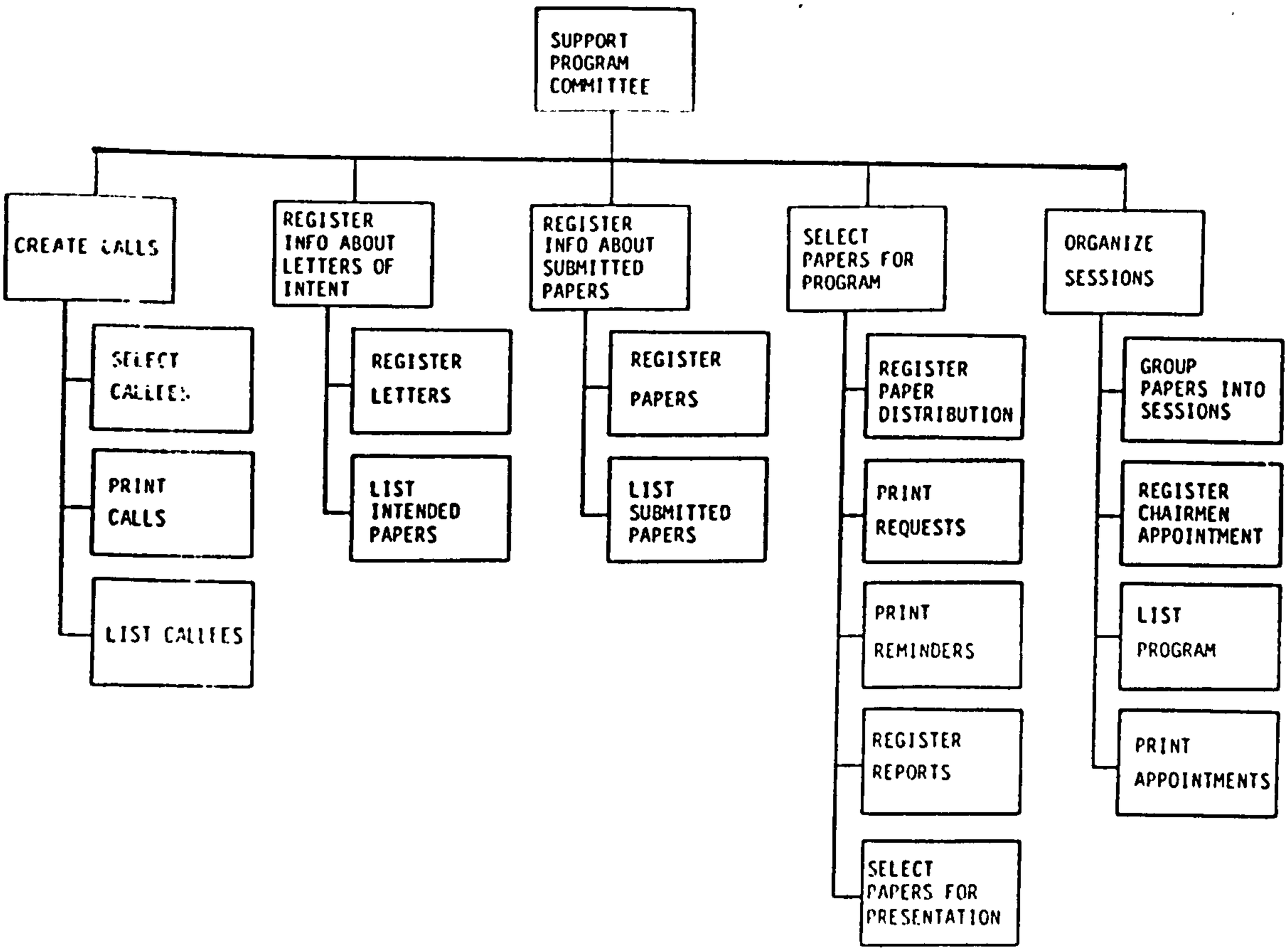
Each function is now decomposed into subfunctions. The subfunctions for each function are again related in an IFD as before (see figure 3-31). The process of decomposition continues until each individual information flow is capable of being expressed as an information structure diagram (ISD: see step 5).

Step 5: Analysis of information flows

Each individual information flow in the set of lowest-level IFDs is now analysed in terms of its component information items. This is the distinctive step in NIAM. The result of the analysis for any individual information flow is a "conceptual grammar" for that information flow, shown in a complex diagrammatic form (see figure 3-32). These diagrams permit the identification of LOTs (lexical object types) and NOLOTs (non-lexical object types), ideas (relationships between NOLOTs), bridges (relationships between a LOT and a NOLOT), relationships between types and subtypes, identification relationships, relationships between sets and subsets, and constraints of uniqueness, equality, disjunction, etc.

Step 6: Integrate ISDs

The set of ISDs is now taken and integrated for the whole system. This is done at two levels: an overview level comprising a single ISD for the system; and a series of lower-level ISDs, each centred on a


SUPPORT PROGRAM COMMITTEE
CREATE CALLS
SELECT CALLEES
PRINT CALLS
LIST CALLEES
REGISTER INFO ABOUT LETTERS OF INTENT
REGISTER LETTERS
LIST INTENDED PAPERS
REGISTER INFO ABOUT SUBMITTED PAPERS
REGISTER PAPERS
LIST SUBMITTED PAPERS
SELECT PAPERS FOR PROGRAM
REGISTER PAPER DISTRIBUTION
PRINT REQUESTS
PRINT REMINDERS
REGISTER REPORTS
SELECT PAPERS FOR PRESENTATION
ORGANIZE SESSIONS
GROUP PAPERS INTO SESSIONS
REGISTER CHAIRMEN APPOINTMENT
LIST PROGRAM
PRINT APPOINTMENTS

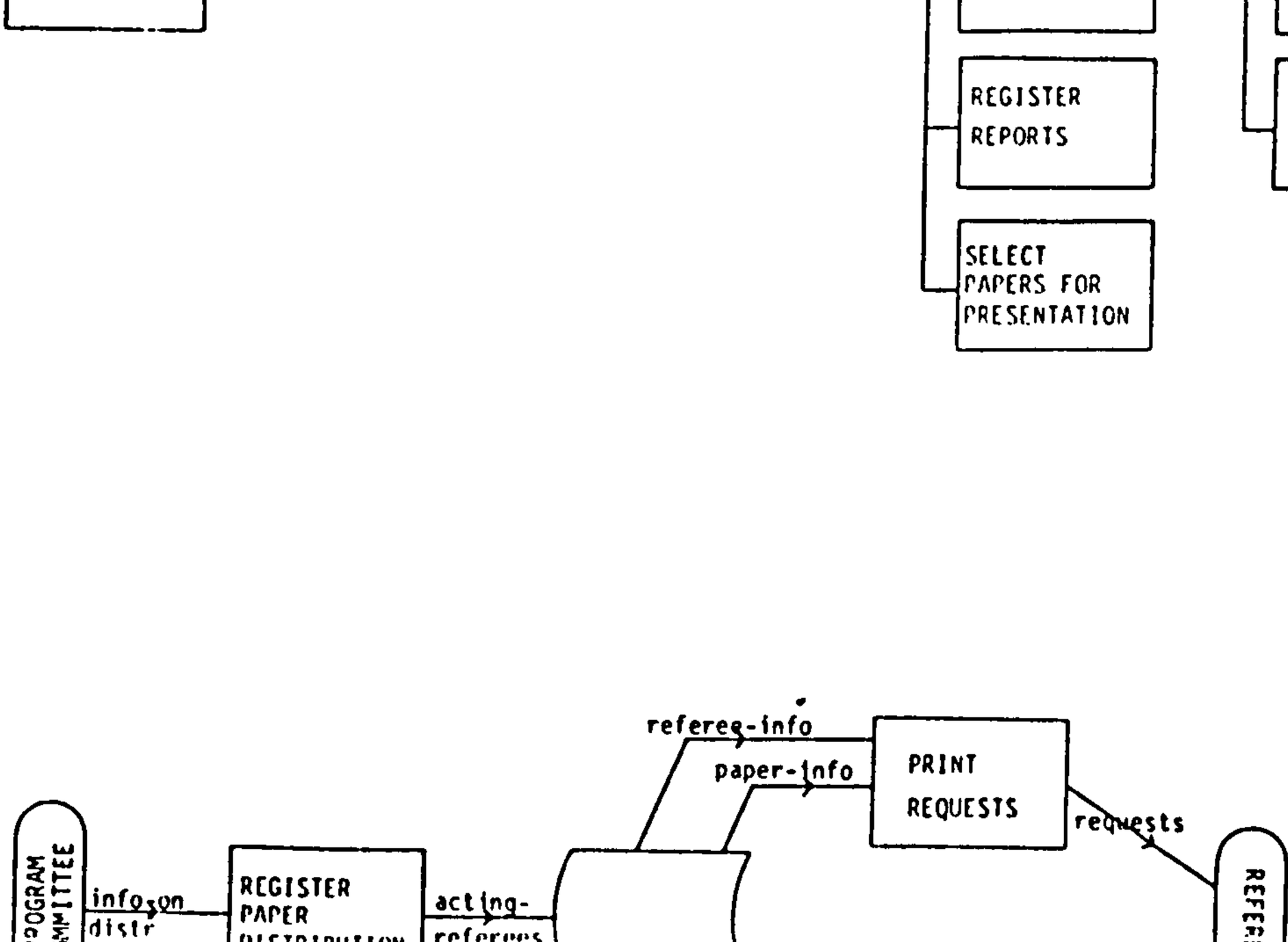

PROGRAM COMMITTEE
info.on distr
REGISTER PAPER DISTRIBUTION
acting-referees
referee-info
paper-info
PRINT REQUESTS
requests
REFEREE
referee-info
PRINT REMINDERS
reminders
paper-info
SUBMITTER
notice
reports
SELECT PAPERS FOR PRESENTATION
reports
qualified-papers
report-info
REGISTER REPORTS
PROGRAM COMMITTEE
list-of accepted-papers

Figure 3-31

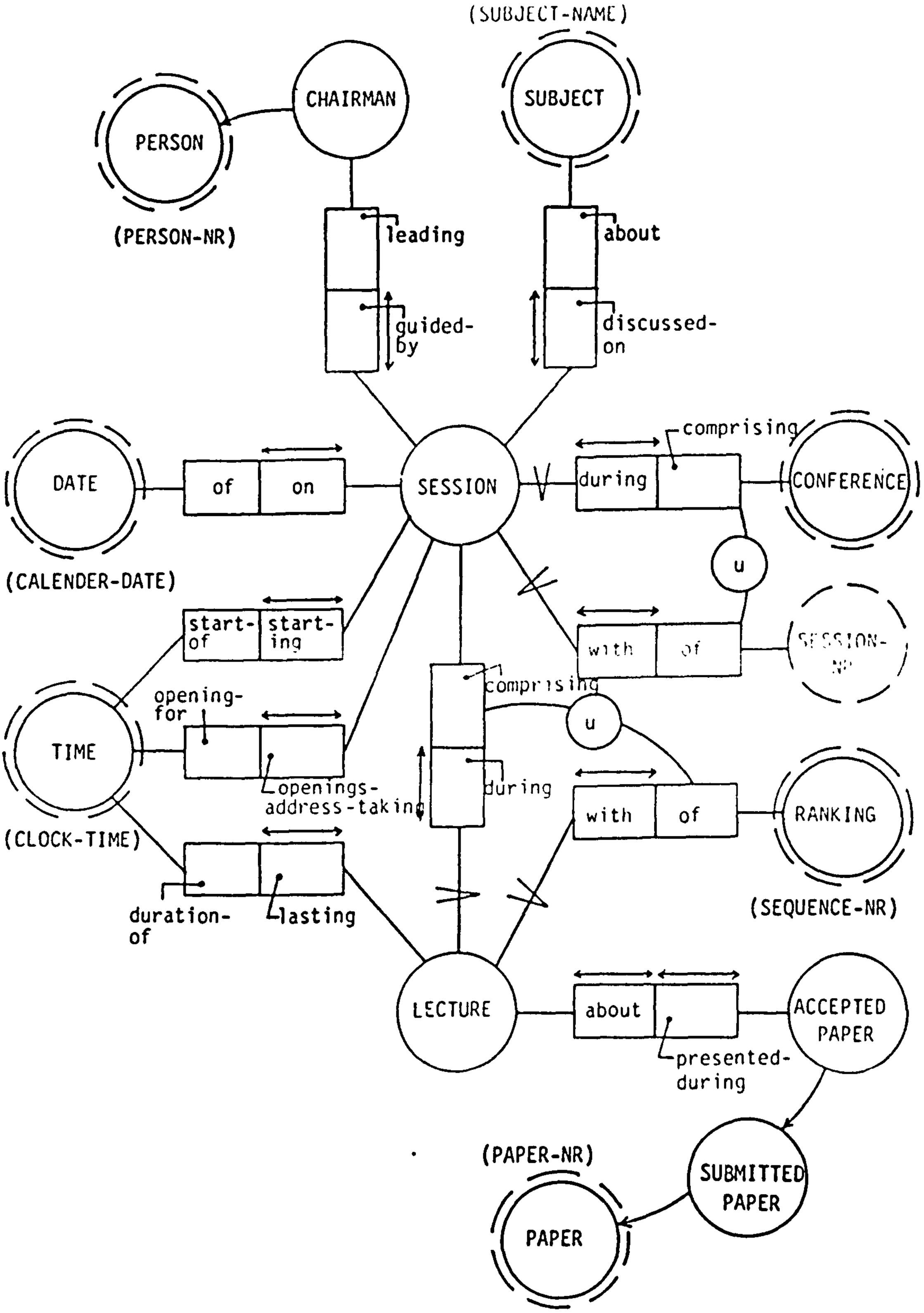

(SUBJECT-NAME)
CHAIRMAN
SUBJECT
PERSON
(PERSON-NR)
leading
about
guided-by
discussed-on
DATE
of
on
SESSION
during
comprising
CONFERENCE
(CALENDER-DATE)
u
start-of
start-ing
with
of
SESSION-NR
comprising
opening-for
u
TIME
openings-address-taking
during
with
of
RANKING
(CLOCK-TIME)
(SEQUENCE-NR)
duration-of
lasting
LECTURE
about
ACCEPTED PAPER
presented-during
(PAPER-NR)
SUBMITTED PAPER
PAPER

Figure 3-32

major NOLOT.

See figures 3-33 and 3-34.

Step 7: Re-express conceptual grammar in RIDL

RIDL is a language of fairly conventional form into which the integrated ISDs from step 6 can be rewritten. Software tools within the ISDIS toolset are available to verify RIDL statements for completeness and consistency.

See figure 3-35.

Step 8: Check RIDL specification against original requirements

This step is carried out informally by the developers.

Step 9: Compile information dictionary

This step is only mentioned in passing. It is probably, in fact, carried out in parallel with earlier steps (say steps 5 to 7).

NIAM is distinctive (a) because of its strong and sound conceptual framework, (b) because of the complexity and difficulty of its conceptual grammar diagrams. It has rightly attracted a good deal of favourable attention. It appears that the ISDIS toolset is available for use.

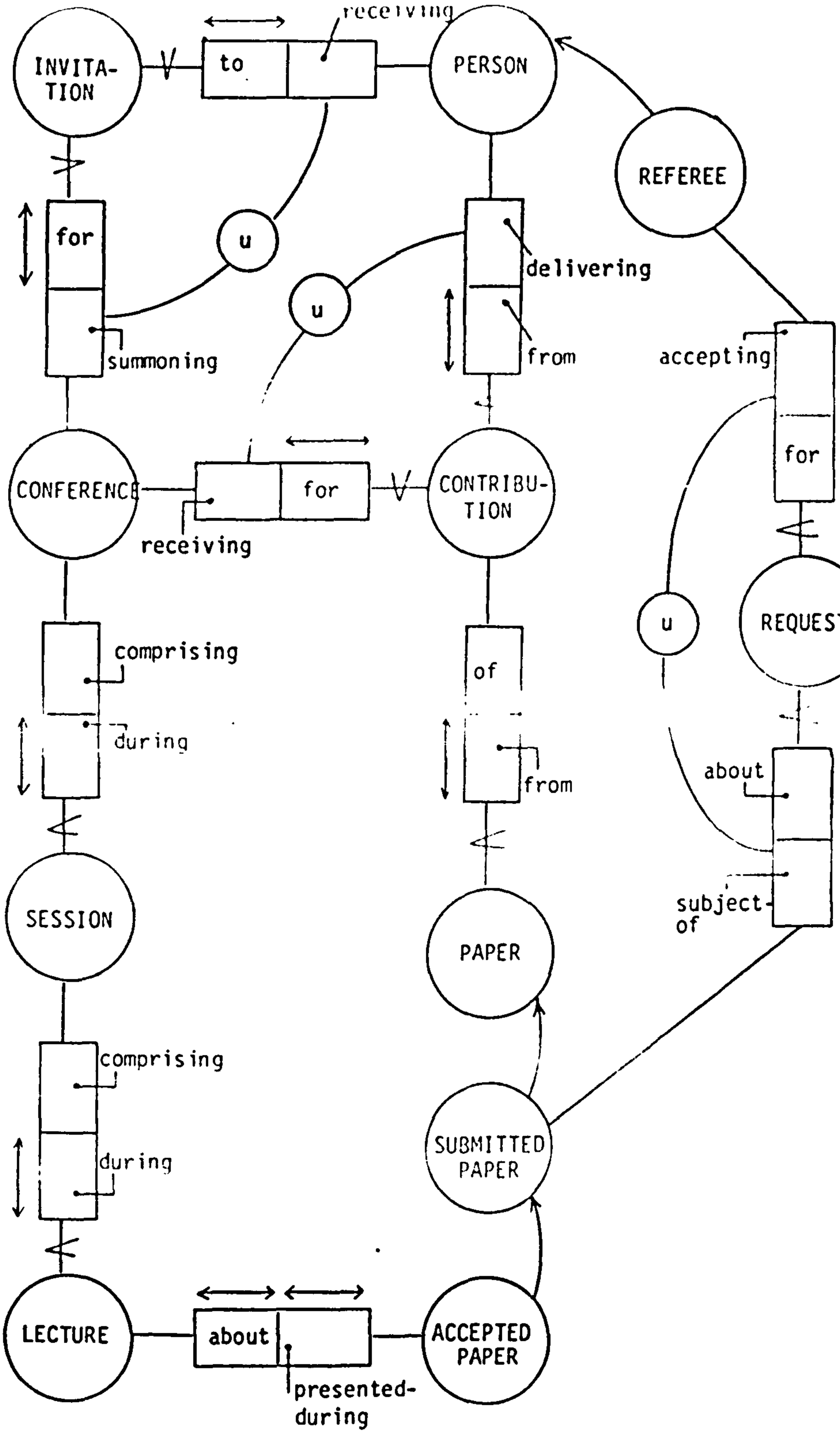

INVITA-
TION
to
receiving
PERSON
REFEREE
for
u
delivering
u
from
accepting
for
summoning
CONFERENCE
for
CONTRIBU-
TION
receiving
comprising
of
during
from
for
about
SESSION
PAPER
subject
of
comprising
during
SUBMITTED
PAPER
LECTURE
about
ACCEPTED
PAPER
presented-
during

Figure 3-33

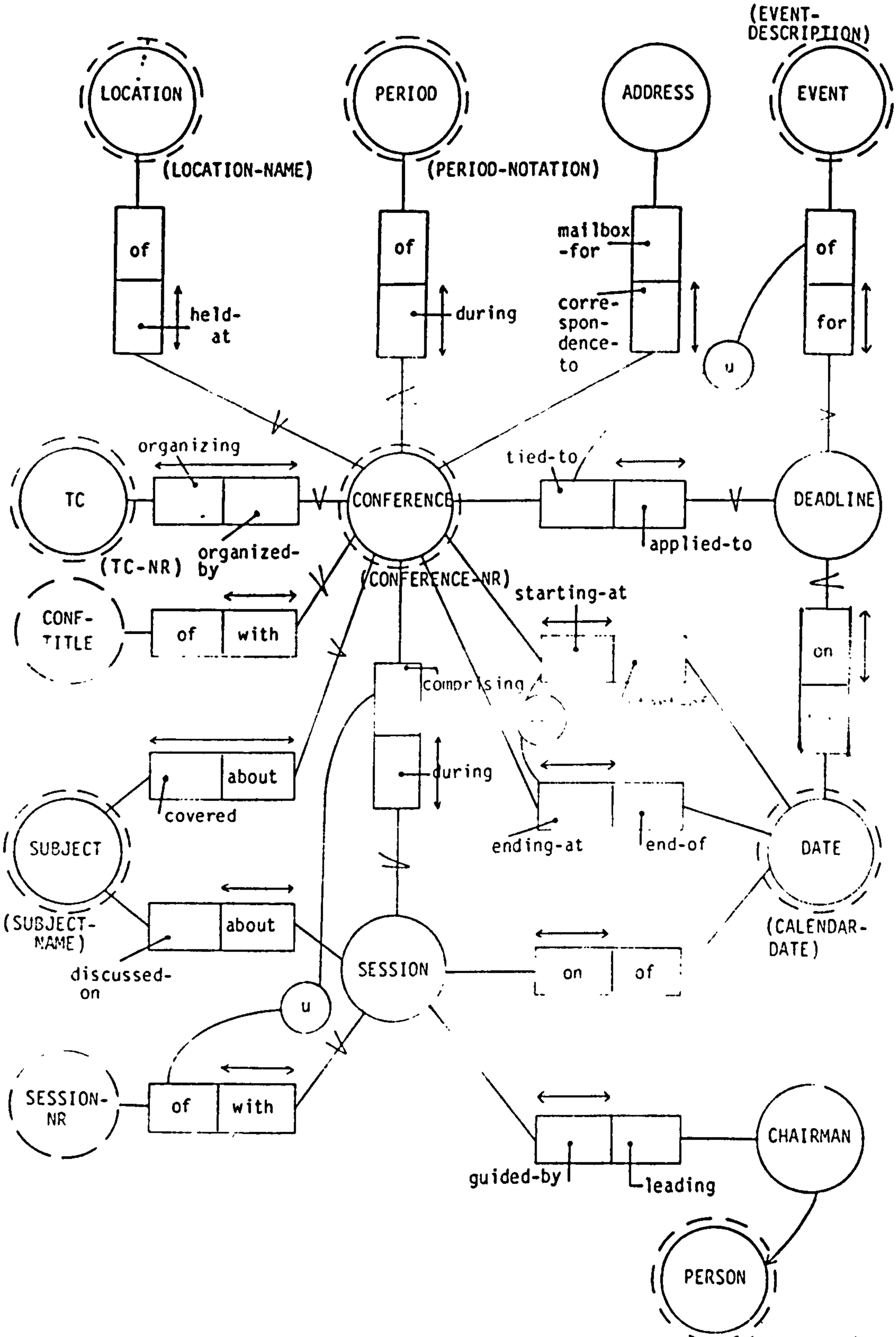

LOCATION
(LOCATION-NAME)
of
held-at
PERIOD
(PERIOD-NOTATION)
of
during
ADDRESS
mailbox-for
corre-spon-dence-to
u
(EVENT-DESCRIPTION)
EVENT
of
for
TC
(TC-NR)
organizing
organized-by
CONFERENCE
tied-to
applied-to
DEADLINE
CONF-TITLE
of
with
(CONFERENCE-NR)
starting-at
comprising
on
about
covered
during
ending-at
end-of
SUBJECT
(SUBJECT-NAME)
about
discussed-on
DATE
(CALENDAR-DATE)
SESSION
on
of
u
SESSION-NR
of
with
guided-by
leading
CHAIRMAN
PERSON
(PERSON-NR)

Figure 3-34

```
begin grammar
   add conceptual grammar IFIP-CONFERENCE;
   :
   add nolot PERSON, PAPER, CONFERENCE, CONTRIBUTION,
             INVITATION                                          ... ;
   add nolot REFEREE subtype of PERSON;
   Note: other nolot and subtype declarations omitted here.
   :
   add lot   PERSON-NR; PAPER-NR, SURNAME, TITLE,                ... ;
   Note: other lot declarations omitted here.
   :
   add idea type CONFERENCE-SOMEWHERE
      roles (CONFERENCE held-at and LOCATION of);
   Note: other idea type declarations omitted here.
   :
   add bridge type PERSON-IDENTIFICATION
      roles (PERSON bearing and SURNAME of);
   Note: other bridge type declarations omitted here.
   :
   add constraint PERSON-SURNAME
      condition
         PERSON bearing only one SURNAME
      holds;
   Note: other identifier-constraints omitted here.
   :
      add constraint CONFERENCE-IN-PERIOD
         condition
            CONFERENCE always during PERIOD
         holds;
      Note: other total-role constraints omitted here.
      :
      add constraint SESSION-IDENTIFICATION
         condition
            SESSION is identified by
               SESSION-NR of SESSION
               and
               CONFERENCE comprising SESSION
            holds;
         Note: other uniqueness constraints omitted here.
         :
         add constraint BOTH-START-AND-END-DATE
            condition
               CONFERENCE starting-at DATE
                  is equal to
               CONFERENCE ending-at DATE
            holds;
         Note: other equality constraints omitted here
         :
               add constraint REFEREE-EXCLUDES-AUTHORSHIP
```

Figure 3-35

## 3.8 ISAC

ISAC is by far the most extensive and comprehensive of the methodologies studied in this chapter. A complete account is not possible. The following includes the most significant steps and representations.

There are five main stages: change analysis (steps 1 to 3), activity studies (steps 4 to 6), information analysis (steps 7 to 9), data system design (steps 10 to 12) and equipment adaptation (steps 13 to 15).

Step 1: Analysis of problems and needs in the current situation

This step generates problem tables (see figure 3-36), lists of interest groups (see figure 3-37), descriptions of the activities of the affected interest groups using A-graphs with associated text pages (see figures 3-38, 3-39, 3-40), property tables showing measurable properties of activities and sets identified in the A-graphs (see figure 3-41), tables of objectives (see figure 3-42) and tables of needs for change (see figure 3-43).

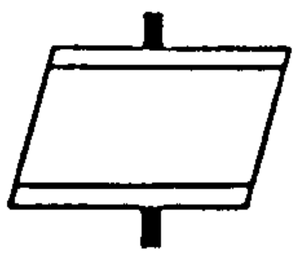
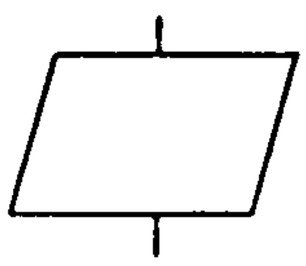
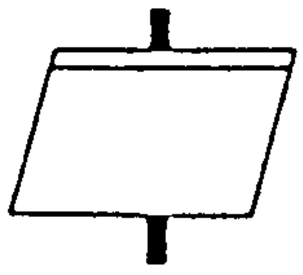
A-graphs are extremely important through many of the steps in ISAC. As indicated in figure 3-40 they are able to show real sets (people, material), message sets, composite sets, real flows, message flows, composite flows, and activities. A-graphs can be decomposed through several levels of detail. Another representation which recurs many times, and in many different detailed forms, is the property table.

| | | |
|---|---|---|
| P1 | Bad order procedures | The customers think that it takes too long a time to order, that it is easy to make mistakes and to forget articles. |
| P2 | Difficult invoices | The customers think that the invoices are difficult to work with, e.g., to compare with the delivery papers. |
| P3 | Late distribution lists | The distribution function obtains the distribution lists too late, which means that the personnel driving the distribution trucks are pressed for time. |
| P4 | Laborious order summaries | The personnel at the order offices of the dairies find it laborious to manually summarize different customer orders into distribution lists and dairy summaries. |
| P5 | Laborious economy routines | People in the economy function are not satisfied with the present laborious routines for invoices, payments, and ledgers. |
| P6 | Different order processing | There are many ways in which the order processing is performed in DAIRCO. This makes cooperation between the different dairies difficult. |
| P7 | Outdated data entry equipment | The equipment for data entry is outdated, expensive to work with, and difficult to maintain. |
| P8 | Deficiencies in the material processing | The forms for material processing are undeveloped and expensive. New packet units and distribution packings are, e.g., needed. |
| P9 | Poor basis for transportation planning | Planning tools for administering internal transportations between dairies are lacking. |
| P10 | Late and poor basis for production planning | The dairy's summaries of the customer orders are inaccurate and are obtained too late in order to plan the production. A lot of "intuition" and "rules of thumb" are used instead. |

Figure 3-36

| *Interest groups* | | *Problem (See figure 3.3.1)* | *Activites in A-graphs (current situation)* |
|---|---|---|---|
| *End users at dairies:* | | | |
| I1 | Order personnel | P1, P3, P4, P7 | C41 |
| I2 | Ledger personnel | P5 | C43 |
| I3 | Invoice personnel | P5 | C43 |
| I4 | Punching personnel | P4, P5, P7 | C41, C43 |
| I5 | Production planners | P9, P10 | C42 |
| I6 | Transport leaders | P3, P9 | C44 |
| I7 | Load personnel | P3 | C44 |
| I8 | Drivers | P3, P6 | C44, C5 |
| I9 | Accountants | P2, P5 | |
| *End users at central office:* | | | |
| I10 | Raw products controllers | P10 | C3 |
| I11 | Internal transport planners | P9 | C3 |
| I12 | Order analysts | P3, P4, P6, P8 | C3 |
| I13 | Auditors | P2, P5 | C3 |
| *The public (environment):* | | | |
| I14 | Customers | P1, P2 | C5 |
| I15 | Owners (i.e., deliverers of milk) | P1–P10 | C2 |
| I16 | Other dairy corporations | P1–P10 | C1 |
| *Funders (with result responsibility):* | | | |
| I17 | Dairy managers | P1- P10 | C4 |
| I18 | Market department at central office | P1, P2 | C3 |
| *Specialists:* | | | |
| I19 | "Prognosis analysts" (forecasters) | P1, P3, P10 | C3 |
| I20 | Systems analysts/systems designers | P1–P7, P9–P10 | C3 |
| I21 | EDP–operations personnel | P1–P7, P9–P10 | C3, C41 |

Figure 3-37

| SYMBOLS IN A-GRAPHS | CORRESPONDENCE IN DESCRIBED ACTIVITY |
|---|---|
|  | *Real Set*<br>Set of persons and/or material. |
|  | *Message Set*<br>Set of messages, e.g., documents or information by telephone. |
|  | *Composite Set*<br>Set comprising persons/material as well as messages |
|  | *Real Flow*<br>Flow of persons/material only. |
|  | *Message Flow*<br>Flow of messages. |
|  | *Composite Flow*<br>Flow of persons/material as well as messages |
|  | *Activity*<br>People and other resources take part in the activity. |

All flows are assumed to go from top to bottom on the graphs, arrows are needed on upward and (possibly) horizontal flows only.

Explanation of symbols used in A-graphs.

Figure 3-38

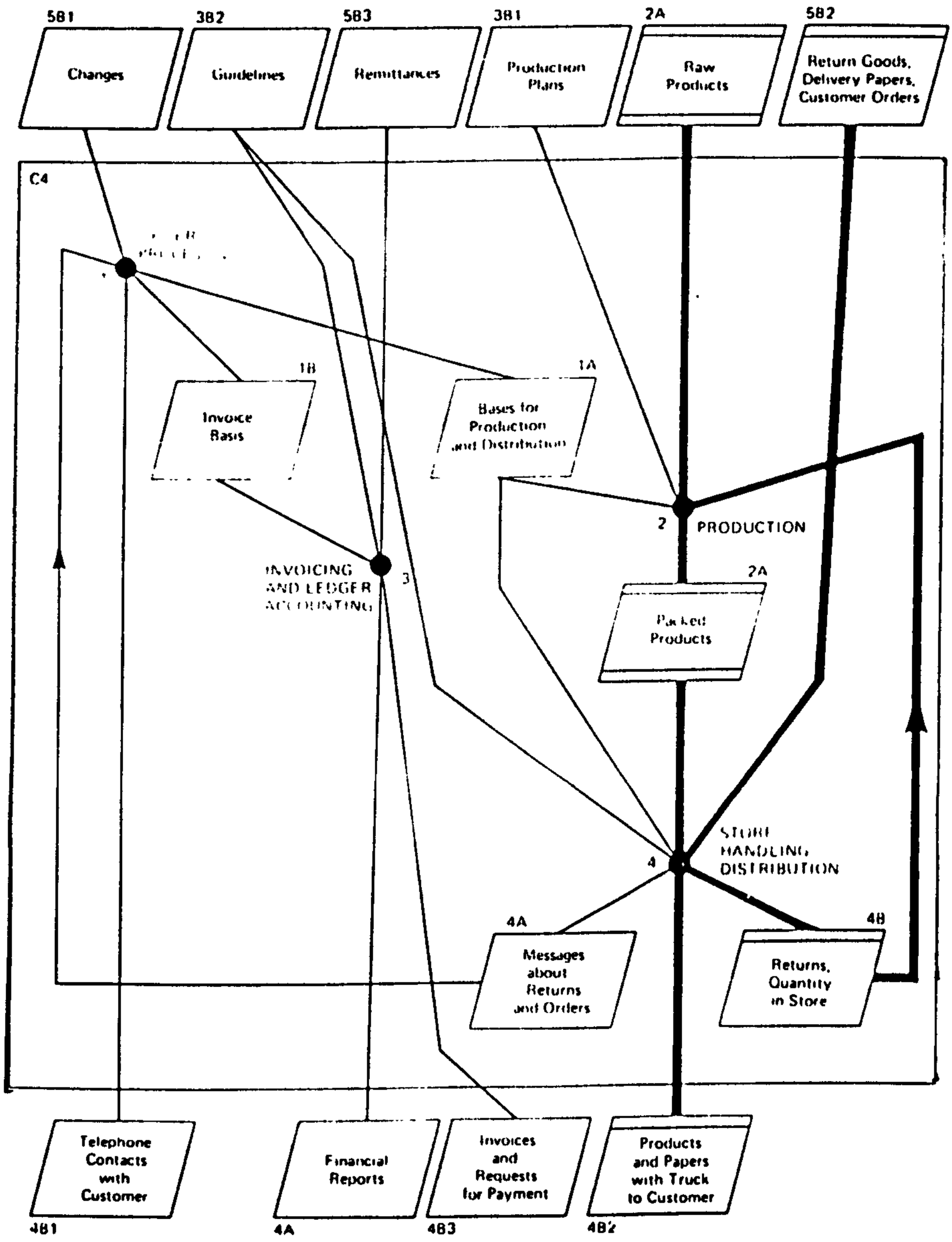

581
Changes
382
Guidelines
583
Remittances
381
Production Plans
2A
Raw Products
582
Return Goods, Delivery Papers, Customer Orders
C4
1B
Invoice Basis
1A
Bases for Production and Distribution
2
PRODUCTION
2A
Packed Products
3
INVOICING AND LEDGER ACCOUNTING
4
STORE HANDLING DISTRIBUTION
4A
Messages about Returns and Orders
4B
Returns, Quantity in Store
Telephone Contacts with Customer
Financial Reports
Invoices and Requests for Payment
Products and Papers with Truck to Customer
381
4A
483
482

Figure 3-39

TEXT PAGE
A-GRAPH

Analyst:
MOB

Date:
1981–04–15

No. C4

Subject:
Current order system:
Dairy

| | |
|---|---|
| 2A | Raw products |
| 2A1 | Cow milk |
| 2A2 | Added ingredients |
| 2A3 | Product packings |
| | |
| 3B1 | Production plans |
| 3B11 | Long-range production plans |
| 3B12 | Short-range control information for the production |
| | |
| 3B2 | Guidelines |
| 3B21 | Customer advertisements per driver (to 44) |
| 3B22 | Guidelines for invoicing e.g., payment conditions and campaign reports (to 43) |
| | |
| 5B1 | Change contacts per telephone from customer |
| 5B11 | Direct changes of driver's order |
| 5B12 | Order of extra delivery |
| | |
| 5B2 | Return goods. Delivery papers in return. Customers' orders. |
| 5B21 | Return goods on truck with driver |
| 5B22 | Signed delivery papers (from customer) with note about returns |
| 5B23 | Order papers from customers (driver's order) |
| | |
| 5B3 | Remittances |

---

| | |
|---|---|
| 4 | Dairy (= a typical dairy) |
| | |
| –1 | Order processing |
| –1A | Real bases for production and distribution |
| 1B | Bases for invoicing |
| | |
| 2 | Production |
| 2A | Packed products in external packings |
| | |
| –3 | Invoicing. Ledger accounting |
| | |
| -4 | Store handling Distribution |
| 4A | Messages about returns and customer orders |
| 4A1 | Signed delivery papers from customer |
| –4A2 | Return papers about approved return goods |
| –4A3 | Order papers from customer (driver's order) |
| | |
| -4B | Return goods and quantity in store |
| –4B1 | Return goods |
| -4B2 | Quantity in cold store |

---

| | |
|---|---|
| 4A | Financial reports about invoicing and ledger accounting |
| 4B1 | Telephone contacts to customer when orders are missing or abnormal |
| 4B2 | Products and delivery papers with truck to customer |
| 4B3 | Invoices and possible requests for payments |

Figure 3-40

| Sets | | Property: average contact volume between dairy and customer/day | | |
|---|---|---|---|---|
| *Reference code* | *Name* | *Number of customer contacts* | *Number of articles/ customer* | *Number of order lines* |
| C44A2 | Return papers | 20 | 2 | 40 |
| C44A3 | Order papers | 800 | 20 | 16 000 |
| C4B1 | Telephone contacts with customer: | | | |
| | –No orders | 4 | 20 | 80 |
| | –Abnormal orders | 4 | 5 | 20 |
| C5B11 | Direct order changes | 8 | 5 | 40 |
| C5B12 | Order of extra delivery | 40 | 10 | 400 |
| Sum | | 876 | – | 16 580 |

| Activity | | Properties | |
|---|---|---|---|
| *Reference code* | *Name* | *Number of Personnel* | *Hours of Business* |
| C41 | Order processing (order personnel) | 6 | 9 am–4 pm |

Source. Investigation of the effects of the current order system at the Charlestown Dairy December 1975.

Figure 3-41

| | | |
|---|---|---|
| O1 | High level of customer service | The order processing, invoicing and information distribution should be considered as a service instrument and thereby gives the customer confidence in DAIRCO. |
| O2 | Suitable planning tools | Planning tools that facilitate a rational flow of products from farmer to consumer should be developed and maintained. |
| O3 | High level of work satisfaction | Stimulating work tasks should be strived for; boring and laborious manual work tasks should be avoided. |
| O4 | Coordinated activities | The activities in the dairies of DAIRCO should be coordinated with due regard to possible differences in ambition levels between large and small dairies. |
| O5 | Suitable equipment | Equipment for material processing and data entry that is adapted to users' needs and technological development in these areas should be purchased and maintained. |
| O6 | Profitable activities | Operating costs must permit acceptable prices for the farmers and a suitable investment level. |

Figure 3-42

| *Needs for changes (project goals)* | | | *Problem (figure 3.3.1)* | *Objective (figure 3.3.11)* | *Priority* |
|---|---|---|---|---|---|
| N1 | Better order procedures | Simpler, faster, and more accurate order procedures for customers | P1 | O1, O6 | 1 |
| N2 | Better distribution basis | Simpler and faster distribution basis via summaries of customer orders | P3 | O2, O6 | 2 |
| N3 | More effective order office work | Rationalization of laborious order summaries at the order offices | P4 | O3, O6 | 1 |
| N4 | Common order system | A common order system that can be extended to fulfill different levels of ambition | P6 | O4, O6 | 1 |
| N5 | Better order entry equipment | | P7 | O5, O6 | 1 |
| N6 | Better production planning basis | Faster and better aids for production planning in form of suitable prognoses based on customer order statistics | P10 | O2, O6 | 3 |

Figure 3-43

Step 2: Study of change alternatives

First, alternative means of meeting the needs for change are considered and listed in an alternatives table (see figure 3-44). Each alternative is then investigated, by means of A-graphs (with associated text pages) and property tables. Social and economic evaluations of each alternative are carried out.

Step 3: Choice of change approach

A choice is made between various alternatives identified in step 2. The chosen alternative is further documented by, among other things, time schedules and resource plans.

Step 4: Partitioning into information subsystems

The A-graphs for the alternative chosen in step 3 are now decomposed into greater detail, to a level at which (subjectively) subsystems are identified. The A-graphs are as usual accompanied by text pages and property tables. Then all subsystems are assessed for "formalizability" in a special property table (see figure 3-45).

Step 5: Study of information subsystems

Each subsystem is now studied in more detail. More detailed A-graphs are produced. Special property tables show properties such as contributions (see figure 3-46), prerequisites and requirements (see figure 3-47), and the results of cost/benefit analysis (see figure 3-48).

| | | |
|---|---|---|
| A0 | Current manner (driver's order) | The driver brings customer orders to the order office. These are considered for the next day's delivery. This is the current system and thus represents the zero alternative. |
| A1 | Telephone order | A pure telephone order system can be developed in two different ways:<br>—the order office calls the customers.<br>—the customers call the order office. |
| A2 | Purchase proposal | A prognosis (forecast) adapted to each customer is produced and mailed from the central office, e.g., once a week. This purchase proposal is structured per article and delivery day within the prognosis week. In such cases when the customer is not satisfied with the prognosis, the customer calls the order office the day before delivery and gives the changes to the proposal. |

Figure 3-44

| *Information systems* | | *Property: Type of information processing* | | | |
|---|---|---|---|---|---|
| | | *Formalizable parts* | | | |
| | | *Automatable parts* | | *Naturally manual parts* | *Non-formalizable parts (manual)* |
| *Reference code* | *Name* | *Calculations* | *Transport of messages only* | | |
| PP41 | *Order processing:* | | | | |
| PP411 | Prognosis processing | X | | | |
| PP412 | Order receiving and order summarizing | | | | |
| PP4121 | Order receiving | | | X | |
| PP4122 | Order summarizing | X | | | |
| PP4123 | Filing of delivery papers | | X | | |
| PP413 | Return processing | X | | | |
| PP42 | *Production.* | | | | |
| PP423 | Producing planning | | | | X |

A practical test of formalizability.

Figure 3-45

| *Activity* | | | | *Measure results* | | |
|---|---|---|---|---|---|---|
| *Reference code* | *Contribution (benefits) to sub-activity* | *Measure* | *Measure unit* | *Feb.–76 current* | *Mar–76 "simple" prognosis model* | *Apr–76 "complex" prognosis model* |
| PP44 | *Store handling and distribution:* | | | | | |
| PP441 | 1 Out-of-stock | Stock-taking | Number of occasions/ week | 10 | 4 | 8 |
| PP441 | 2 Stock surplus | Stock-taking | Number of occasions/ week | 15 | 7 | 5 |
| PP442 | 3 Rapidness in distribution basis | Availability before start of distribution | Number of hours | 1/2 | 3 | 2 1/2 |
| PP442 | 4 Variation in order flow (⇨ work load) | Difference between maximum and minimum/ month | Number of ordered units (in thousands) | 80 | 30 | 50 |
| PP442 | 5 Personnel attitudes of drivers, transport leaders | Interviews | % positive | 5 | 80 | 10 |

Extract from table of contributions

Figure 3-46

| *Prerequisites for order processing (PP41):* | | |
|---|---|---|
| P1 | Correct information of sufficiently high quality must be given to prognosis processing in time. | In the A-graph PP41 (figure 4.3.2) that contains prognosis processing (PP411) we find two input sets: guidelines (PP3B23) and information about adjusted daily delivery (PP413A). Guidelines for prognosis preparation must be available at each preparation occasion. Before we arrive at an adjusted daily delivery (PP413A) there are several error possibilities, e.g., an erroneous change (PP5B1) of purchase proposal (PP411A11) or an erroneous entry of returns (PP413). Good motivation on the part of the customers and the order personnel is a necessary prerequisite for correct prognoses. |
| P2 | Prognosis models (calculation methods) that "forecast" the outcome with acceptable accuracy. | |
| P3 | The volume of the change contacts may not exceed approximately 30%. | In such cases other types of order systems will be more profitable (see figure 3.3.18). 30% can be seen as a maximal load when determining the size of the telephone order receiving personnel. |

| *Set* | | *Requirements (properties)* | | | |
|---|---|---|---|---|---|
| *Reference code* | *Name* | *Frequency* | *Extent of prognosis period* | *Number of customers* | *Age of underlying sales statistics (PP413A)* |
| PP4B1 | Purchase proposal to customer | At least once/ week | Delivery days one week ahead | 24 000 (out of a total of 27 000) and 800 (out of a total of 900) per average dairy | Maximum one week old |

Tables of prerequisites and requirements.

Figure 3-47

| *Information system:* Prognosis processing (PP411)<br>*Property:* Cost/benefit calculus | *"Simple" prognosis model (thousand $/year)* | *"Complex" prognosis model (thousand $/year)* |
|---|---|---|
| *Separate benefits per typical dairy and year:* | | |
| 1 Direct monetary benefits (based on figure 4.3 10 among others): | | |
| —Personnel savings at order office (salary costs per year 14 000 $) | 3 * 14 = 42 | 2 * 14 = 28 |
| —Savings at overtime protion (8 $ per hour) | (100 - 10) * * 52 * 0.008 = 37 | (100 - 30) * * 52 * 0.008 = 29 |
| —Out-of-stock in cold storage (appr. 100 $ sales loss per occasion) | (10 - 4) * * 52 * 0.1 = 31 | (10 - 8) * * 52 * 0.1 = 10 |
| —Stock surplus in cold storage (appr. 60 $ loss per surplus occasion) | (15 - 7) * * 52 * 0.06 = 25 | (15 - 5) * * 52 * 0.06 = 31 |
| 2 Nonmonetary benefits such as<br>—Improvements at the customers<br>—Personnel attitudes and social effects<br>—The effects of the variation of the order flow on distribution loads | | |
| Sum of separate benefits | 135 | 98 |

Cost/benefit calculus.

Figure 3-48

Step 6: Coordination of information subsystems

The main task in this step is to rank subsystems in priority order for development.

Step 7: Precedence and component analysis

Precedence and component analysis is carried out for each subsystem in turn. Precedence analysis shows how the outputs from a subsystem are derived from its inputs and is represented in I-graphs (see figure 3-49). Component analysis shows the composition of message sets, and is represented in C-graphs (see figure 3-50). As in the case of A-graphs, I-graphs are accompanied by text pages. The atomic items, or terms, identified in C-graphs are entered into a table of terms (or data dictionary) (see figure 3-51).

Step 8: Process analysis

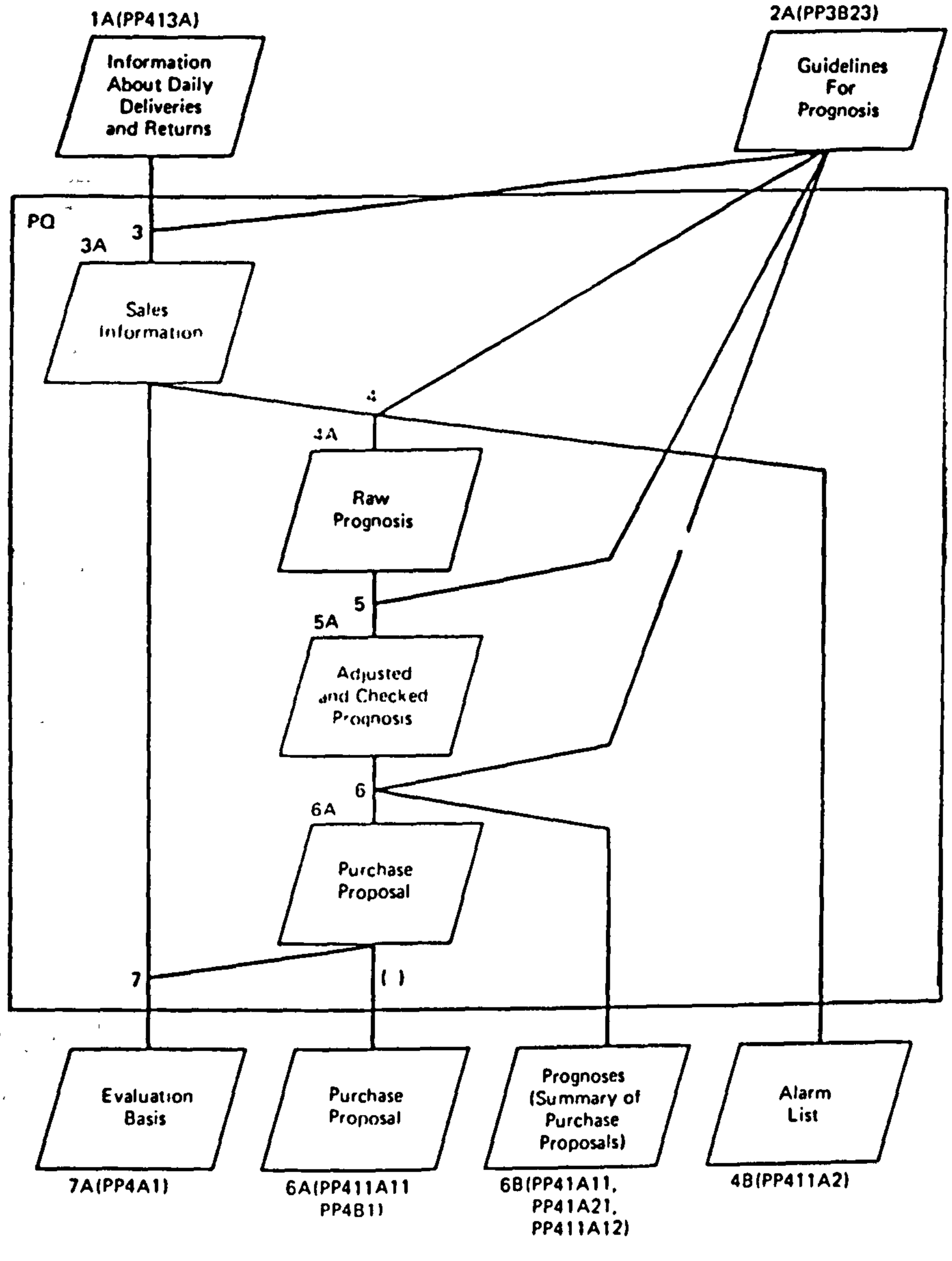
The processes (or functions) to be performed in a subsystem are now listed in a process list (see figure 3-52), and each process defined in a process table using (where appropriate) decision table techniques (see figure 3-53).

Step 9: Property analysis

The measurable features of the subsystems as thus far specified are recorded in further property tables (see figure 3-54).

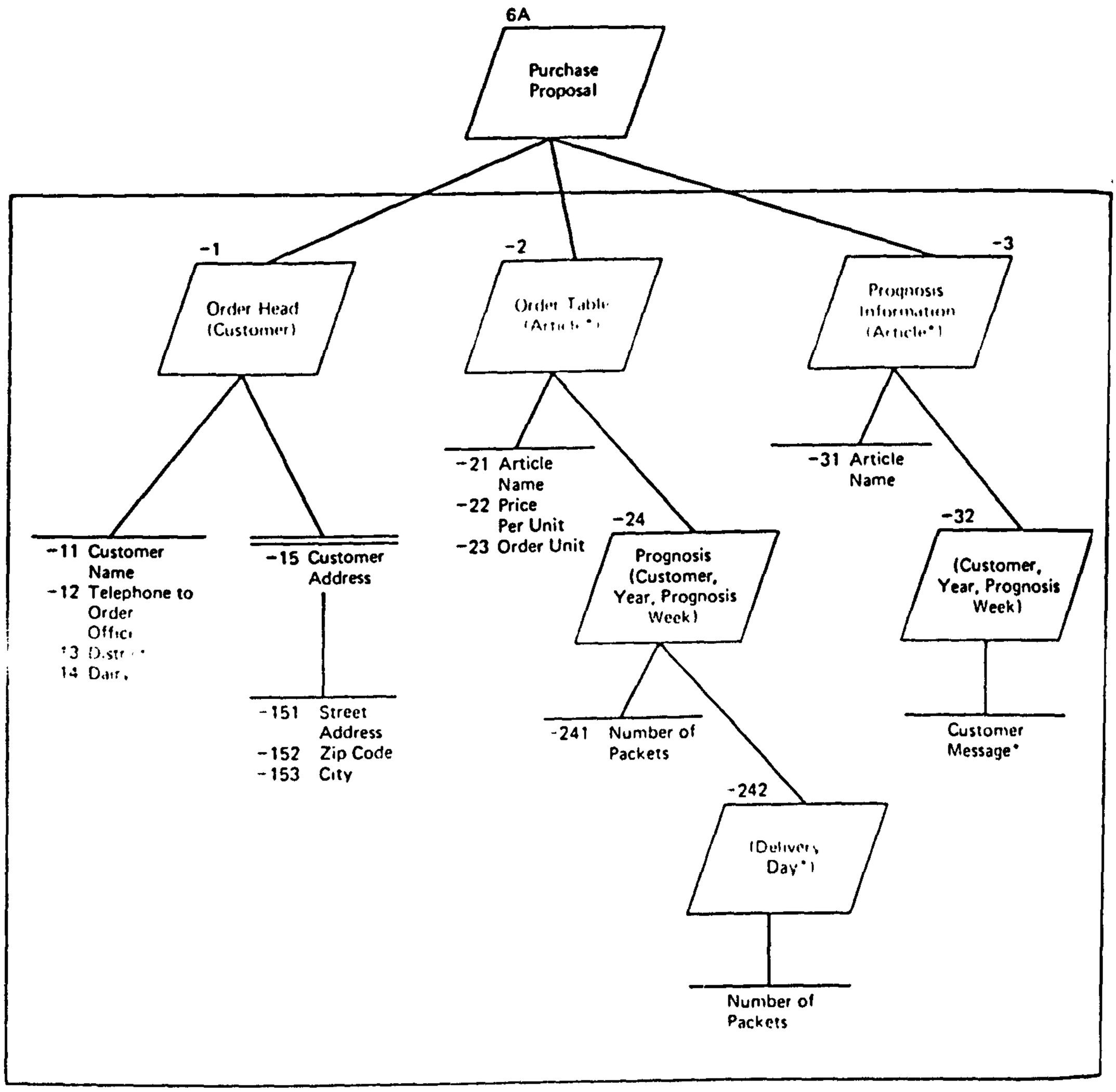
Step 10: Determine processing philosophy

"Processing philosophies" include, for instance, manual, computer




I-graph PQ0.

Figure 3-49




G-graph for purchase proposal.

Figure 3-50

| *Term* | *Information set* | *Data term* | *Type*[1] | *Number of occurrences* | *Value scope* | *Sort* |
|---|---|---|---|---|---|---|
| Article | PQ6A | Article number | ID | | See article file | |
| Customer | | | | 27 000 | See customer file | |
| | PQ6A | Customer number | ID | Appr. 24 000 | See customer file | |
| Dairy | PQ6A14 | Dairy name | P | 30 | | |
| | PQ6B | Dairy name | ID | 30 | | |
| District | PQ6A13 | District name | P | Maximum 20 per dairy | See district catalogue for dairy in question | |
| | PQ6B | District number | ID | | 1–20 | |
| Model type | PQ2A4 | | P | 2 | Holiday week, normal week | |
| Number of packets | PQ6A241 | | P | | 0–30 000 | Piece |
| | PQ6A242 | | P | | 0–5 000 | Piece |
| Order unit | PQ6A23 | | P | | 0 · 500 | (even) multiple of number of packets |
| Price per unit | PQ6A22 | | P | | 0 00–9 99 | Dollars + cents per packet |
| Production quantity | PQ6B25 | | P | | | Packet unit litre |
| | PQ6B32 | | P | | | |

[1] ID = identification term  P = property term.

Figure 3-51

| *Reference code* | *Name* |
|---|---|
| ' | ' |
| ' | ' |
| ' | ' |
| PQ42 | Selection of purchase proposal customers and production of alarm list |
| PQ43 | Calculation of raw prognosis per week |
| PQ44 | Spreading of raw prognosis to delivery days |
| ' | ' |
| ' | ' |
| ' | ' |
| ' | ' |

Figure 3-52

| *Prerequisites* |
|---|
| Find a message in 2A4. |
| Find messages in 41A for the *same* customer. |

| *Calculations* | *1* | *2* | *3* |
|---|---|---|---|
| Prognosis customer (41A11 | N | Y | Y |
| First prognosis week (41A121) ≤ < Prognosis week (2A4) ≤ < Last prognosis week (41A122) | | N | Y |
| 4B–5 := 41A2 | X | X | |
| Prognosis week (4B6) := Prognosis week (2A4) | X | X | |
| Reason (4B6) := "Not prognosis customer" | X | | |
| Reason (4B6) := "Outside prognosis interval" | | X | |
| Prognosis week (42A) := Prognosis week (2A4) | | | X |
| Customer (42A) := Customer (41A) | | | X |

Process table 1

Figure 3-53

| *Information sets* | | *Properties* | | |
|---|---|---|---|---|
| *Reference code* | *Name* | *Frequency* | *Latest time of completion* | *Sort order* |
| PQ4B | Alarm list | Once/week | Thursday 3 pm | 1) Dairy<br>2) District<br>3) Customer |
| PQ6A | *Purchase proposal* | Once/week | Thursday 1 pm | 1) Dairy<br>2) District<br>3) Customer<br>4) Article |
| PQ7A | Evaluation basis | Once/month | First Monday in every month | 1) Dairy<br>2) District<br>3) Customer<br>4) Article<br>5) Week<br>6) Delivery day |

Figure 3-54

batch, computer direct, etc. Appropriate decisions are made for each subsystem.

Step 11: Design computer-based routines

Like step 1, this step involves the use of many representations. First are D-graphs (data system design graphs), roughly equivalent to conventional program run charts (see figure 3-55). They identify data sets (of various kinds) and programs. These are then designed. For data, first a data set description is produced showing the contents of each record type (see figure 3-56) and then D-structure diagrams are produced for each record type. A D-structure diagram (see figure 3-57) is taken straight from Jackson's JSP. For programs, first a program/process list is produced showing, for each program, which processes it incorporates (see figure 3-58) and then P-structure diagrams are produced for each program. A P-structure diagram is also taken straight from Jackson's JSP, although there there is no discussion about whether Jackson's techniques (eg inversion) are used to derive P-structures from D-structures. ISAC does follow JSP, however, in that the P-structures are derived in three stages: first a control structure is produced (see figure 3-59), then a list of operations (see figure 3-60) and then a final program structure with the operations attached to the control structure (see figure 3-61).

Step 12: Design manual routines

The representation used in this step is the work task table (see figure 3-62).

Figure 3-55

Data Set

Sequential Data Set

Data Set with Direct Access

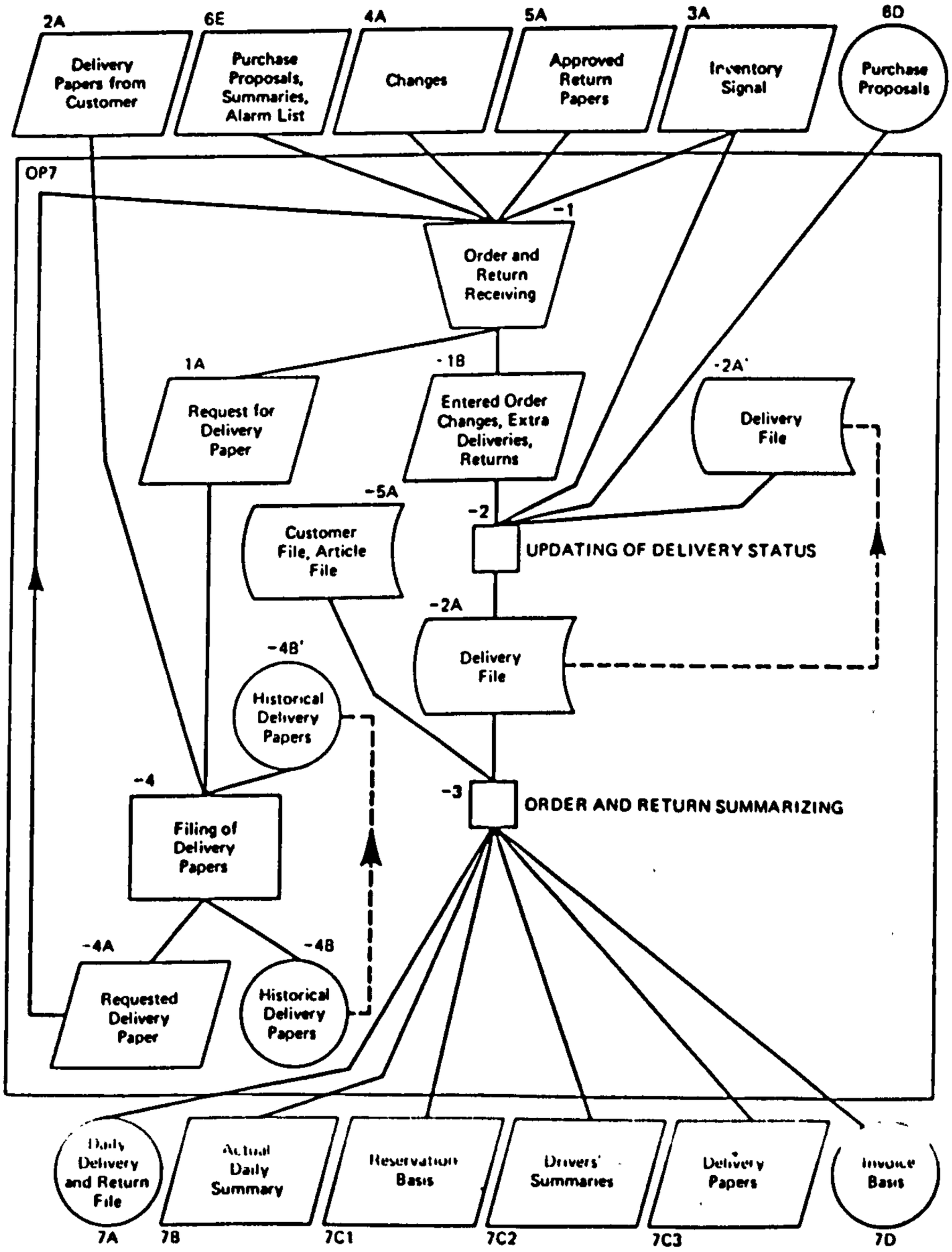
Data Set with Direct and Sequential Access

Computer Process

Manual Process

Computer Program

Subject:
Order processing:
Decentralized order processing



| *File/record type* | *Identification data terms* | *Property data terms* | *Sort order* | *Access method/ search path* |
|---|---|---|---|---|
| Customer | Customer number | Customer name<br>Delivery address<br>Customer telephone number<br>Dairy number<br>District number<br>Prognosis code<br>Prognosis interval<br>Telephone number of the order office | 1) Dairy number<br>2) District number<br>3) Customer number<br>Rising | –Direct<br>–Sequential |
| Article | Article number | Article name<br>Price per unit<br>Size of packet<br>Type of packet<br>Smallest order unit<br>Campaign addition<br>Check number | Article number<br>Rising | –Direct<br>–Sequential<br>–Via customer/ article |
| Customer/ article | Customer number<br>Article number | | Within each customer linked in article number order | –Via customer<br>–Via article |
| Week record | Customer number<br>Article number<br>Week number | Number of packets for 6 delivery days | Within each customer linked in week number order | –Via customer/ article in both directions |

Figure 3-56

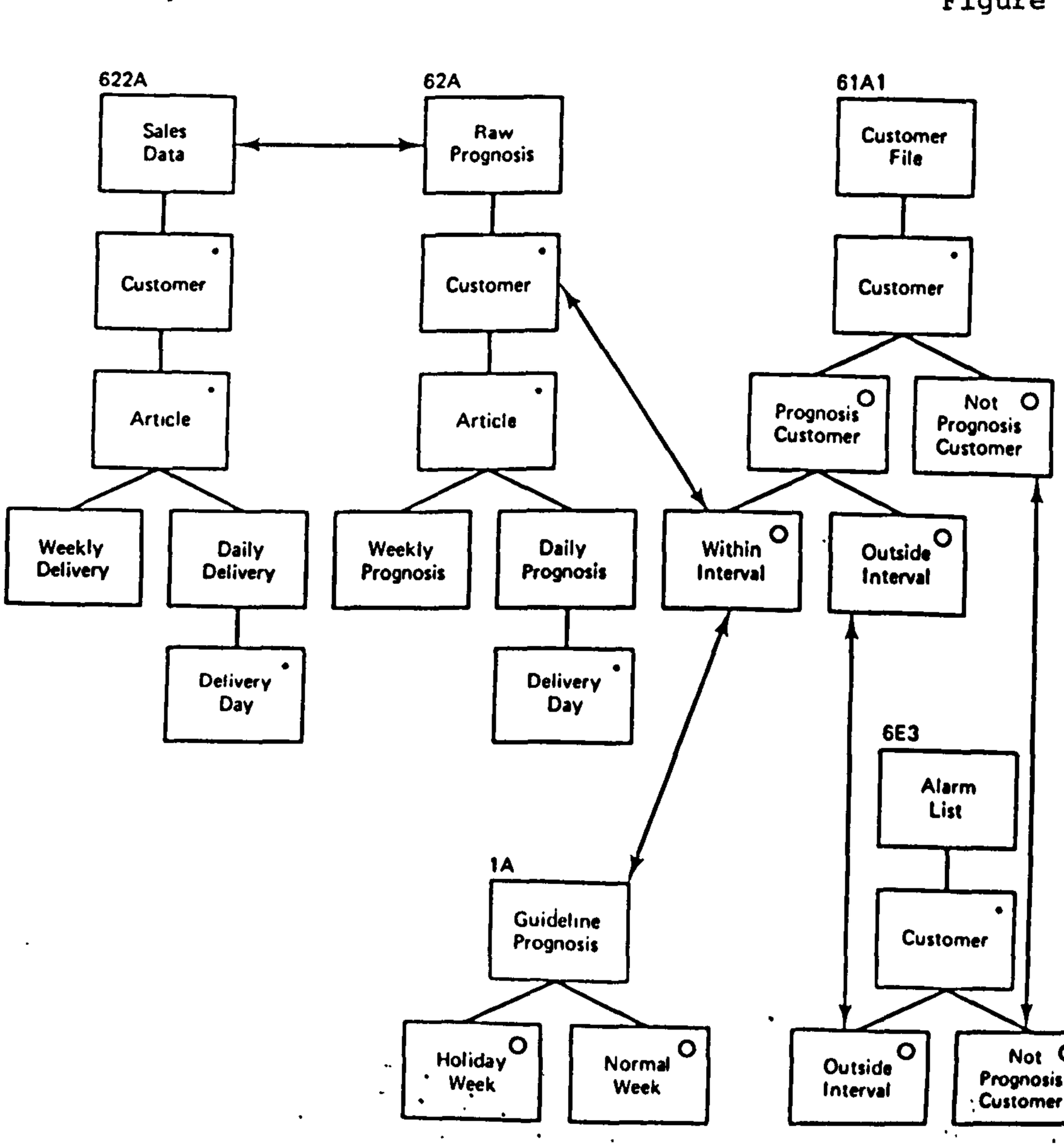



Figure 3-57

| *Prognosis processing* | | |
|---|---|---|
| *Program* | | *Processes from the information analysis model* |
| *Reference code* | *Name* | |
| OP61 | Storing of sales data | PQ31, PQ32 |
| OP62 | Raw prognosis calculation | PQ33, PQ42, PQ43, PQ44 |
| OP63 | Production of purchase proposals | PQ51, PQ53, PQ55, PQ57, PQ62, PQ64, PQ66 |
| OP64 | Preparation of evaluation basis | PQ72, PQ74 .. |

List of programs/processes for prognosis proc-

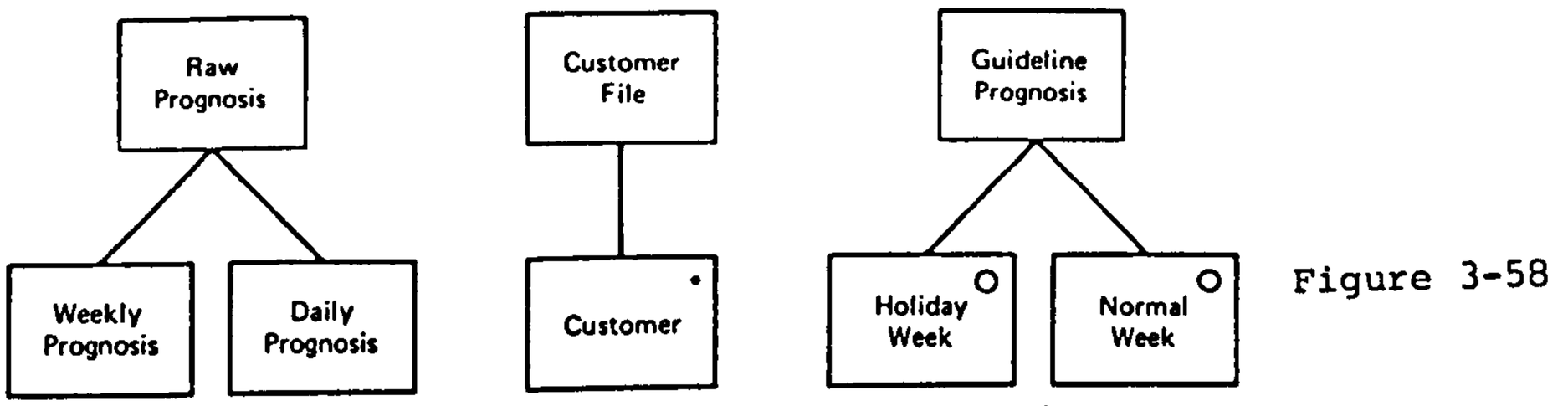



Figure 3-58

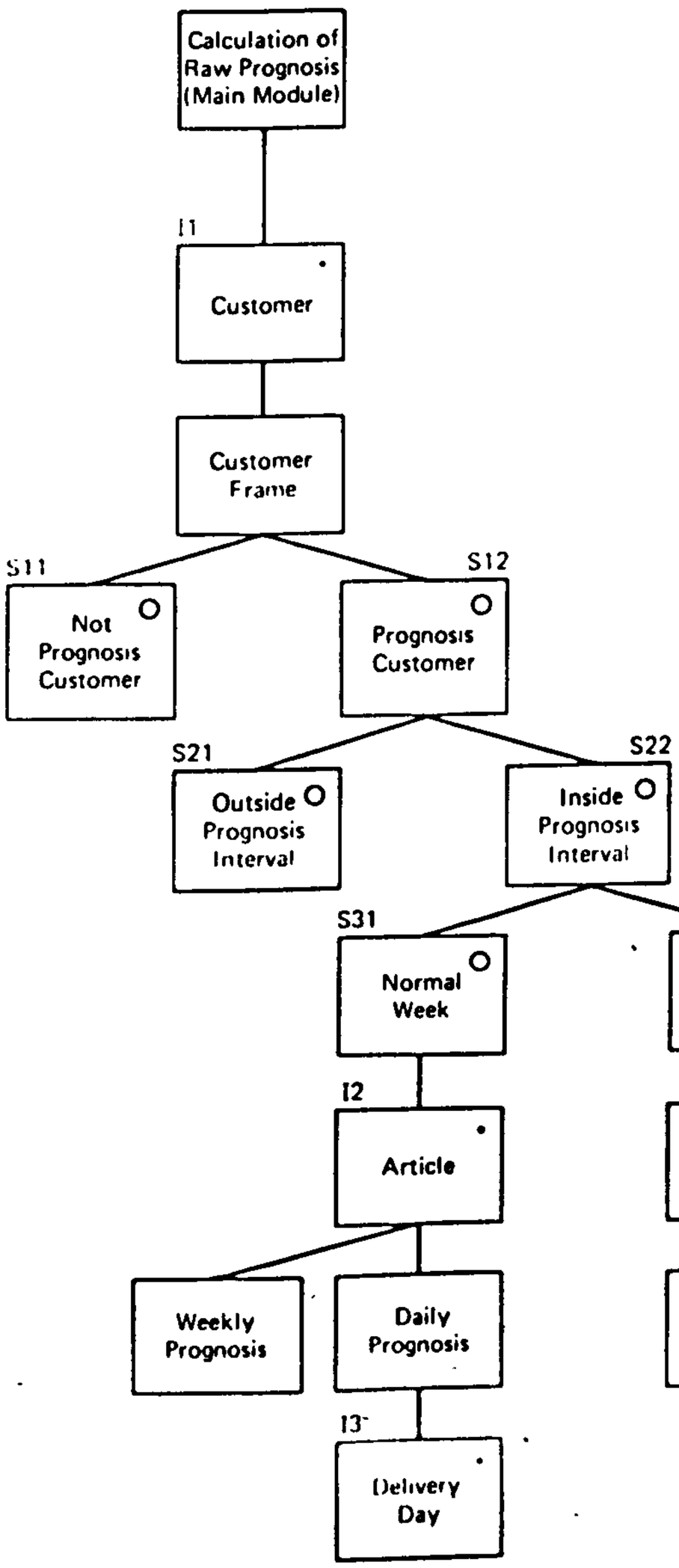



Figure 3-59

**Calculation operations**

1. Customer number (OP62A) := Customer number (OP61A)
2. Article number (OP62A) := Article number (OP61A)
3. Prognosis week (OP62A) := Prognosis week (OP1A)
4. Delivery day (OP62A) := Delivery day (OP622A)

16. Reason (OP6E3) := "Not prognosis customer"
17. Reason (OP6E3) := "Outside prognosis interval"

**Input/output operations**

20 Read guidelines (OP1A)
21 Read customer file (OP61A)
22 Read sales data (OP622A)
23 Write alarm list (OP6E3)
24 Write raw prognosis (OP62A)
25 Termination

**Conditions**

***Iteration conditions***

I1 Until end of customer file
I2 Until end of customer/article records (for certain customer) or until end of customer file
I3 Until delivery day counter > 6

*Selection conditions*

S11 Prognosis code (OP61A) = "No"
S12 Prognosis code (OP61A) = "Yes"
S21 First prognosis week (OP61A > Prognosis week (OP1A) or
Last prognosis week (OP61A) < Prognosis week (OP1A)
S22 First prognosis week (OP61A) ≤ Prognosis week (OP1A) or
Last prognosis week (OP61A) ≥ Prognosis week (OP1A)
S31 Model type (OP1A) = "Normal week"
S32 Model type (OP1A) = "Holiday week"

**Conditions operations**

26 Delivery day counter := 0
27 Delivery day counter := Delivery day counter +1

Operations list for raw prognosis calculation.

Figure 3-60

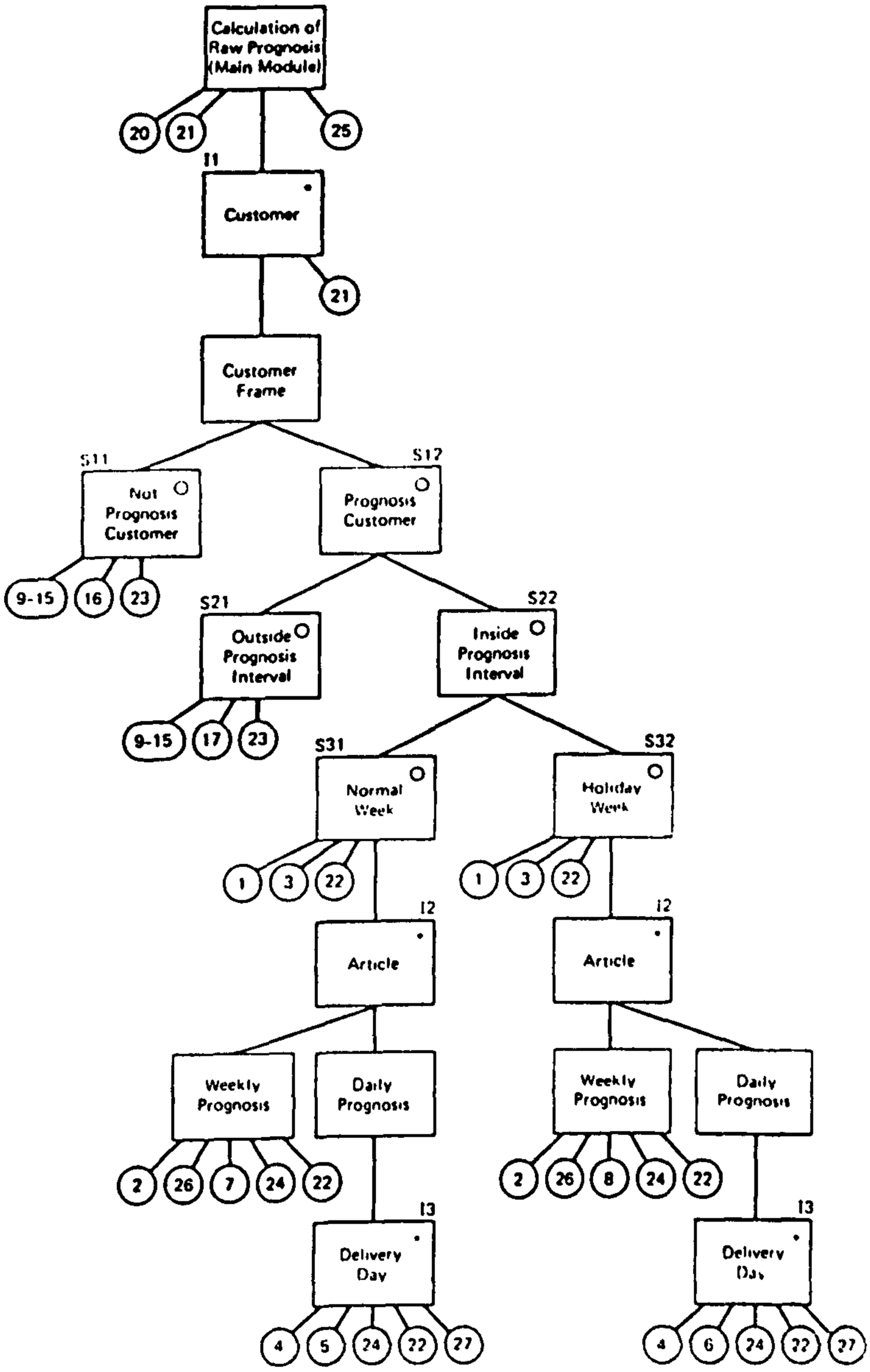

Calculation of Raw Prognosis (Main Module)
20
21
25
11
Customer
21
Customer Frame
S11
Not Prognosis Customer
S12
Prognosis Customer
9-15
16
23
S21
Outside Prognosis Interval
S22
Inside Prognosis Interval
9-15
17
23
S31
Normal Week
S32
Holiday Week
1
3
22
1
3
22
12
Article
12
Article
Weekly Prognosis
Daily Prognosis
Weekly Prognosis
Daily Prognosis
2
26
7
24
22
2
26
8
24
22
13
Delivery Day
13
Delivery Day
4
5
24
22
27
4
6
24
22
27

Figure 3-61

*Main Order Receiving*

1 A customer calls (PP5B1, OP4A)

2 Determine if it is a
- direct change of purchase proposal (PP5B11, OP4A1)
- order of extra delivery (PP5B12, OP4A2)
extra deliveries are issued for "alarm list" customers or ordinary customers who call too late, i.e., after the order summarizing (PP4122, OP73) has started.

*Change of Purchase Proposal (PP5B11, OP4A1)*

1 Ask for customer name and number

2 Find copy of purchase proposal (PP411A11, OP6E1)

For each article the customer wants to change:

3 Ask for desired quantity change (i.e., not the new order quantity) (PP5B11, OP4A1)

4 Check the reasonableness (±50%). If "unreasonable" ask the customer if it is correct

5 Check if the inventory signal (PP44A3 OP3A) can affect the change

6 Enter definite change on
type writer terminal (PP4121B1, OP71B1)
purchase proposal copy (PP411A11, OP7E1)
for "back-up" reasons

*Extra Delivery (PP5B12, OP4A2)*

1 Ask for customer name and number

2 If "alarm list customer" make a mark on the alarm list (PP411A2, OP6E3)

For each article the customer wants to change:

3 Ask for desired extra delivery quantity (PP5B12, OP4A2)

4 Check if the inventory signal (PP44A3 OP3A) can affect the order

5 Enter definite order (PP4121B2, PP4121B3, OP71B2) on typewriter terminal

*Background Work Order Receiving*

1 Enter approved return papers (PP4A2, OP5A) on typewriter terminal (OP71B3)

2 Analyze purchase proposal summaries (PP411A12, OP6E2) if they seem reasonable. The "know-how" of the order personnel is used in the study of old delivery papers (PP4123A, OP74A)

Figure 3-62

Step 13: Equipment study

In this step, the record types and programs designed in step 11 are mapped onto physical equipment. This is done by means of E-graphs (equipment graphs) (see figures 3-63 and 3-64).

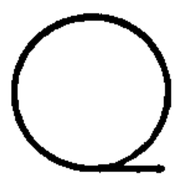
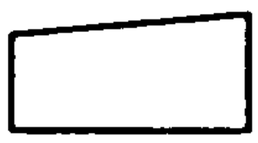
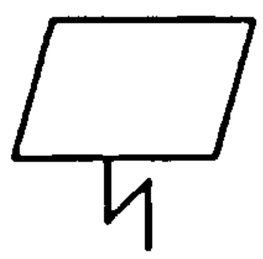
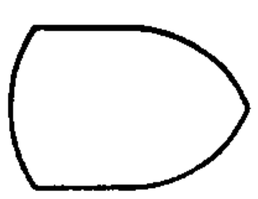
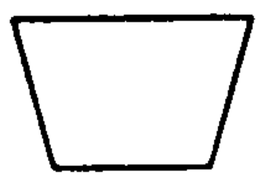
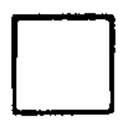
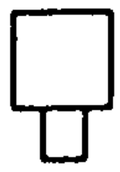
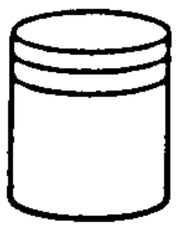
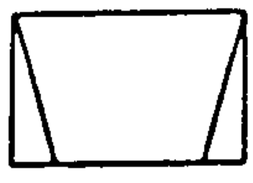
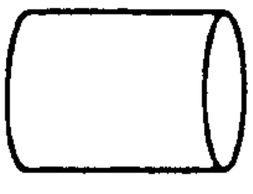
Step 14: Adaptation of computer-based routines

This step is primarily concerned with detailed physical layouts of inputs/outputs (see figure 3-65) and records (see figure 3-66).

Step 15: Creation of side routines

This step defines the manual tasks to be performed in conjunction with the computer-based routines (eg. operation, data control). They are recorded in the form of work descriptions (see figure 3-67).

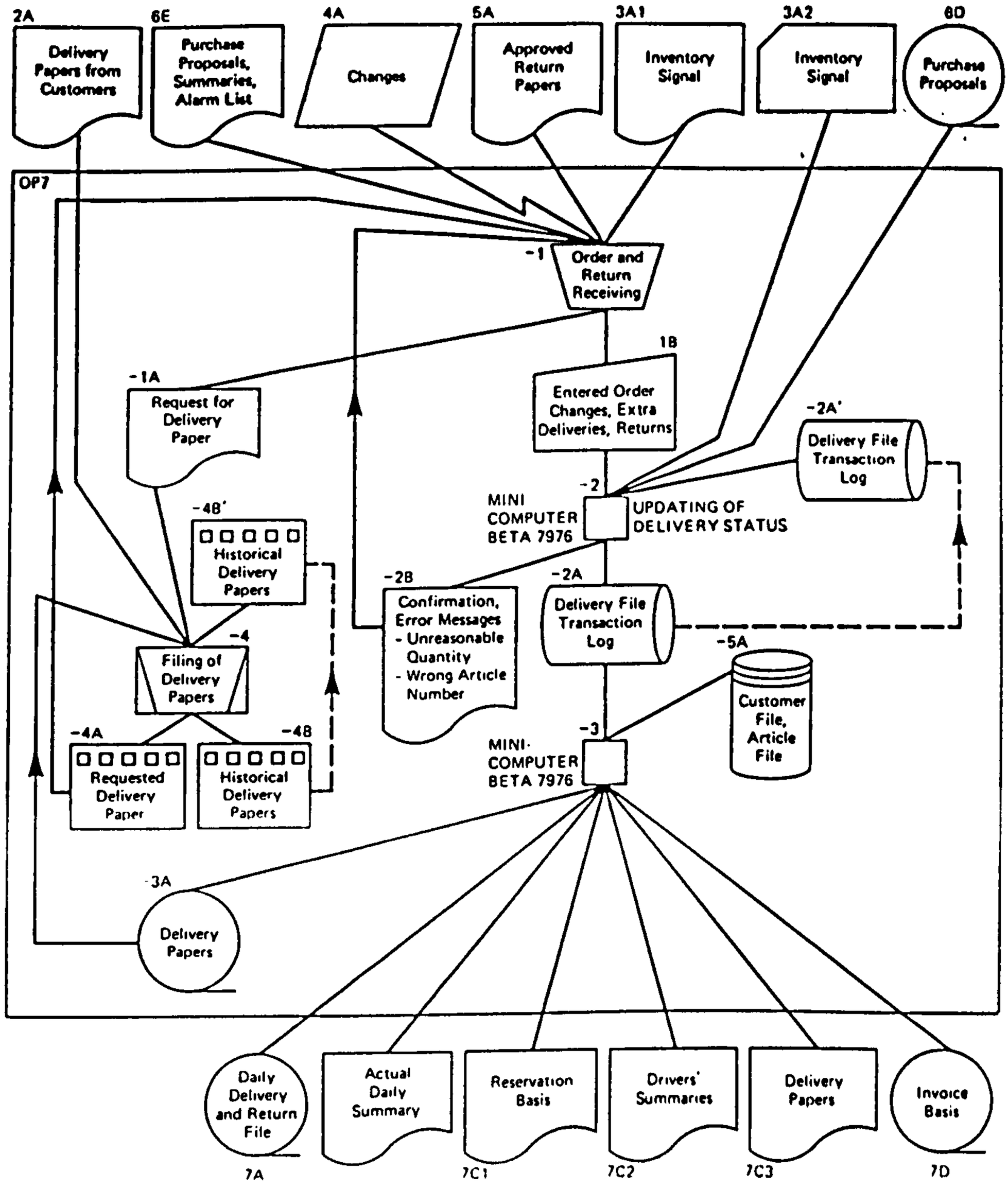
As already indicated, ISAC is notable for its comprehensive coverage, particularly for the fact that it covers the study of both the human activity system and the designed system in unusual detail. It stresses the user-view in addition to the developer's viewpoint; the project management view-point is also accomodated, but not so strongly as the other two. ISAC draws from a variety of different approaches, including Langefors (precedence and component analysis), Jackson, decision tables and cost-benefit analysis. A-graphs are similar to data flow diagrams, and there is a strong flavour of functional decomposition throughout the methodology. ISAC seems to be widely used and well accepted.

| SYMBOLS IN E-GRAPHS | CORRESPONDENCE IN DESCRIBED SYSTEM | SYMBOLS IN E-GRAPHS | CORRESPONDENCE IN DESCRIBED SYSTEM |
|---|---|---|---|
|  | *Document (List)* |  | *Tape Memory* |
|  | *CRT-Terminal (Input)* |  | *Telephone Message* |
|  | *CRT-Terminal (Output)* |  | *Computer Processing* |
|  | *Punched Card* |  | *Manual Routine* |
|  | *Punched Paper* |  | *Computer Program* |
|  | *Microfiche* |  | *Computer Program with Adaptation Routine* |
|  | *Disc Memory* |  | *Combined Manual and Computer Processing* |
|  | *Drum Memory* |  | *Relation* |
| | |  | *Indication of Permanent Storage Media* |

All flows are assumed to go from top to bottom on the graphs, arrows are needed on upward and (possibly) horizontal flows only.

Explanation of symbols used in E-graphs.

Figure 3-63




E-graph of the decentralized order processing.

Figure 3-64

| | | | | | |
|---|---|---|---|---|---|
| Dairco | Alarm List | | Prognosis Week 727 | | HEADING 1 |
| Dairy | Charlestown | | District 40 | | HEADING 2 |
| Customer Number | Name | Address | Telephone | Reason | HEADING 3 |
| 4085 | Isac's Market | 1.Midas Street 12345 Isactown | 468-015-0160 | Outside Prognosis Interval | CUSTOMER LINES |
| 4086 | The Milk Bar | 4.Cream Street 12345 Isactown | 468-787-7000 | Not Prognosis Customer | |
| – | – | – | – | – | |
| – | – | – | – | – | |

FORMAT: LINE PRINTER LIST, STANDARD SIZE
SORT ORDER: DAIRY NAME, DISTRICT NUMBER, CUSTOMER NUMBER
NUMBER OF RECORDS/PAGE: 20 CUSTOMER LINES

Figure 3-65

| *Data term* | *Number of occurrences* | *Number of characters* | *Type of characters* |
|---|---|---|---|
| Customer number | 1 | 4 | Numeric |
| Article number | 1 | 3 | Numeric |
| Prognosis week | 1 | 4 | Numeric |
| Number of packets/ week | 1 | 4 | Numeric |
| Number of packets/ day | 6 | 5 | Numeric |
| Record length | 45 characters | | |
| Block size | 20 records = 900 characters | | |

Figure 3-66

| *Preparation of evaluation basis (OP64)* |
|---|
| 1 Get the 4 or 5 different weekly tape series of purchase proposal (OP63A) that concern the current monthly processing for one dairy at a time. |
| 2 Get the disc packs of sales data (OP61A) that concern customers for the dairy that shall be processed. |
| 3 Mount/demount necessary tapes and disc packs during the run. |
| 4 See to it that the data lists for evaluation basis (OP6C) are postprocessed. |

Figure 3-67

## 3.9 CONCLUSION

The detailed summary of six selected methodologies, in section 3.3 to 3.8, permits a number of conclusions to be drawn.

There is no agreement on development models (as manifested by the step structure of methodologies).

There is no agreement on product system models (as manifested by the representations of systems produced at various stages of the methodologies).

There is great diversity of representations (though some recur: for instance decision tables, data flow diagrams in several guises, data dictionaries) and of terminology.

The relational model and functional decomposition occur fairly frequently. Methodologies tend to have one (sometimes a few) key concept(s) - for instance triggers, entity life cycles, conceptual grammar, prototyping.

There is a lack of attention to important "separate concerns" such as performance, error management or project management.

There is a lack of effective tools to support the developer.

There is a lack of attention to verification.

The detailed summary approach seems to be more effective (though more demanding both of the author and of the reader) as a means of summarising a set of methodologies than the use of a features list (as

used in the appendices and in published comparisons). The main reason is probably that the detailed summary permits a methodology to be described in its own terms, subject to the imposition of only the broadest framework, rather than under a number of headings which may be more or less appropriate and which may conceal its most important characteristics.

# CHAPTER 4

# APPROACHES TO THE DEVELOPMENT OF METHODOLOGIES

## CONTENTS

4.1 Introduction

4.2 Approaches observable in the methodologies surveyed

4.3 Other relevant viewpoints

4.4 Conclusion

## 4.1 INTRODUCTION

The purpose of this chapter is to indicate the variety of viewpoints which might be relevant to the development of an information system development methodology.

Section 4.2 proposes a number of approaches which can be observed as underlying some of the methodologies surveyed in Appendix-B. This set of approaches was generated as follows. Each of the entries in Appendix-B was studied to see whether it suggested any candidate viewpoints. The resulting list of candidate viewpoints was then reviewed to eliminate synonyms and to merge viewpoints which significantly overlapped. The result was a set of nine viewpoints, each of which represents a background set of ideas which authors of methodologies have brought to bear upon their task. (It is often the case, of course, that a particular methodology can be seen to be based on more than a single viewpoint.)

While this classification tries to be reasonably empirical, insofar as it is based on an analysis of existing methodologies, it nevertheless has obviously a strong subjective element, (a) because it is based upon a subjective evaluation of the methodologies surveyed, (b) because subjective judgement was used for the final selection of categories.

Section 4.3 presents a smaller set of approaches which either can be seen to have influenced methodologies not surveyed in this thesis or which, in the author's view, could yield useful ideas for the

development of methodologies. This section is necessarily more subjective and speculative in nature than section 4.2.

## 4.2 APPROACHES OBSERVABLE IN THE METHODOLOGIES SURVEYED

The following nine broad approaches can be identified as a result of analysing the methodologies surveyed in Appendix-B.

### 1. Modelling of human activity systems

This approach is concerned with the description, or modelling, of organisations in terms of the activities of individuals or groups, the information objects which they use (e.g. forms, files), existing information systems regarded as black boxes, and the flow of information between people and between peopole and systems. It is an approach which views organisations in terms of human activities, flows and stores of information, and is distinct from viewpoints which see the organisation in mechanistic terms (see 10 below) or which model entities and/or events within an organisation (see 4 below). Using the terminology of CHECKLAND (1981), such descriptions are soft systems models: they are relatively informal and in general it is not possible to attach metrics to them or to carry out formal manipulations on them.

Methodologies illustrating this approach include: CORE, EDM, DADES, ISAC, NIAM.

### 2. Formal problem/requirement specification

This approach is concerned with describing the external characteristics required of a designed system. Not surprisingly it underlies many methodologies. Individual approaches may vary

according to, for instance, what should be included in such a description and the type of notation (graphic, mathematical etc.) in which it should be expressed. It is common to all instances of this approach that the specification is seen as being distilled from a variety of sources of informal information about system requirements. In most cases this distillation is to be carried out by systems developers free of any constraints. In a methodology such as LEGOL, however, the distillation process is based on information which is already fairly well structured, and has to be carried out in a fairly systematic and constrained manner. Yet again, there are those who envisage the possibility of creating formal specifications automatically as the output of a natural language understanding process.

Methodologies illustrating this approach include: ASSET, ADS, CASCADE, ACM/PCM, CORE, DADES, EDM, HOS, ISAC, INFORMATION ALGEBRA, LBMS-SDM, NIAM, PRISMA, PSL/PSA, REMORA, SYSDOC/SYSTEMATOR, SYSTEMATICS, SDM, SDS, TAT, YOUNG AND KENT ALGEBRA.

## 3. Mathematical modelling of designed systems

This approach is concerned with providing mathematical notation for describing the internal characteristics of designed systems, in terms say of information sets, precedence relationships or sets of axioms. It would normally the case that some useful mathematical manipulation could be performed on such system descriptions.

Instances of this approach have often been thought of as very high

level languages, abstracting from the implementation detail of conventional programming languages. In this respect they have something in common with recent developments in program specification languages and non-procedural programming languages (although they are at a level below that of formal problem requirements specification languages described in 2 above).

Methodologies illustrating this approach include: INFORMATION ALGEBRA, CASCADE, IML-INSCRIBED NETS, HOS, LANGEFORS ALGEBRA.

## 4. Conceptual schema

Database theorists and practitioners were for a long time concerned only with limited problems of designing and implementing the database itself, which is a subset of the total problem of system development. More recently, however, their recognition that a database is a model of reality has led them to an interest in that reality, which parallels the interest of system developers. In ANSI-SPARC the term "conceptual schema" was proposed to refer to the level of analysis and modelling concerned with reality, abstracting from any consideration of representation or storage.

This database approach sees reality in terms of entities, relationships between them, events involving them, and properties of these things. It is increasingly the case that this approach is being broadened to include the modelling of processes as well entities, at least as far as those processes can be defined in terms of constraints to be maintained by a DBMS.

Note that a methodology such as JSD, which explicitly ignores the traditional database approach, but nevertheless models entities and events, can properly be seen as an example of this approach.

Methodologies illustrating this approach include: ACM/PCM, CSE-DBD, CIM, DADES, D2S2, JSD, NIAM, PRISMA, REMORA, SYSDOC/SYSTEMATOR, SOLVBERG.

## 5. Data dictionaries

This approach is a means of organising information about all data items in a system, which may be regarded as an important part of any modern methodology. A brief description of data dictionary systems is presented in chapter 5. At present, data dictionary systems tend to be freestanding and to vary considerably as to the information that can be held. Their use in practice tends to be correlated with the existence of a data administration function, and to be concerned with relatively mundane (though not unimportant) problems such as controlling names and picture definitions.

Methodologies illustrating this approach include: D2S2, MASCOT, PSL/PSA, SD, TAG, USE.

## 6. Commercial program design methods

The writing and testing of programs is an unavoidable part of the system development process (irrespective of whether any methodology is used). The activity of programming is the best understood of all the

activities that constitute system development. Early development in "commercial" programming concentrated on languages and on compilers and other software tools to assist in program writing. Subsequently attention moved to the earlier activity of program design, and a number of methods were proposed (notably including Jackson's JSP, Warnier's LCP and the structured approaches of Yourdon et-al). Since program design itself depends upon the yet earlier activities of systems analysis and design, it was not surprising to find the authors of program design methods shifting their attention "backwards" to systems analysis and design. A similar shift of attention is observable in the Ada language community.

A common shortcoming of this viewpoint is the temptation to suppose that concepts and structures appropriate for program design are sufficient at the higher level of systems analysis and design.

Methodologies illustrating this approach include: JSD, GEIS, HIPO, SD.

## 7. Project management

It has been generally recognised for two decades at least that system development projects overrun estimated costs and times, and that the resulting products do not meet user's requirements. The project management approach responds to these problems by applying well established principles of management to development projects: the complete activity is decomposed into a large number of small tasks; the outputs ("deliverables") of tasks are defined, in standard forms wherever possible; traditional project scheduling and control

techniques are employed; and management decision points are specified at key stages throughout the project. This approach is entirely pragmatic, and is in distinction to those which seek to improve our understanding of the development process or to develop better software support tools. It places great emphasis on documentation and standardisation, and is often seen as imposing a big bureaucratic overhead on a project. Methodologies based on this approach tend to be used in large organisations and/or large projects.

Methodologies illustrating this approach include: LBMS, SREM, SDS.

## 8. Prototyping

This approach is based on a view which sees system development as a process which produces a succession of models of the eventual system, each model more detailed than the one before. It should then be possible to take some model from this sequence, provided it meets certain criteria of completeness and detail, and submit it to a software tool which will interpretively animate it, thus simulating at least some aspects of the behaviour of the ultimate system. In rare cases, provided it is functionally complete, development beyond the prototype stage may not be necessary; usually, however, it is necessary to proceed to normal implementation for reasons of efficiency. The real benefit, then, is that a prototype permits both developer and user to investigate the behaviour of the eventual system in advance of its implementation. It is thus a technique for considerably reducing the length of the feedback loop from developer

to user.

Some proponents of prototyping claim that it is a complete alternative to specification. This view is usually associated with the adoption of some existing software package for prototyping purposes (e.g. NOMAD, PROLOG). Other advocates recognise that, at least for systems of significant size, it remains necessary to specify before prototyping, and that prototyping is simply a very useful addition to the techniques available within the traditional system development approach.

Methodologies illustrating this approach include: USE, GEIS.

## 9. Investment appraisal

Investment appraisal refers to methods for measuring and comparing the benefits and costs associated with an investment project. If the ratio of benefits to costs is judged satisfactory, according to whatever criterion, the project should be undertaken. The biggest problems with such techniques arise with those costs and benefits which cannot, or cannot easily, be measured in money terms. Such analysis may be limited to "internal" costs and benefits - ie. those which affect only the organisation which is considering the investment. Alternatively the analysis may attempt to take into account full social costs and benefits - ie. including those external to the organisation: in this case the term cost - benefit analysis is used, particularly for public sector projects.

Cost - benefit analysis is conventionally described as being undertaken in five steps (which are equally applicable to any form of investment appraisal). They are:

(1) identification of effects;

(2) quantification of effects;

(3) monetary quantification of effects;

(4) aggregation (discounting);

(5) sensivity analysis.

The main criticism of such methods is that they feign an objectivity which they lack, in attempting to express all costs and benefits in money terms. Public sector cost - benefit analysis has specially attracted this criticism because of the visibility and large scale of the projects for which it was employed. Provided it is recognised that such decisions cannot be reduced to the terms of economic calculus, however, such techniques are the most scientific we have. They are of obvious application to information system development projects, in connection with which they are sometimes used. A methodology which aimed to be comprehensive in its support of system development activities should incorporate such techniques.

Methodologies illustrating this approach include: ISAC.

## 4.3 OTHER RELEVANT VIEWPOINTS

The following six approaches can be identified as additionally relevant to the development of methodologies.

### 10. Cybernetic modelling of organisations

This approach views an organisation in terms of control theory, where management decisions control processes, and where the network of the decisions determines the values of one or more variables which measure the performance of the organisation as a whole. The best known example of this approach is Forrester's industrial dynamics. A key characteristic of this modelling method is the identification of information flows as inputs to decisions. Its weakness is that only highly programmed decisions, based on quantitative measurements, can be represented.

### 11. Systems theory

Systems theorists study systems *per se* of any kind, seeking characteristics common to all systems or to classes of systems. They may be most concerned with the development of theory for its own sake, in which case their work is most often called general systems theory; or they may be more concerned with the applications of systems ideas within particular disciplines or problem areas, to solve problems which are not amenable to traditional "reductionist" approaches. Systems theory has been called the study of organised complexity. There have been a number of attempts to categorise systems; perhaps

the simplest and most useful is by CHECKLAND (1981), who proposes four categories - natural systems, designed physical systems, designed abstract systems, and human activity systems. He also proposes four concepts which are central to systems thinking; "the notion of whole entities which have properties as entities (emergent properties ..); the idea that the entities are themselves parts of larger similar entities, while possibly containing smaller similar entities within themselves (hierarchy ..); the idea that such entities are characterised by processes which maintain the entity and its activity in being (control ..); and the idea that, whatever other processes are necessary in the entity, there will certainly be processes in which information is communicated from one part to another, at the very minimum this being entailed in the idea 'control'".

In the USA especially, the term "systems analysis" is often used to mean the application of the systems approach to large, complex and otherwise intractable problems, with extensive resort to operational research and computing techniques. It has in general been of doubtful success.

In systems analysis as more conventionally understood, and in computer science, although systems concepts are ubiquitous, systems theory has had little or no impact. Despite the distinction of many of its practitioners, and the attraction of many of its ideas, it has not yet demonstrated that it is a practical discipline for handling the different problems which it claims to address.

## 12. Programming theory

Academic computer science has seen a considerable amount of activity in the past few years directed at providing a more formal and rigorous basis for the construction of programs. At least four strands of thought, distinct but interrelated, may be detected.

First is the use of non-procedural languages for program specification. Reasoning about the properties of programs is easier in such languages than in procedural ones, and they have the further advantage that the specification is executable, even if inefficiently. Transformation into a procedural language can be carried out if necessary. Although some non-procedural languages were developed quite early (eg. LISP), there has been a considerable recent renewal of interest in them.

Second, there is an interest in proving correctness of programs in procedural languages. This is much more difficult than proving correctness in non-procedural languages, and it is commonly thought that work in this area will remain of specialist academic interest for some time yet.

Third is an area of interest known as data abstraction. This is concerned with the provision of facilities to permit the statement of properties of abstract data types (ie. data types described quite independently of their means of representation) and of the operations associated with them, and to permit reasoning about these abstract types and operations.

Fourth is the attempt to develop notations for defining the semantics (as opposed to the syntactics) of languages.

While none of the strands of thoughts is directly observable in any of the methodologies surveyed in this thesis, and while they are intended to be applicable to the programming process rather than at the systems level, it seems most unlikely that systems development methodologies will remain uninfluenced by these important ideas, with their emphasis on specification and verification.

## 13. Application program generation

An application program generator (APG) is a member of "a class of software products .. concerned with producing data processing applications. The main objective of the APG is to enable such applications to be produced more easily, cheaply and quickly than hitherto possible". That description is from LOBELL (1983), on which the rest of this passage is based. As is apparent, APGs have similar objectives to those of prototyping. Rather than animating early design models, however, they aim to translate into executable code.

It was recognised early that there were a number of standard tasks which were common to all or most applications. Among them were sorting and reporting, and these became the subjects of successful attempts to provide program generators. Areas of later standardisation, though by means of standard packages, were teleprocessing monitors and database management. These tasks are all "house-keeping" functions, which are common to applications of all

types. What distinguishes one application from another are the procedures to be carried out. Associated with the approaches already described has often been the provision of high level language facilities (ie. above the level of COBOL) for procedure definition; these facilities might be specific to applications of a certain type or general to all types of applications.

The new generation of APGs offers an integrated means of defining a program in terms of its inputs, data files, outputs and procedures. (Sorting may be incorporated in either data definitions or procedure definitions.) LOBELL (1983) identifies sixtyseven APG products.

This practical approach is of obvious relevance to the development of methodologies, which could at least allow for the use of APGs at the programming stage. A more interesting question is the extent to which it is possible to apply the techniques developed at the program level to the system level.

## 14. Management styles

There is a considerable literature of styles of management and their effects. Specially well known is McGregor's distinction between Theory X and Theory Y management, primarily concerned with operational efficiency and with worker motivation respectively. System development involves the management of often large project teams; it involves communication with users during development; and its outcome affects users work patterns. For all these reasons the style of management employed in system development is significant. Since, to

over simplify, a Theory Y approach involves more adaptability in system design and in project planning, its adoption or otherwise is likely to have technical implications. In other words, a methodology to support a theory X approach could afford to take a more authoritarian and thus simpler view of development projects.

On the whole, such behavioural views have been developed in isolation from the main stream of methodologies. ETHICS (Mumford - 1979) is a good example. The behavioural approach, not surprisingly, is explicitly Theory Y. The approach of the traditional methodologies, with their emphasis on rationality, is implicitly Theory X, though many authors would be upset to be told that. It is important that the two different cultural backgrounds should be amalgamated, and that the technical implications of management styles should be taken into account.

## 15. Artificial intelligence

One way of classifying problems is according to whether their solutions are more or less "programmable". Put at its weakest, the objective of artificial intelligence (AI) is to discover ways of programming which can be used for problems to which the solutions have been regarded as less programmable. Decisions made in organisations, particularly management decisions, range all the way from the highly programmable to the highly non-programmable. Since programmable decision making in organisations has commonly been delegated to computers, there is a natural interest in ways of shifting computer

capability further towards the non-programmable end of the scale. Recent developments in decision support systems are evidence of such an interest. The general objectives of AI are also clearly relevant, although in practice AI research is too often directed to problem domains which are so remote from those of management as to make its applicability obscure. Nevertheless, "expert systems" are a rather mundane spin-off from AI which has some promise of being useful in this respect. Opinions differ about the application domain of expert systems and whether they represent a radically new approach or merely a new style of implementation. It is undoubtedly the case that an expert system knowledge base is quite different from a conventional database, and that logic programming, if used, is quite different from conventional procedural programming. (It in fact represents one way of prototyping: see 8 above.)

Whatever the final judgement on expert systems may be, the objectives stated for them make expert systems techniques relevant both to information systems and to the system development process; and it can therefore can be expected that they will have an increasing impact on methodologies. Whether the much more ambitious work in mainstream AI will have the effect on information systems which it should, must remain to be seen.

## 4.4 CONCLUSION

While opinions will vary on the relative values of the fifteen viewpoints identified in this chapter, it would probably be accepted that none of them is without relevance to the task of developing a methodology. This diversity of relevant viewpoints should not come as a surprise: it is a measure of the richness and complexity of the activity of developing organisational information systems.

The diversity indicates two points. First, many methodologies, particularly the earlier ones, were based on a single or a very few viewpoints; to that extent their capability to assist was limited to a relatively a small part of the total system development effort. Second (the reverse of the same coin), a methodology which offers to assist in a major proportion of system development effort must be based on a rich amalgam of viewpoints or approaches.

# CHAPTER 5

# SURVEY OF TECHNIQUES

## CONTENTS

5.1 Introduction

5.2 Diagrammatic representations of flow or precedence

5.3 Non-diagrammatic process representations

5.4 Data representations

5.5 Conclusion

## 5.1 INTRODUCTION

A methodology involves an integrated approach to tackling a linked set of system development activities in relation to the system development process. A technique, on the other hand, may be regarded as offering a means of tackling a specific class of problems, of fairly limited scope, within the complete development process. In some cases, a technique may have been proposed as part of a methodology, but may nevertheless be usable on its own; most techniques are methodology independent.

This chapter presents a review of classes of techniques which have been found useful in addressing problems arising during system development. Individual techniques are discussed briefly but not in detail; many of them are too well-known to justify detailed description, and references are available for others.

## 5.2 DIAGRAMMATIC REPRESENTATIONS OF FLOW OR PRECEDENCE

The common characteristic of graphic representations in this class is that the concept of sequence is involved, although that is not immediately obvious in every case. There is a wide variety of such representations, and the differences between them are more apparent than real. They include the following, which are specifically considered in this section.

- Program flowcharts
- Program structure diagrams
- System run charts
- Data flow diagrams
- Precedence graphs
- Jackson structured diagrams
- Petri nets
- Decision trees

### Program flowcharts

Program flowcharts were the earliest form of graphic representations and have been most widely used. Despite the development of various sets of standards (e.g. ASME, ECMA, NCC) they are subject to considerable variation in their detailed use. The characteristic that all such flowcharts have in common is that nodes represent some action

(at a greater or lesser level of detail), that arcs represent time sequence or flow of control, and that the constructs of branching and iteration can be represented. Figure 5-1 shows an example. Additional references are CHAPIN (1970, 1981).

## Program structure diagrams

With the development of better designed programming languages and more disciplined approaches to programming, program flow charts have increasingly become regarded as unnecessary adjuncts to program source text. Nevertheless in all but the most trivial programs it remains necessary to represent the relationships between program components (e.g. modules, sub-routines, procedures, functions etc.). A program structure diagram provides such a representation. It is commonly in the form of a tree and, although it may not be immediately thought to show sequence, the fact that it represents a calling structure means that the sequence is implicit. Figure 5-2 shows an example: it is in fact a HIPO diagram.

## System run charts

These show the time or precedence relationships among a set of programs constituting all or a part of a system. The nodes represent program runs. In the minimal case the arcs represent not only time but also data files or messages passed from one program run to another. In other cases such files and external interfaces are shown by separate symbols, and the arcs linking them to program runs

Figure 5-1: example of a program flow chart.
Source: NCC (1971).

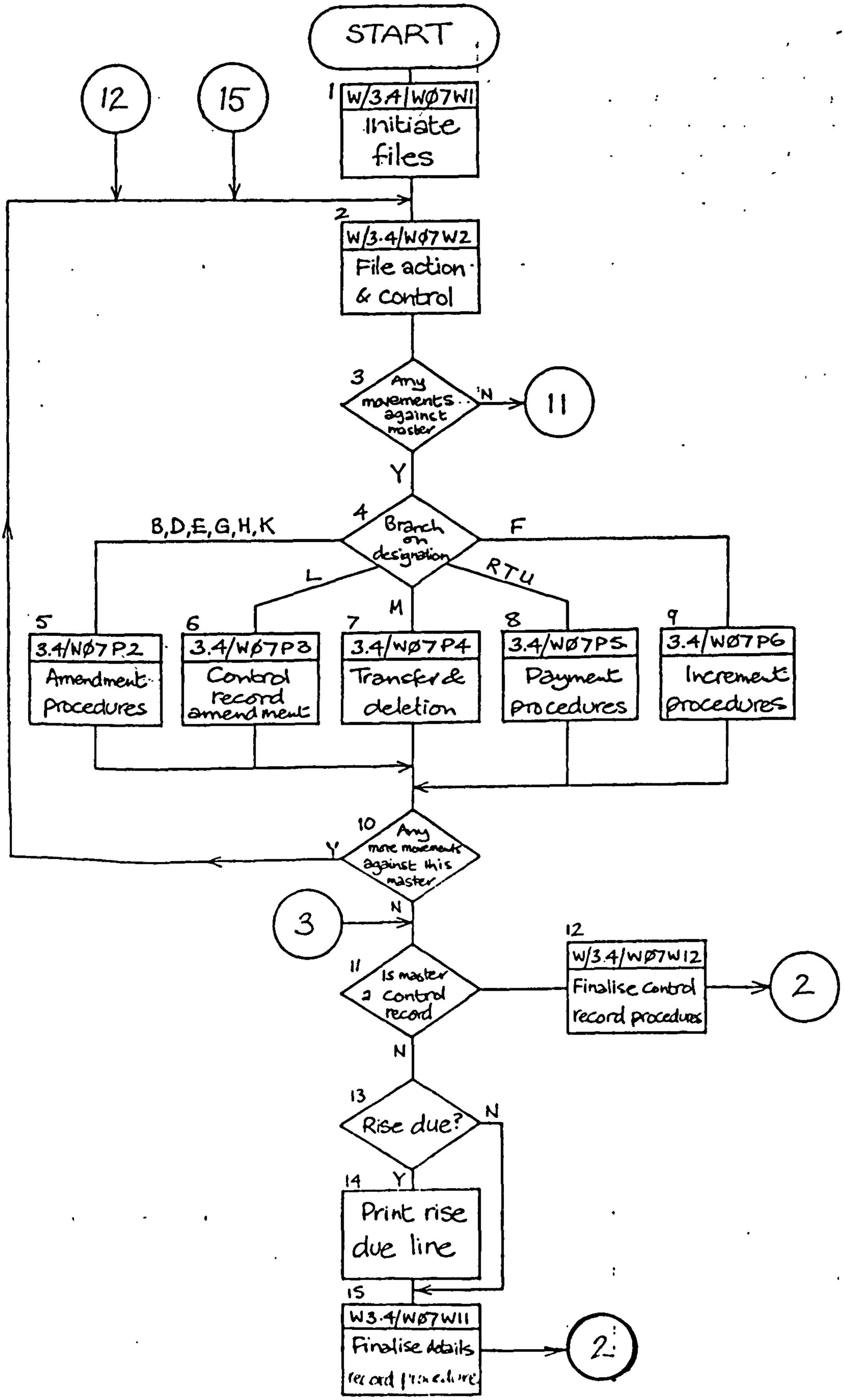
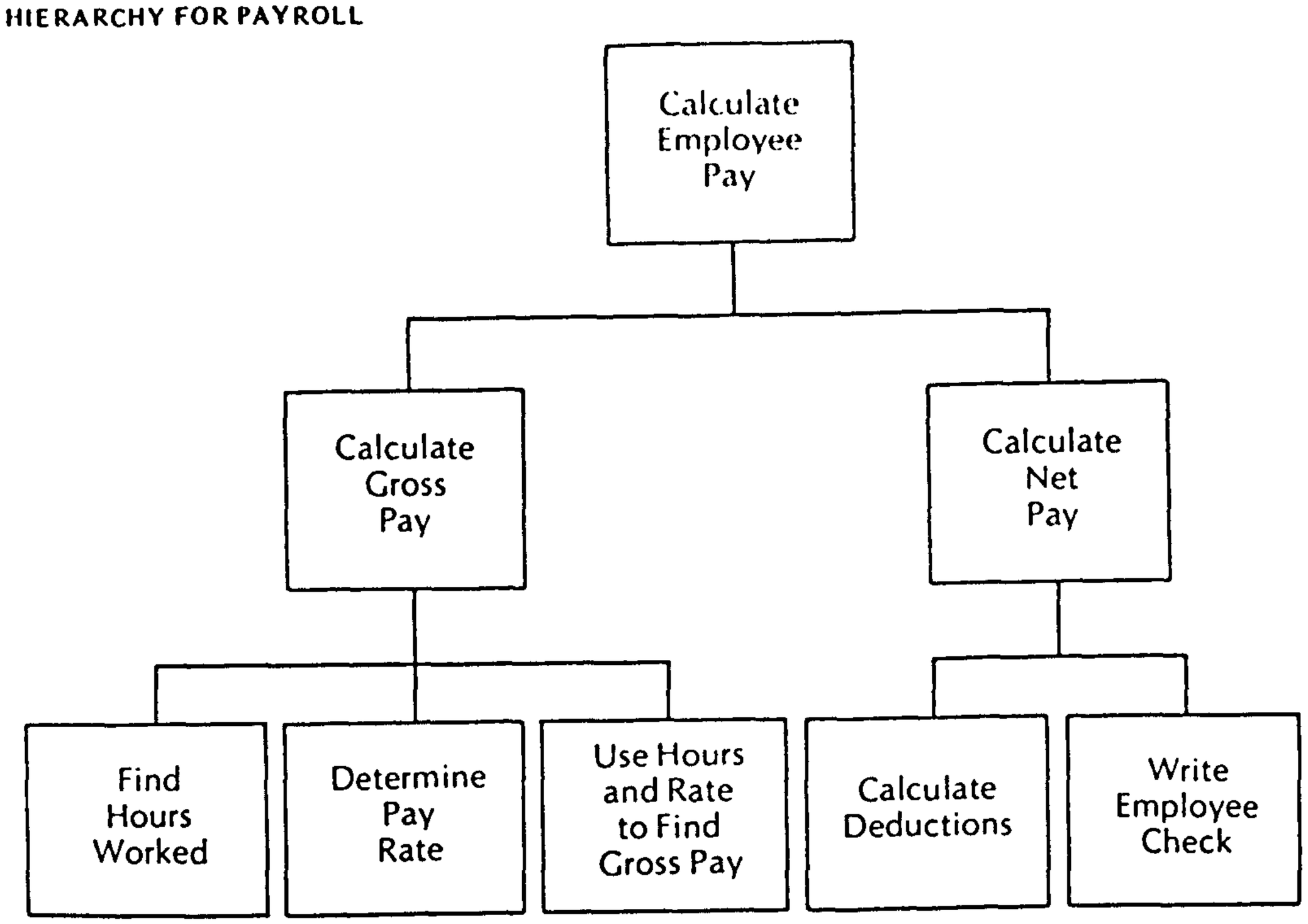


Figure 5.2: example of a program structured diagram.
Source: SASS C J (1979).



represent data movement. Figure 5-3 shows an example.

Large systems may comprise too many runs to be shown in a single diagram in which case (as with program flowcharts) there may be a hierarchy of diagrams in which, at the higher levels, the nodes represent subsystems, or groups of runs, rather than individual runs.

## Data flow diagrams

Data flow diagrams are particularly associated with the Structured Design group of methodologies: DEMARCO (1979), GANE and Sarson (1979), MYERS (1978) and YOURDON (1979). Figure 5-4(a) shows the meanings of the symbols employed. Data flow diagrams have the following characteristics.

- They are primarily intended for use at the so-called "logical design level", meaning that they represent processes and data both in the designed system and in its environment.

- They allow the representation of (groups of) human beings as sources/recipients of data.

- They allow the possible boundaries of designed systems to be shown - see figure 5-4(b).

- There is no single starting point (as there is in the types of chart described earlier in this chapter).

- In common with other forms of flow chart, process boxes can be decomposed on lower-level charts.

Salaries
Computer Run Chart

Figure 5.3: example of system run chart.
Source: NCC (1971).

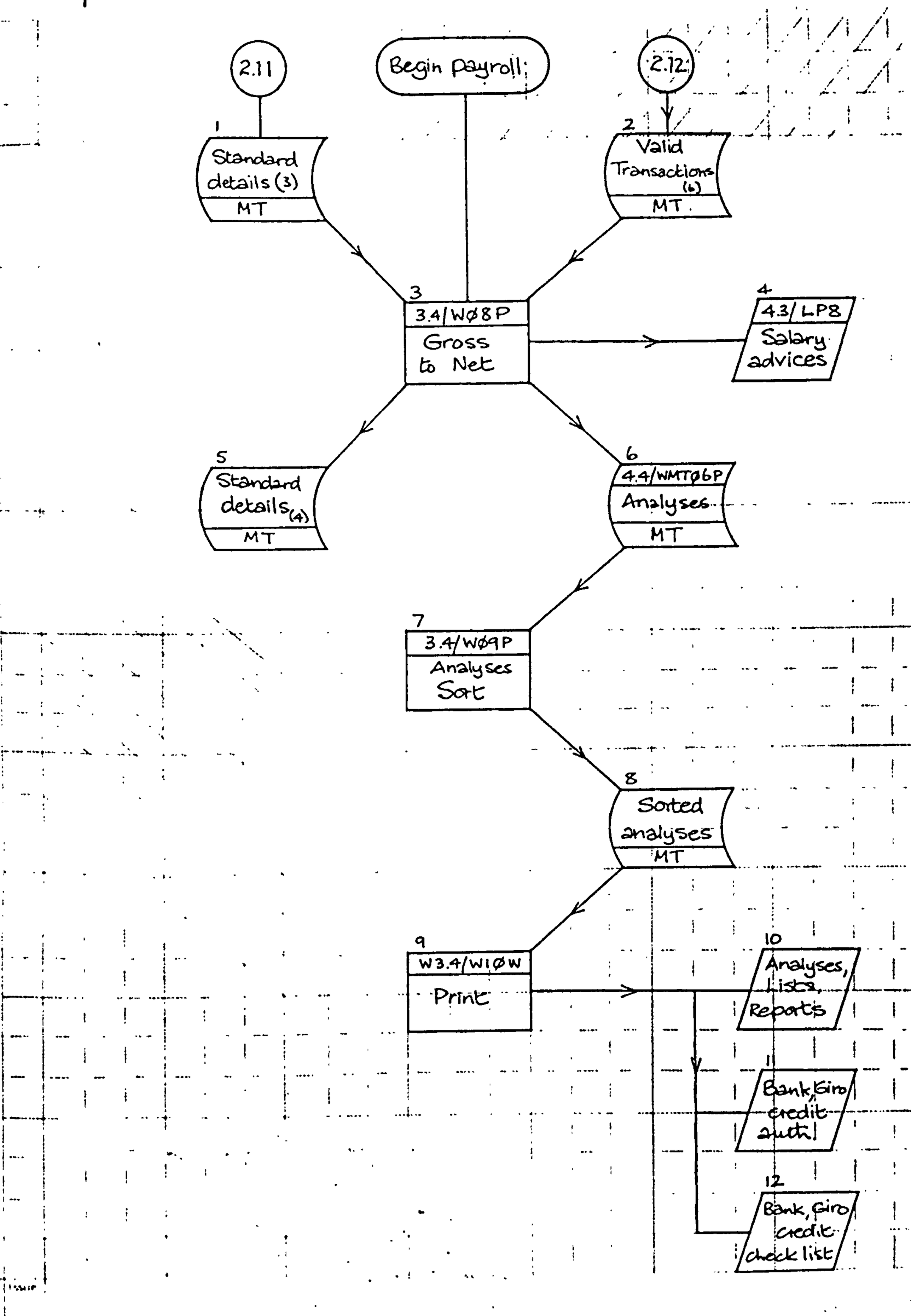


Figure 5.4: example of data flow diagram.
Source: GANE and Sarson (1979).

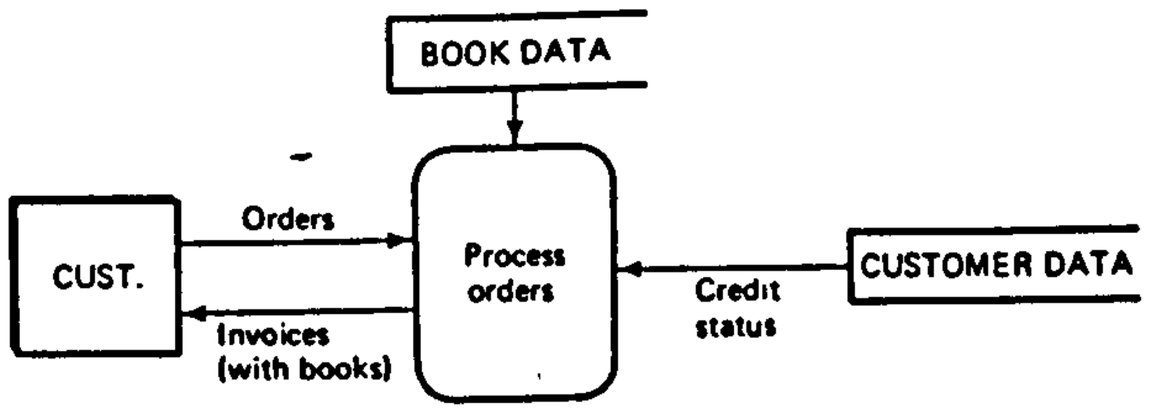
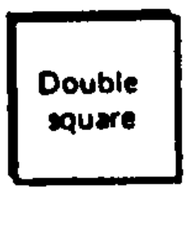



**Figure 2.1 Logical data flow diagram**



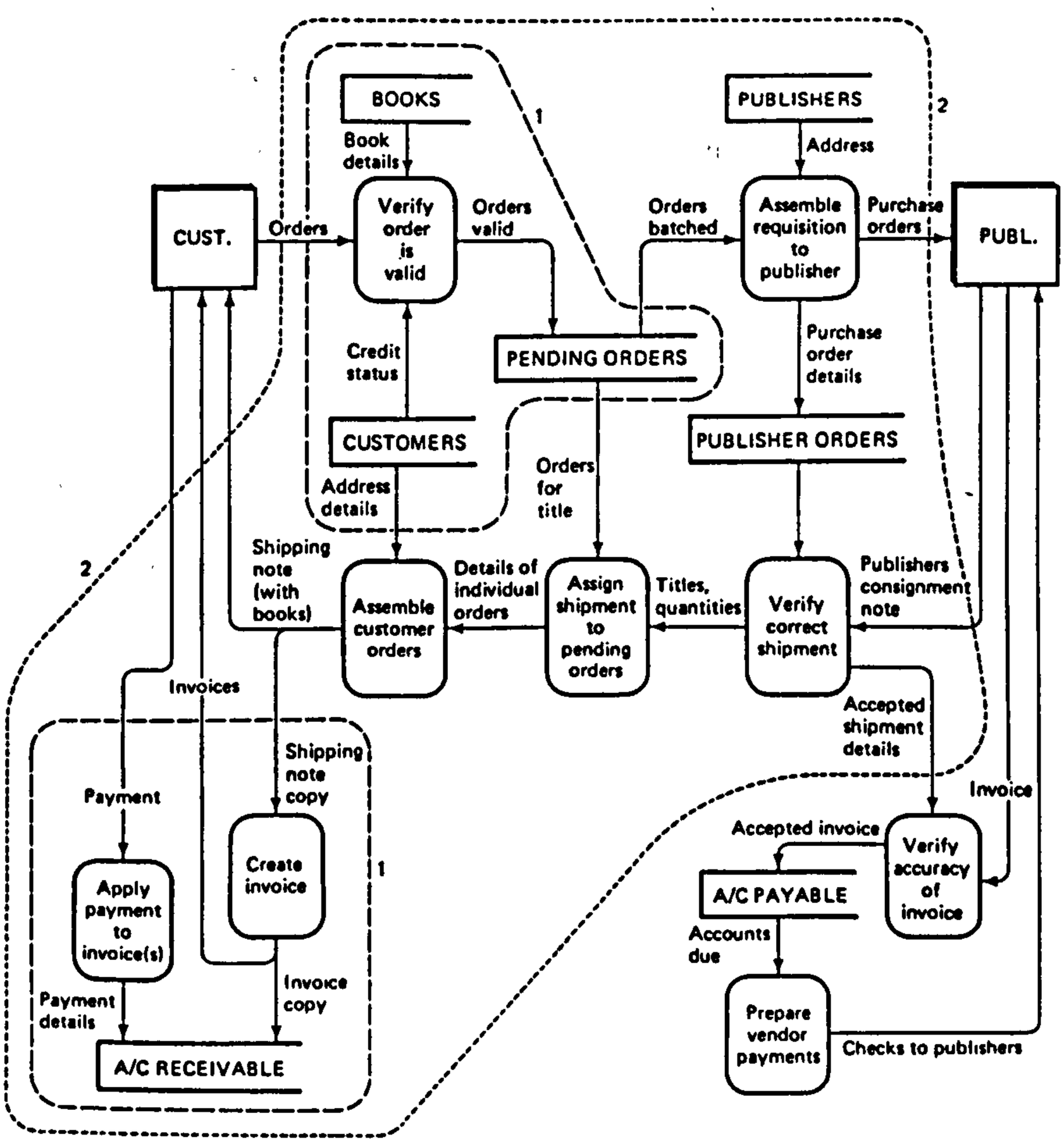

Source or destination of data


Flow of data


Process which transforms flows of data


Store of data

5-4(a)




5-4(b)

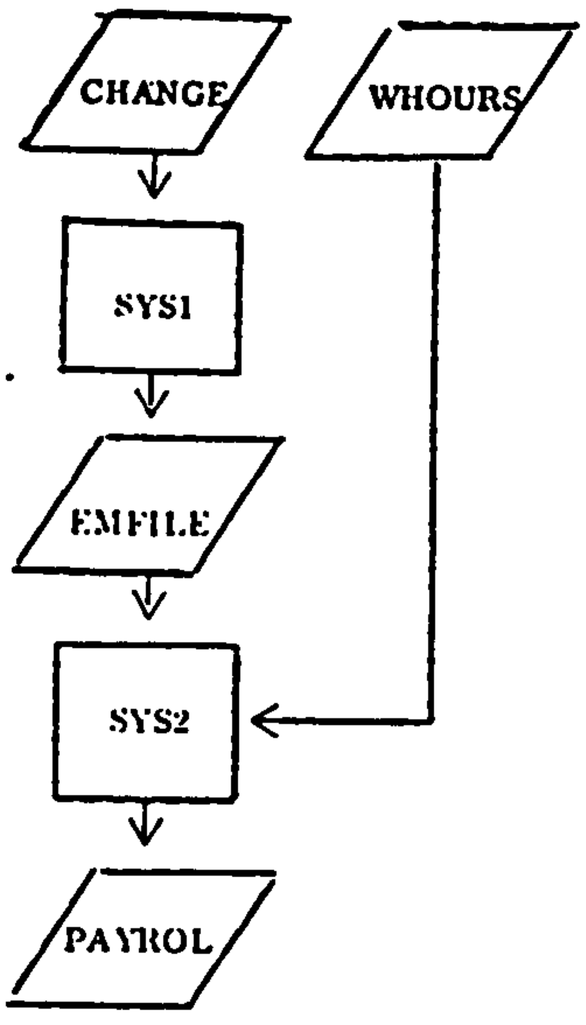
## Precedence graphs

The idea of a precedence graph was introduced by LANGEFORS (1973), but was not much developed by him. It was used more extensively in some of the Scandinavian methodologies based on Langefors's founding work; the example shown in figure 5-5 is from Solvberg's CASCADE project.

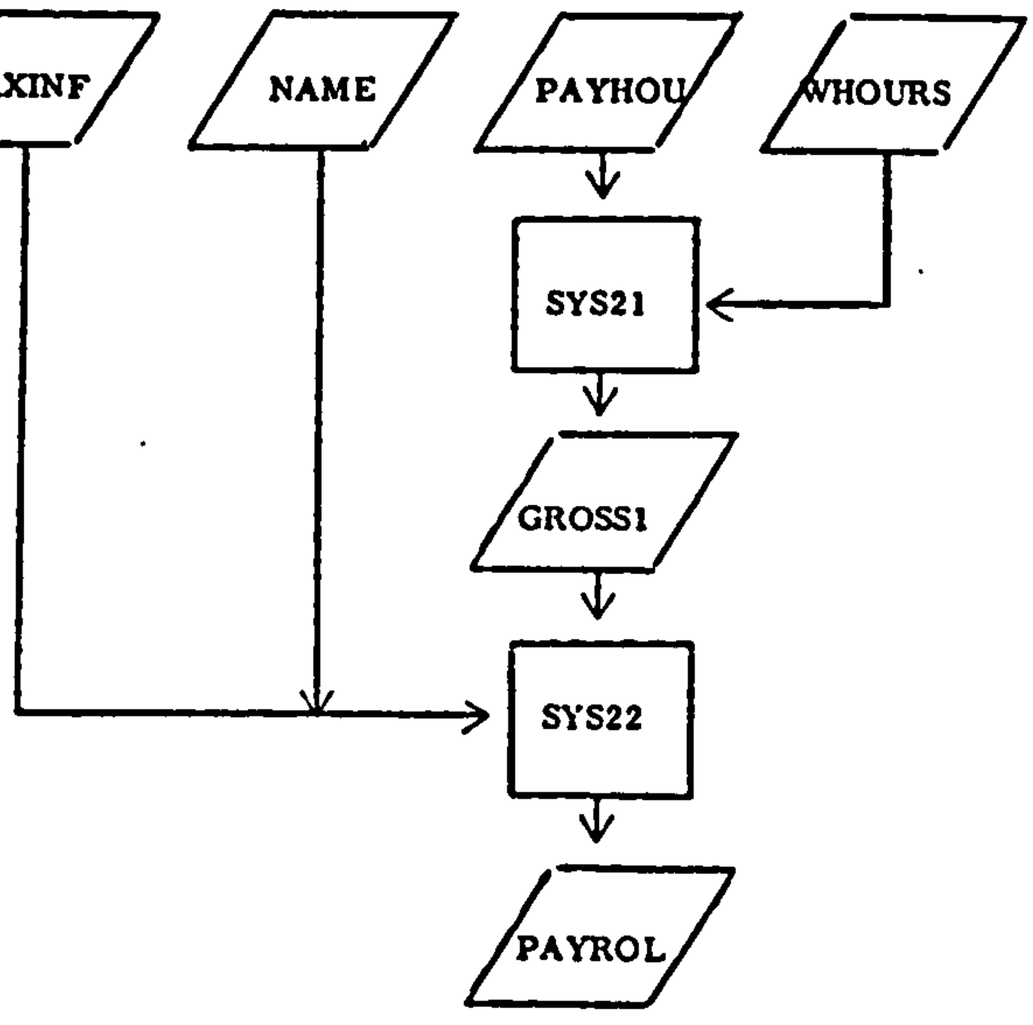
As can be seen from the example, a precedence graph can be used to show the decomposition of a system into subsystems and files (similarly to a system run chart), and subsequent decomposition down to the level of individual programs, which can be represented purely as precedence graphs of data elements. This last form is in a sense the opposite of a conventional program flowchart: whereas the program flowchart shows the sequence of operations, leaving the passing of data between them implicit, the data precedence graph shows the sequence of production of data element values, leaving the operations (functions) implicit. Of the two forms, the precedence graph is more concise and satisfactory: it is implementation - independent, and each implicit function is specified in terms of its arguments.

Precedence graphs can be alternatively represented as precedence matrices. Langefors is far more concerned with the matrix representation and with operations that can be performed using it. The matrix form is most suitable as an internal (database) representation of precedence relationships.

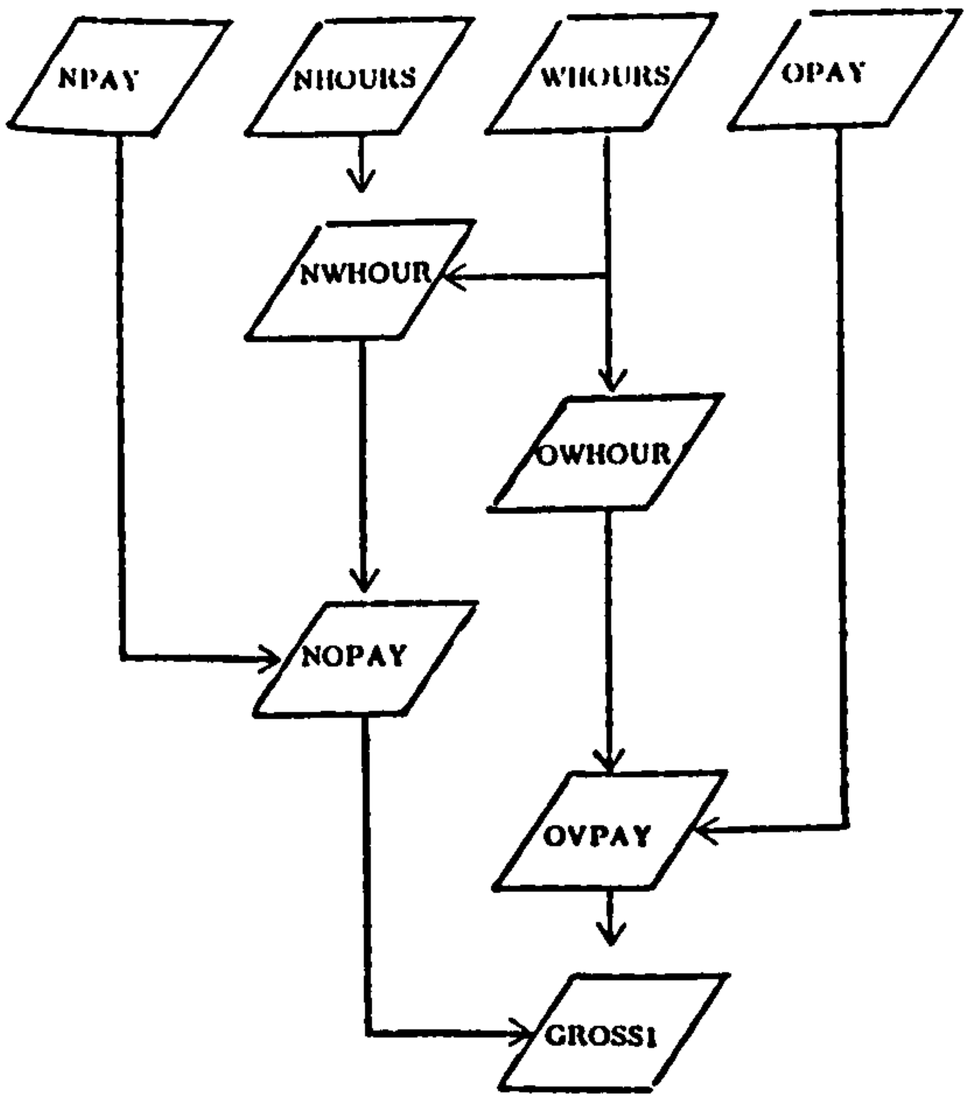
Figure 5.5: example of precedence graph.
Source: BUBENKO, Langefors, Solvberg (Edtrs) (1971).




The PAYROL-system




SYS2 is detailed, EMFILE is decomposed




Precedence graph of SYS21, with PAYHOU decomposed

| IDENTIFIER | NAME |
|---|---|
| CHANGE | CHANGES IN EMPLOYEE-FILE INF. |
| EMFILE | EMPLOYEE FILE INFORMATION |
| GROSS1 | GROSSPAY CALC |
| GROSS2 | GROSS PAY |
| NAME | EMPLOYEE NAME |
| NAME2 | EMPLOYEE NAME (OUT) |
| NETPAY | NET PAY |
| NHOURS | NORMAL WORKING HOURS PR WEEK |
| NOPAY | NORMAL PAY |
| NPAY | NORMAL HOURLY PAY |
| NWHOUR | NORMAL WORKING HOURS |
| OPAY | OVERTIME HOURLY PAY |
| OVPAY | OVERTIME PAY |
| OWHOUP | OVERTIME HOURS WORKED |
| PAYHOU | INFO. ON PAY AND NORMAL WORK. |
| PAYROL | PAYROLL |
| TAXDED | TAX DEDUCTED |
| TAXINF | TAX INFORMATION |
| TAXPAY | TAX PAID |
| WHOURS | REPORT ON HOURS WORKED PR.WEEK |

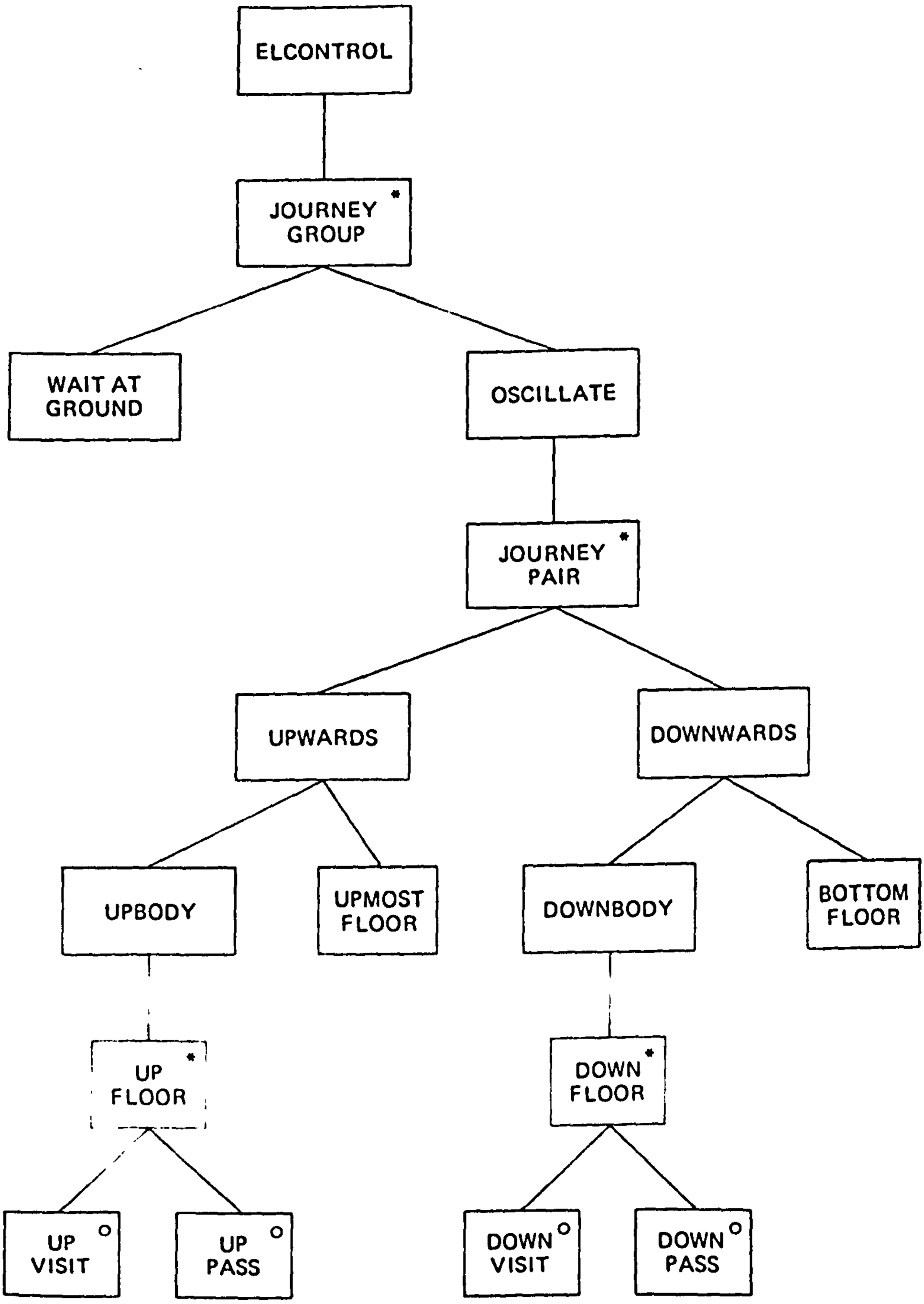
IDENTIFIERS USED IN THE GRAPHS WITH CORRESPONDING LONG NAMES.

## Jackson structured diagrams

Jackson is as important an influence in the user community in the UK as the Yourdon school is in the USA. An important difference at the programming level is that Yourdon emphasises functional decomposition whereas Jackson emphasises the derivation of program structures from data structures. JSP diagrams allow the representation, for both data streams and programs, of the structures of sequence, branching and iteration. Unlike program flowcharts, these diagrams are hierarchical in form. As with the more generalised program structure charts, however, control flow can be followed by traversing the tree in the appropriate order.

Entity structure diagrams in JSD are of the same form but are concerned with the structure, and therefore implicitly the sequence, of events and actions generated by or happening to real world entities. An example is shown in figure 5-6. It is important to note that these diagrams cannot handle parallelism in real world events. The inadequate justification is offered (1) that events are the source of data to be handled by the programs, (2) that event models must be isomorphic with data and program models, (3) that few current programming languages support parallelism, and therefore (4) that it is not necessary for an event model to do so. These diagrams are worth including in this survey, however, because (together with JSP diagrams) they are of a form which has become fairly well known and influential. The same cannot be said of system specification diagrams in JSD, which are highly specific to that particular methodology.

Figure 5.6: example of Jackson structured diagram.
Source: JACKSON (1983).



## Petri nets

A Petri net is an abstract formal model of information flow. As described in PETERSON (1977), the theory of Petri nets has developed from the work of Carl Adam Petri, A W Holt, Jack Dennis and others.

The structure of a Petri net is formally defined as a four-tuple, c = (P, T, I, O), where P stands for process, T stands for transition, I stands for input function and O stands for output function. The components of the above structure may be defined as follows.

P = {pi} ..... (* shows the set of processes *)

T = {tj} ..... (* shows transitions or mappings *)

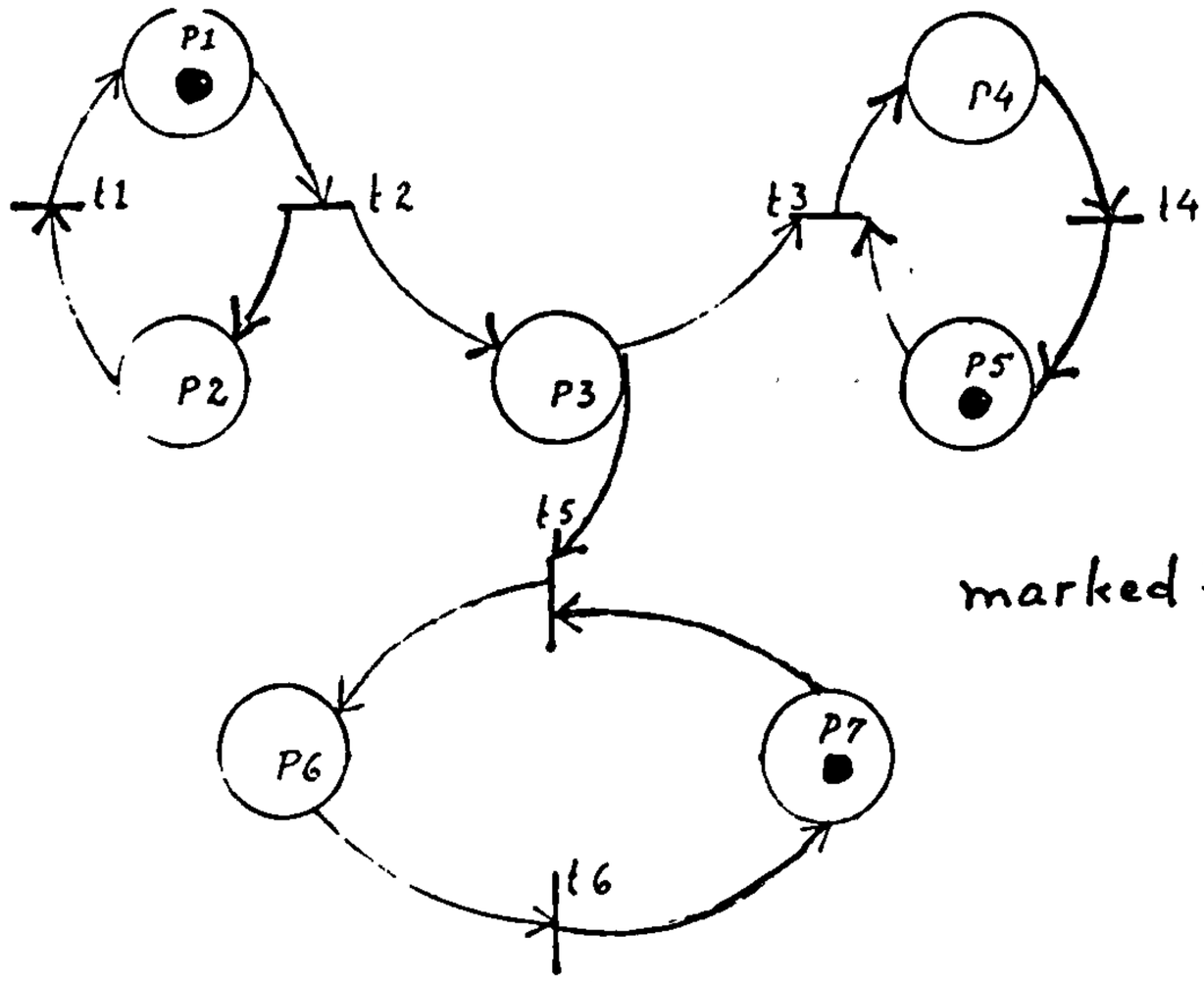
I = {pi, tj} ..... (* shows that input of transition tj is pi *)

O = {pi, tj} ..... (* shows that output of transition tj is pi *)

In a Petri net graph there are two types of nodes corresponding to (a) places and (b) transitions. A circle represents a place, and a bar represents a transition. The input and output functions are represented by directed arcs from a place to a transition and vice versa. Figure 5-7(a) shows the Petri net graph corresponding to the formal structure defined above.

A Petri net in addition to its static properties has dynamic properties that result from its execution. The execution of a Petri net is controlled by movement markers (called tokens), which are

Figure 5-7 a, b, c : examples of Petri-net.
Source: PETERSON (1977).




marked Petri net figure 5-7(a)

a marked petri-net : figure 5-7(b)

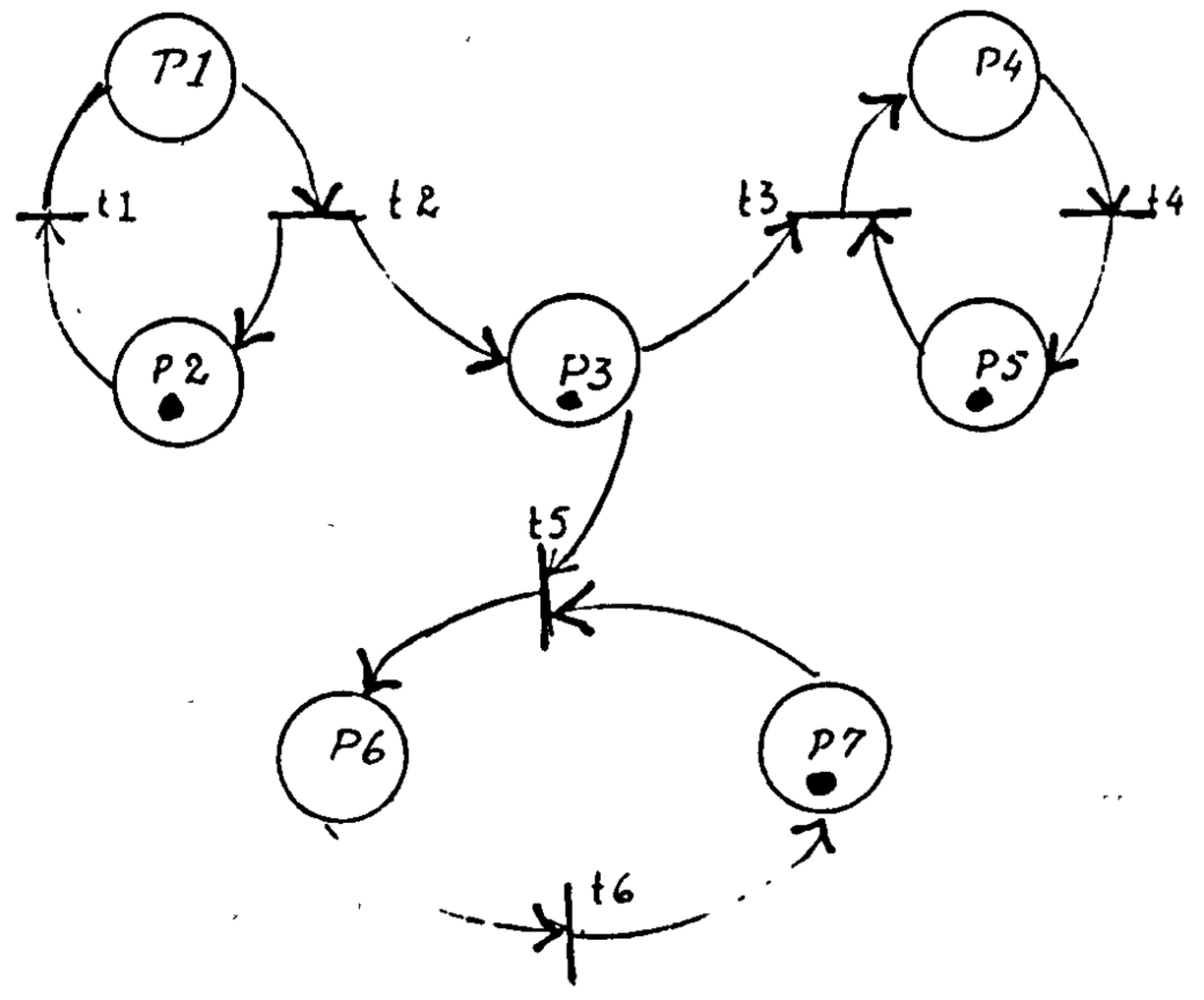
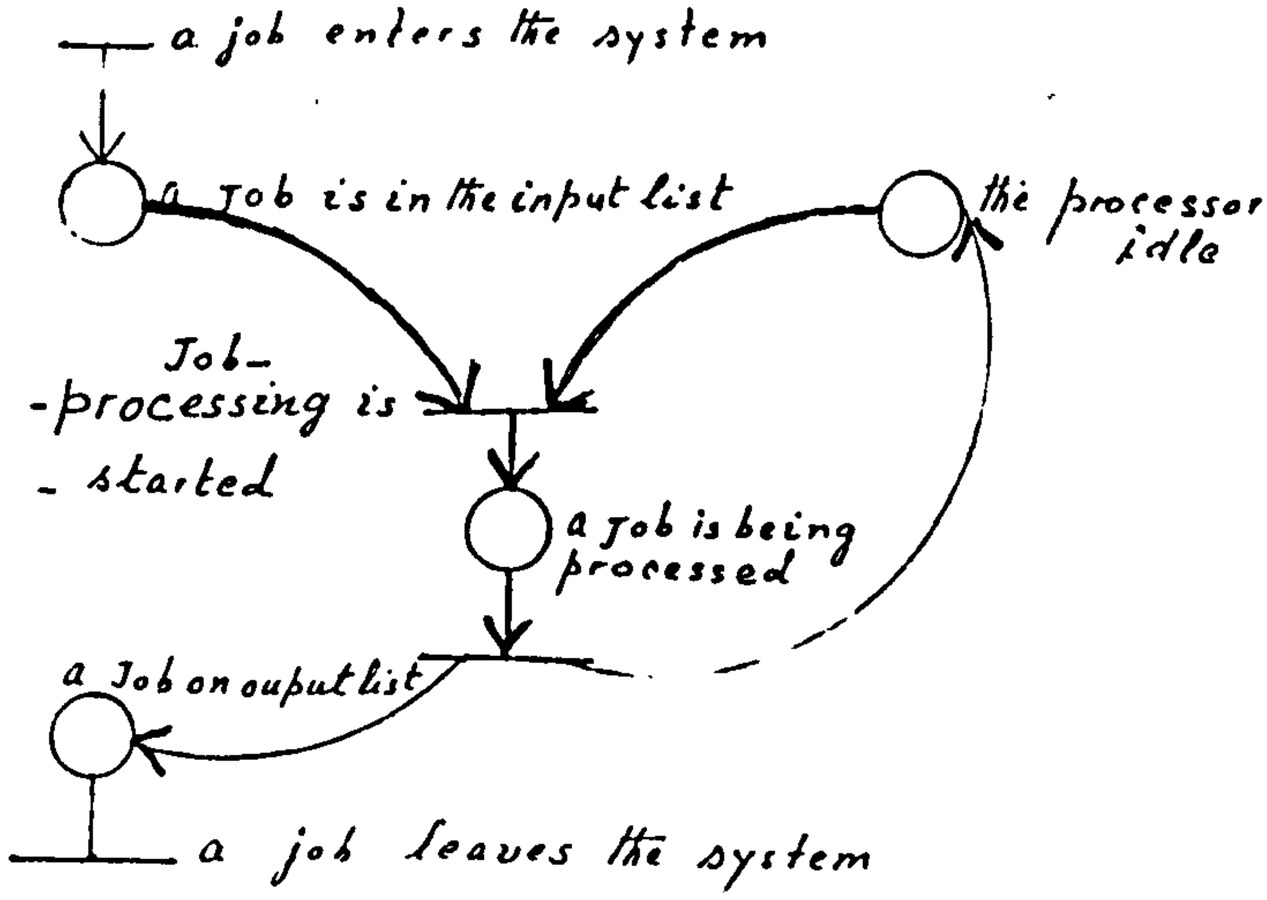



figure 5-7(c)



represented by dots residing within the circles.

A Petri net with tokens is called a 'marked Petri net'. Tokens are moved by the firing of the transitions of the net. A transition is called 'enabled' when all of its input places have tokens in them, and only enabled tokens can be fired. The transition fires by removing the enabling tokens from their input places and generating new tokens which are stored in the output places of the transition.

Figures 5-7(a) and 5-7(b) show the dynamic properties of a Petri net. Both the figures represent marked Petri nets. In figure 5-7(a) transition (t2) is enabled since it has a token in its input place (p1), while (t5) is not enabled since one of its inputs (p3) does not have a token.

If (t2) fires, the marked Petri net of figure 5-7(b) results. The firing of (t2) in figure 5-7(a) removes the enabling token in (p1) and generates tokens in (p2) and (p3).

The distribution of tokens in a marked Petri net defines the state of the net, and is called its 'marking'. In different markings, different transitions may be enabled. In figure 5-7 (b), three transitions (t1), (t3) and (t5) are enabled, none of which were enabled in figure 5-7(a).

Petri nets were devised for use in the modelling of specific classes of problems, such as discrete-event systems with concurrent or parallel events, systems of distributed control with multiple processes occurring concurrently, and systems in which events occur

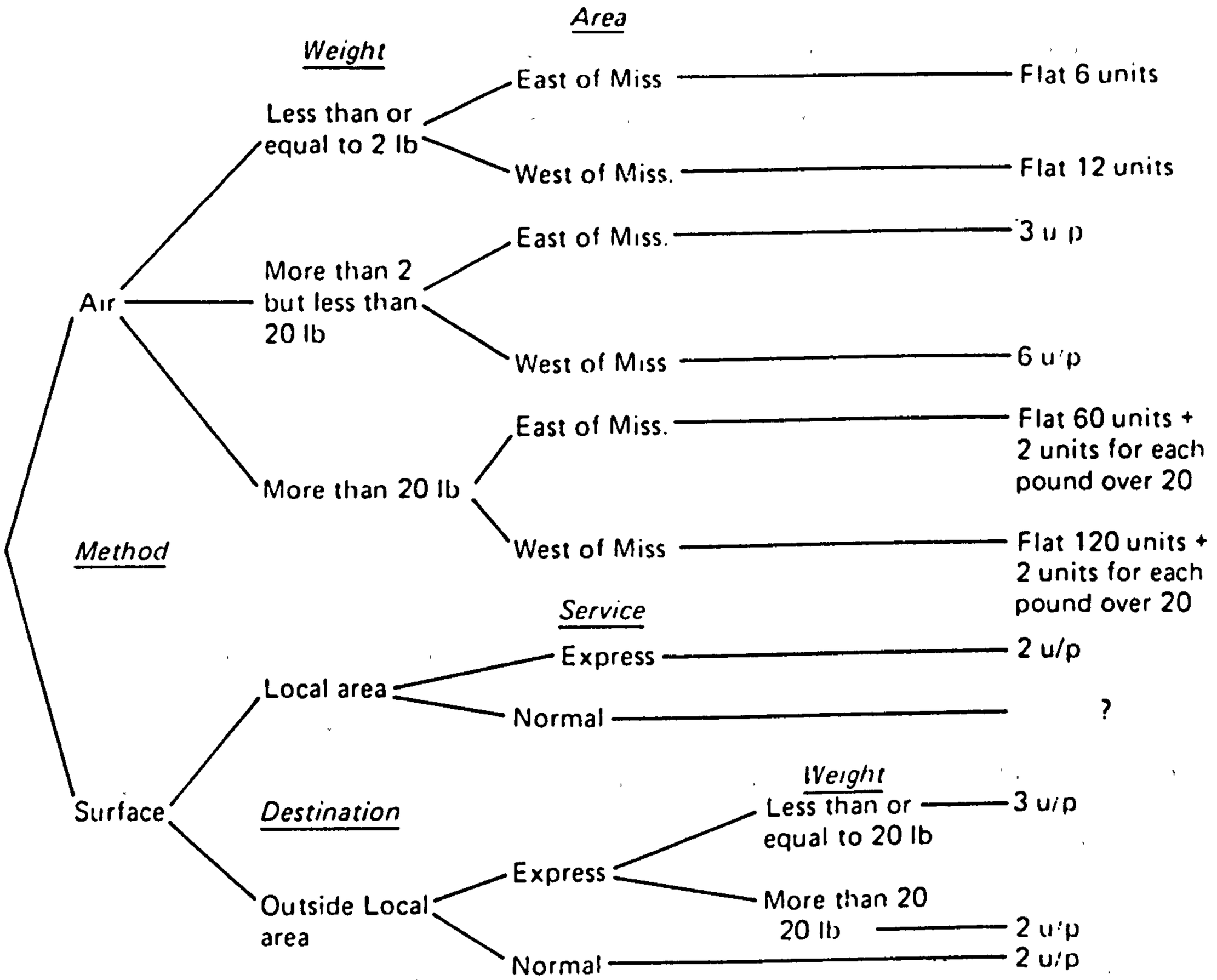
asynchronously and independently.

They are used to take the description of the system and analyse it for the presence of desirable and undesirable properties. Figure 5-7(c) shows the modelling of a computer system.

## Decision trees

Decision trees are used to represent complex decision structures where the final design is reached by a process of successive partitioning of the solution space. They are traditionally laid out horizontally. Each path from the root to a leaf node represents an ordered sequence of condition evaluations, or branches in programming terms. Figure 5-8 shows an example. Additional references are DEMARCO (1979), YOURDON (1979) and MYERS (1978).

Figure 5.8: example of decision tree.
Source: GANE and Sarson (1979).




Complete decision tree

## 5.3 NON-DIAGRAMMATIC PROCESS REPRESENTATIONS

By far the major emphasis of the diagrammatic techniques in the previous section was on activities or processes, although some of the techniques provided for the representation of data. In this section we look at further techniques, of a non-diagrammatic nature, for representing processes. They comprise the following.

- Decision tables
- Pseudo code
- Techniques based on formal logic

### Decision tables

Like decision trees, decision tables are used to represent the relationships between a complex set of conditions and a set of outcomes. Whereas decision trees can only be used where there is a partial ordering of decisions into a tree structure, there is no such constraint in the case of decision tables. Each rule is to be regarded as an independent statement mapping from a particular set of conditions to a particular set of actions. Although the conditions must necessarily be set out in some order, that ordering is not regarded as significant in the evaluation of conditions; nor is the ordering among rules significant.

Apart from the major distinction between limited entry and extended entry decision tables, there is a considerable variety of detailed

rules for their construction. In particular it is necessary to observe certain conventions if decision tables are to be checked for non-ambiguity and completeness, whether manually or by machine.

Decision tables may be used equally to record decision processes in the real world or those to be carried out in a computer program. In the latter case software tools may be available to convert decision tables into source code modules. An example is shown in figure 5-9. Additional references are KING P J H (1966, 1967b), POLLACK (1974), GANE and Sarson (1979), DEMARCO (1979) and FERGUS (1977).

## Pseudo code

Pseudo code, of which many detailed variants have been proposed, is an abstraction of certain features common to many programming languages - ie. the standard control structures of sequence, branching and iteration. Pseudo code defines the way in which these constructs are recorded; beyond that, there are few if any other rules and there is freedom in the naming and description of data and processes. Pseudo code thus stands between natural language and compilable programming languages. The structured English of the Yourdon school may be regarded as a major variant of pseudo code, omitting some of the program-oriented detail.

Like decision tables, these approaches can be used earlier or later in the system development process, offering a semi-formal means of recording either human activities or tasks to be carried out by a program. In the latter case they form a class of program

Figure 5.9: example of decision table.
Source: NCC (1971).

| C = 5<br>A = 12<br>R = 16 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 MR5F grade= * working hours | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | N | N | N |
| 2 code = 8 | Y | Y | Y | Y | Y | Y | N | N | N | N | N | N | Y | Y | N | N |
| 3 MR21 grade = 0 | Y | Y | Y | N | N | N | Y | Y | Y | N | N | N | Y | N | Y | N |
| 4 Basic/merit = 4 | Y | - | N | Y | - | N | Y | - | N | Y | - | N | - | - | - | - |
| 5 Basic/merit = 2 | - | Y | N | - | Y | N | - | Y | N | - | Y | N | - | - | - | - |
| 1 Insert grade and operator type | | | | | | | | | | | | | X | X | X | X |
| 2 Insert next review date | X | | | X | | | X | | | X | | | X | X | X | X |
| 3 Apply formula 1 | | | | | X | X | | | | | | | | X | | |
| 4 Apply formula 2 | | | | | | | | | | | X | X | | | | X |
| 5 Get input amount | | X | X | | | | | X | X | | | | X | | X | |
| 6 Get result of formula | | | | | X | X | | | | | X | X | | X | | X |
| 7 Negate | | | X | | | X | | | X | | | X | | | | |
| 8 Put in basic rate | | | | | | | | | | | | | X | X | X | X |
| 9 Put in addit.rate | | X | X | | X | X | | X | X | | X | X | | | | |
| 10 Print sterling amendment | | X | X | | X | X | | X | X | | X | X | X | X | X | X |
| 11 Print name amendment | | | | | | | | | | | | | X | X | X | X |
| 12 Print date amendment | X | | | X | | | X | | | X | | | X | X | X | X |

* Formulae are applied to the amount in the transaction file record (4.7/MR5F)

Formula 1 $\quad \text{Result} = \frac{\text{amount}}{13} \times \frac{\text{standard working hours}}{36.25}$

Formula 2 $\quad \text{Result} = \frac{\text{amount}}{13} \times \frac{\text{standard working hours}}{40}$

specification and design languages. They are of more general application than decision tables, since they are not confined to representing the evaluation of a set of conditions.

Strict pseudo code (not structured English) can be employed in the stepwise refinement method of program development. In some cases software tools are available which will carry out a transformation from pseudo code to source code skeleton in a given language. Examples of pseudo code and structured English are shown in figure 5-10.

## Techniques based on formal logic

The concepts and notations of formal logic are used in a number of contexts to provide a non-procedural means of describing rules and processes. In addition to its use in some of the methodologies in appendix-B (e.g. DADES, IML-inscribed nets), formal logic is the basis of Prolog and other logic programming languages, IPL (interpreted predicate logic - a proposal for specifying constraints in databases), the relational calculus (for specifying database queries), Legol (a system for recording and simulating the effects of a complex set of regulations), and formal program specification methods. Although formal logic approaches are non-procedural, there is a mapping from formal logic constructs to conventional programming language constructs. For instance implication maps if..then; and the similarity between Prolog rules and decision table rules has been observed.

Figure 5.10: example of pseudo code and structured English
Source: GANE and Sarson (1979).

```
GENERATE INVOICE
      DO COMPUTE-INVOICE-TOTAL
      DO COMPUTE DISCOUNT
      DO COMPUTE-SHIPPING-HANDLING
      Subtract discount from invoice-total to get invoice-net
      Add shipping-handling-fee to invoice-net to get total-payable
      Write invoice.

COMPUTE-INVOICE-TOTAL
      REPEAT EXTEND-ITEM-LINE UNTIL all item-lines have been extendedded
      Add all item-line-totals to get invoice-total

EXTEND-ITEM-LINE
      Multiply quantity by unit-cost to get item-line-total.

COMPUTE-DISCOUNT
      IF          invoice-total is GE $1000
                  discount is 5% of invoice-total
      ELSE IF     invoice-total is GE $250 but LE $1000
                  discount is 2½% of invoice-total
      ELSE IF     invoice-total is GE $100 but LE $250
                  discount is 1% of invoice-total
      ELSE        (invoice-total is LT $100)
            SO    discount is nil

COMPUTE-SHIPPING-HANDLING
      IF    order specified air shipment
        THEN DO COMPUTE-AIR-FREIGHT
      ELSE (order specifies surface shipment or method is open)
            SO DO COMPUTE-SURFACE-FREIGHT
      Multiply rate by current-unit-value to get shipping-handling-fee

COMPUTE-AIR-FREIGHT
      IF          weight is LE 2
                  rate is 6 units
      ELSE IF     weight is GT 2 but LE 20
                  Multiply each pound of weight by 3 units to get rate
      ELSE        (weight is GT 20)
            SO    Subtract 20 from weight to get excess
                  Multiply excess by 2 units per pound and add 60
                    (20 pounds at 3 units per pound) to get rate

COMPUTE-SURFACE-FREIGHT
      IF    destination is local
        and-IF  service-code is express
              THEN Multiply each pound of weight by 2 units to get rate
      . . .
      . . .
      and so on
```

5-10 (a) Structured English

```
Initialize the program (open files, set counters)
Read the first order-record
DO-WHILE there are more order-records
      DO-WHILE there are more items on the order

            Compute item-total
            Add item-total to invoice-total

      END-DO

      Compute discount
      Compute shipping and handling fee
      Compute invoice-net, total-payable
      Print invoice
      Write invoice to accounts-receivable file
      Add invoice-detail to summary counters
      Read next order record

END-DO

Print summary of day's invoices
Terminate program
```

5-10 (b) Top-level pseudocode

Formal logic tends to be a far more concise form of representation than a procedural programming language. Software tools can be defined to "animate" sets of formal logic statements: they are slow in execution but nevertheless can be valuable for prototyping purposes. An example of formal logic, as used in IPL, is shown in figure 5-11.

41. (Garages = {G \ For some d (G Begins to trade on d)}).

42. For all G For all d (If G Begins to trade on d Then d Is among Days).

43. For all G For all d
(If G Ceases to trade on d
Then G Is among Garages & d Is among Days).

44. (Garages In I).

45. (Cnv Denotations N (Garages X Strings) Is among Fcn).

46. For all G For all d
(If G Ceases to trade on d
Then For some d´
(G Begins to trade on d´
& d´ Is among Before;{d})).

Figure 5.11: example of formal logic.
Source: ISO (1982).

## 5.4 DATA REPRESENTATIONS

This section includes techniques, both diagrammatic and non-diagrammatic, for representing the relationships between data items or data structures. Just as the techniques in sections 5.2 and 5.3 cover between them both real world activities and machine processes, so the techniques in this section cover both real world entities and the data items which represent their properties within the computer. The following are included.

- Relational schemas
- Conceptual schemas
- Bachman diagrams
- Identification matrices
- Data abstraction
- Data dictionaries

### Relational schemas

A relational schema permits the declaration of one or more relation-types, where a relation-type declaration defines a set of associated data types (or attributes). In any relation there must be one or more key attributes and zero or more non-key attributes. Relations must be declared in such a way that non-key attributes in a relation are functionally dependent on the key-attribute(s), and that

duplicate values of the key at any moment of time are impossible.

Key attributes are most commonly thought of as identifying real world entities or events; they may also, though less frequently, identify abstract properties. A multiple-key relation defines the relationship(s) between the entities, events or properties represented by each of the elements of the key and any properties of these relationships. A foreign key is said to exist where a non-key attribute in one relation is a key attribute in another.

The keys and foreign keys that exist in a relational schema imply a network of real world relationships; but the relational model provides no mechanism for displaying that network of relationships explicitly.

The relational model provides the basis for a rich field of theoretical studies on data semantics. It is also used practically as a sound starting point for record design, and a number of DBMSs have been implemented on the basis of this model. They have the advantage of relative simplicity of schema definition, and of concise and powerful query facilities, but the disadvantage of relative inefficiency in execution. An example of a relation is shown in figure 5-12. Additional references are DOBOSZ (1981), GLAGOWSKI (1978), HUTT (1979), KENT (1983), MACLEOD (1981), RONALD (1982) etc.

## Conceptual schemas

A conceptual schema is a way of representing all or most of the data in a system at a relatively high level of abstraction - ie. without

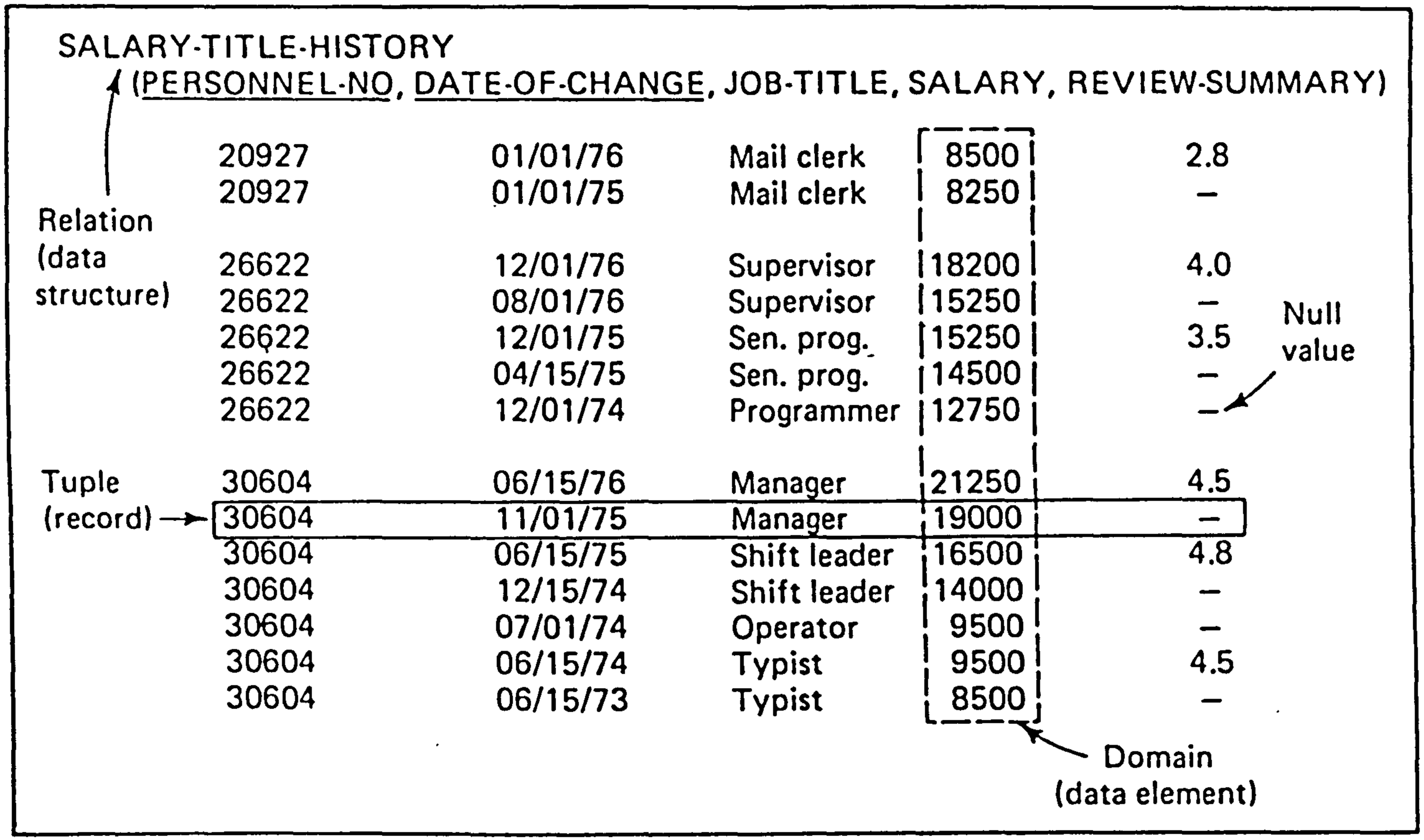



Figure 5.12: example of a relation.
Source: GANE and Sarson (1979).

consideration of syntactic or physical representations and in relationship to the real world entities etc. from which the data derives. These approaches have usually been strongly influenced by relational theory, while not limited by the constraints of that theory. Two of the best known models are the entity-relationship model and the binary relationship model. Using different conventions both models permit the representation of both real world and data objects, the relationships between them, and the attributes of both objects and relationships. In each case the representation may be either graphic or textual. Graphic representations, for systems of any size, become extremely large, and difficult to draw and to read. Textual representations consist of many individual statements, any one of which may be easy to read but which are difficult to grasp as a totality and need the support of software tools for their effective use by the developer.

One has the impression that, whereas the simpler relational model is employed in the user community for primitive conceptual modelling, these more sophisticated conceptual schemas are still confined to the research community. An example of a conceptual schema using the entity-relationship model is shown in figure 5-13 (diagram form) and 5-14 (textual form).

Bachman diagrams

These are closely associated with network database models, which are in extensive practical use. They permit the representation of record

Figure 5.13: example of conceptual schema (diagram form).
Source: ISO (1982).

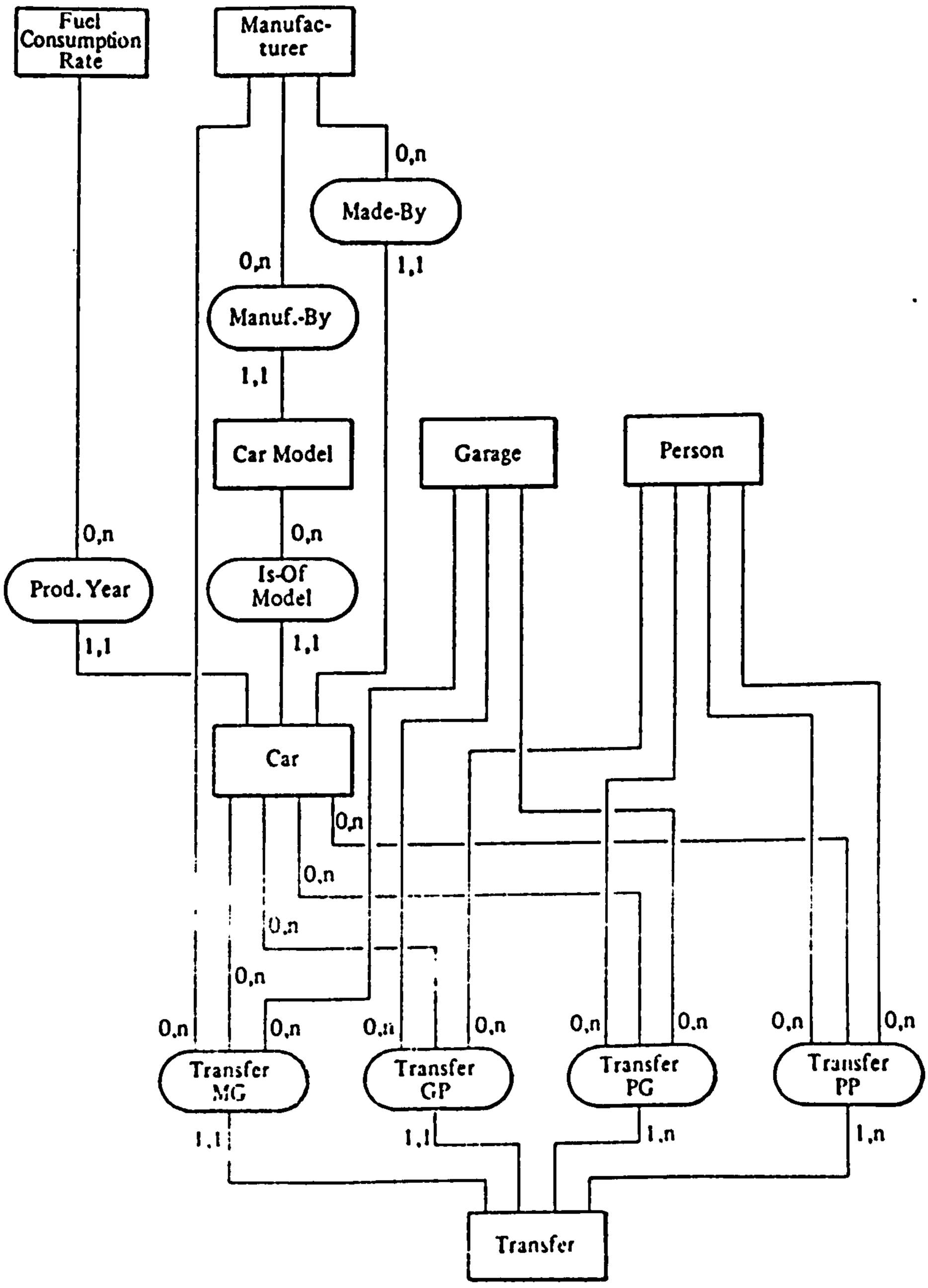


Figure 5.14: example of conceptual schema (textual form).
Source: ISO (1982).

A description of the conceptual schema in the language of the grammar defined in section D.3, is as follows:

```
CONCEPTUAL SCHEMA    car-registration

ENTITY-TYPE          manufacturer
IDENTIFIER           manuf-id
DESCRIPTION          manuf-id
                     is-operating

ENTITY-TYPE          car-model
IDENTIFIER           model-id
DESCRIPTION          model-id
                     fuel-cons-spec

ENTITY-TYPE          car
IDENTIFIER           reg-no
DESCRIPTION          reg-no
                     serial-no
                     destroyed-date

ENTITY-TYPE          fuel-consumption-rate
IDENTIFIER           year-id
DESCRIPTION          year-id
                     max-cons

ENTITY-TYPE          garage
IDENTIFIER           garage-id
DESCRIPTION          garage-id
                     is-trading

ENTITY-TYPE          person
IDENTIFIER           person-id
DESCRIPTION          person-id

ENTITY-TYPE          transfer
IDENTIFIER           transfer-car, transfer-date, seq-no
DESCRIPION           transfer-car
                     transfer-date
                     seq-no

RELATIONSHIP-TYPE    manuf-by
DIMENSION            2
COLLECTION           manufacturer
                     car-model
CARDINALITY          manufacturer 0,n
                     car-model    1,1

RELATIONSHIP-TYPE    made-by
DIMENSION            2
COLLECTION           manufacturer
                     car
CARDINALITY          manufacturer 0,n
                     car          1,1
```

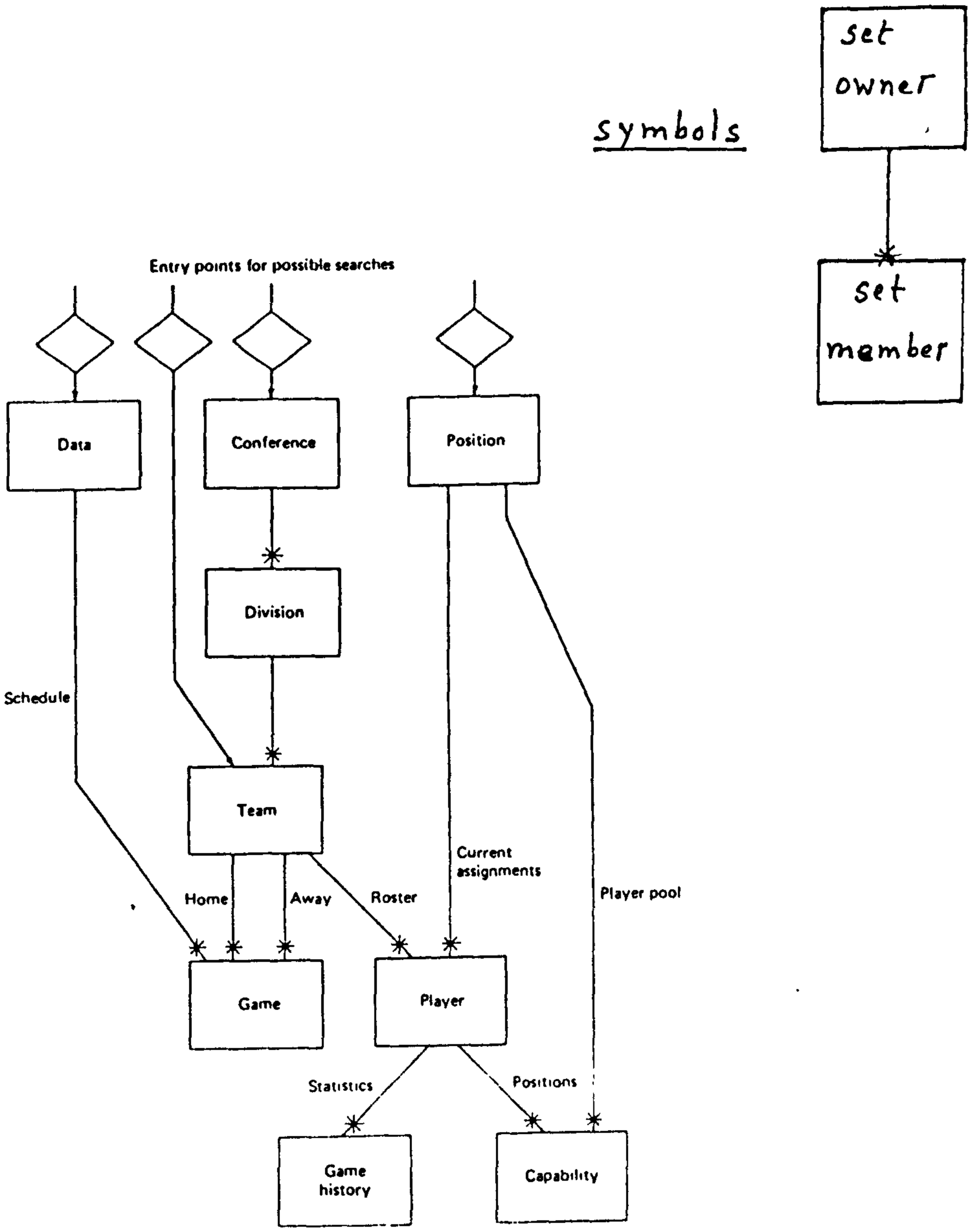
types and of set membership and ownership (using CODASYL terminology). While they are initially intended for use at the more detailed stages of database design, they have more recently been used also for conceptual modelling, and perhaps may be seen as occupying an intermediate position between the two. An example of a Bachman diagram is shown in figure 5-15.

## Identification matrices

The early versions of Systematics (see Grindley 1972, 1975) preceded relational database theory in realising the importance of the key/non-key relationship. In Systematics, key attributes are called primary identifiers and non-key attributes are called secondary identifiers. The concept of grouping non-key attributes with a common key into a named relation is not developed in Systematics. Instead it provides a notation for recording identification relationships between data items and for combining these in a matrix form (identification dictionary). There is a matrix column for every primary identifier and a matrix row for every identifier whether primary or secondary. The relational concept of foreign keys is not explicitly discussed. The matrix provides a concise tool for analysing possible access paths, which in the relational model must be specified by use of operations of relational calculus/algebra.

The purpose of identification analysis in Systematics (for which no software tools exist) is to permit the design of a system to be validated for consistency in the particular sense that a given trigger

Figure 5.15: example of Bachman diagram.
Source: NCC (1971).



input is sufficient to produce a given output. The technique was not intended to apply to databases, which indeed are essentially ignored in Systematics. An example of an identification dictionary in Systematics is shown in figure 5-16.

## Data abstraction

Data abstraction is an attempt by the programming research community to develop ways of describing and handling data objects independently of their syntax and physical representation. (Conceptual schemata, as already noted, are the outcome of a similar attempt by the database research community.) The main thrust of data abstraction has been concerned with general data types (e.g. stacks); with asserting their properties, the operations that can be carried out on them, and the effects of these operations; and with demonstrating useful proofs of correctness. Such general types, with their associated proofs, are seen as useful building blocks in the design of provably correct programs.

Less attention is paid to what might be called specific data types (e.g. product number, marital status), which it is equally possible to design and study at a similar level of abstraction. An example of data abstraction is shown in figure 5-17.

## Data dictionaries

A data dictionary offers a structure for holding information about named objects in or related to a system; in practice these need not be

Figure 5.16: example of identification dictionary.
Source: GRINDLEY (1965).

PRIMARY IDENTIFICATION DICTIONARY

System: Order Processing — Page 1

| DATA SETS | ET | Our Order No | Customer No | Product No | Despatch No | Invoice No | Batch No |
|---|---|---|---|---|---|---|---|
| ET | | 1 | | | 2 | 3 | 4 |
| Despatch No | 1 | 1 | | | | | |
| Customer No. | | 1 | | | | | |
| Customer Name | 1 | | 1 | | | | |
| Delivery Address | | 1 | | | | | |
| Customer Order No. | | 1 | | | | | |
| Order Date | | 1 | | | | | |
| Product No. | | | | | | 1 | |
| Qty. to Despatch | | | | 1 | 1 | | |
| Our Order No. | | | | | 1 | | |
| Invoice No | | | | | 1 | | |
| Customer Address | 1 | | 1 | | | | |
| Qty. Despatched | | | | 1 | 1 | | |
| Price | 1 | | | 1 | | | |
| Item Total | | | | 1 | | 1 | |
| Discount Total | | | | | | 1 | |
| Despatch Date | | | | | 1 | | |
| Invoice Total | | | | | | 1 | |
| Balance | 1 | | 1 | | | | |
| Discount | | | | 1 | | 1 | |
| Anticipated Account Balance | 1 | | 1 | | | | |
| Credit Limit | 1 | | 1 | | | | |
| Outstanding Qty. | 1 | 1 | | 1 | | | |
| Qty. Ordered | | 1 | | 1 | | | |
| Receipts | | | | | | 1 | |
| Free Stock | 1 | | | 1 | | | |
| Pipeline Qty. | | | | 1 | 1 | | |
| Stock | 1 | | | 1 | | | |
| Discount Rate | | | | 1 | 1 | | |
| Area | 1 | | 1 | | | | |
| Customer Type | 1 | | 1 | | | | |
| Quota | 1 | | 1 | 1 | | | |
| Payment | | | | | 1 | | |
| Pipeline Value | | | | 1 | 1 | | |
| Batch No | | | | | | | |

Figure 5.17: example of data abstraction.
Source: DARLINGTONS and others (edtrs) (1983).

NOTES ON USING TYPES AND TYPE ABSTRACTION IN FUNCTIONAL PROGRAMMING

***Abstract Type*** GradeBook

***Operators***

```
create: any                                          --> GradeBook
addGrade: GradeBook X Student X Test X Grade --> GradeBook
remGrade: GradeBook X Student X Test         --> GradeBook
inCourse: GradeBook X Student                --> Boolean
sGrades: GradeBook X Student                 --> Sequence[Grade]
tGrades: GradeBook X Test                    --> Sequence[Grade]
allStudents: GradeBook                       --> Sequence[Student]
```

***Axioms***

*for all* s,s':Student, t,t':Test, g:Grade, gb:GradeBook

remGrade∘[create,s,t] = create
remGrade∘[addGrade∘[gb,s,t,g],s',t'] =
    Student$eq∘[s,s'] ∧ Test$eq(t,t') →
        remGrade∘[gb,s',t'] ;
        addGrade∘[remGrade∘[gb,s',t'],s,t,g]

inCourse∘[create,s] = $\overline{F}$
inCourse∘[addGrade∘[gb,s,t,g],s'] =
    Student$eq∘[s,s'] ∨ inCourse∘[gb,s']

sGrades∘[create,s] = [ ]
sGrades∘[addGrade∘[gb,s,t,g],s'] =
    Student$eq∘[s,s']→
        apndl∘[g,sGrades∘[remGrade∘[gb,s,t],s']];
        sGrades∘[gb,s']

tGrades∘[create,t] = [ ]
tGrades∘[addGrade∘[gb,s,t,g],t'] =
    Test$eq∘[t,t']→
        apndl∘[g,tGrades∘[remGrade∘[gb,s,t],t']];
        tGrades∘[gb,t]

allStudents∘create = [ ]
allStudents∘addGrade∘[gb,s,t,g] =
    inCourse∘[gb,s]→
        allStudents∘gb;
        apndl∘[s,allStudents∘gb]

confined to data objects (though they will predominate) but may also include processes and real-world entities (physical and abstract). A data dictionary may be in manual or automated form.

Figure 5-18 shows one approach to organising a manual data dictionary, using index cards, with a different card layout for each of following entity types: data element, data structure, data flow, data store, process. The layouts indicate the information which might usefully be collected for each type of entity.

Automated data dictionaries offer obvious advantages over manual ones, in terms of ease of editing, searching, production of listings, etc. Many such systems are on the market, and a recent survey indicated about 2,000 users for the top fifteen products in this field. DATAMANAGER is one such product, and figure 5-19 shows some sample listings produced by it.

The American National Standards Institute and the National Bureau of Standards have initiated projects for the standardisation of data dictionary software. Additional references are FRANK W A and others (1982), LEONG HONG (1982), LOMAX (1977), WINDSOR (1980), BCS (1977) and EHRENSBERGER (1977).

Figure 5.18: example of manual data dictionary.
Source: GANE and Sarson (1979).

STATE-PROVINCE-CODE — Data Element

Short description: Two letter code, for each state/territory of U.S. or province of Canada

Type: (A) AN N

Aliases (contexts): (STATE (BAL) STATE-CODE (SALES SYSTEM)
SHORT-STATE (mail room)

| IF Discrete | | IF Continuous |
|---|---|---|
| Value | Meaning | Range of values |
| AK | Alaska | |
| AL | Alabama | Typical value |
| AR | Arkansas | Length: 2 characters |
| AS | American Samoa | Internal representation: Not yet assigned |
| AZ | Arizona | |

(If more than 5 values, continue on reverse or give reference to separate sheet) see over: total 63

Other editing information: May be required to match zip-code

Related data structures/elements: Customer address, supplier address

figure 5-18 (a). Specimen form for recording data elements

ORDER — Data Structure

Short description: Data structure representing customer order for 1 or more books.

ORDER-IDENTIFICATION
ORDER-DATE
[CUSTOMER-ORDER-NUM]
CUSTOMER-DETAILS
ORGANIZATION-NAME
[PERSON-AUTHORIZING]
PHONE
[SHIP-TO-ADDRESS]
[BILL-TO-ADDRESS]
BOOK-DETAILS* (1-)

Related data flows/structures: C-1, 1-3, 1-5/6, 6-D4, 6-13, 6-7, 13-D8, 13-D10, D8-16, 16-7

Volume information: Averaging 100/day in current system. New system may rise to 1,000/day.

figure 5-18 (b) Specimen form for recording data structure

IN|O|N|L-|S|H|I|P|P|A|B|L|E|-|I|T|E|M|S| . . . . . | | | DATA FLOW

Source ref: 6 Description Verify inventory available

Destn. ref: 13 Description Create back order or requisition

Expanded description Details of each item for which an acceptable order has been received, but which cannot be shipped, either because it is out of stock or because it is not carried in inventory

Included data structures:

Order
- Order-identification
- Customer-details
- Book-details *
- Non ship-cause

Where the original order is for multiple books, only some of them may appear in this data flow.

Volume information:

Out of stock - about 5 per week (this is acceptable to management)

Non inventory items - about 30 per week

No growth data

figure 5-18(c) Specimen form for recording data flow

O|R|D|E|R|-|H|I|S|T|O|R|Y| | | | | | | | | | | | | | | | | | | | Data Store ref: D4

Description All orders accepted for fulfillment - last six months.

Data flows in:

6 - D4 All orders

Data flows out (search arguments)

D4-10 Order details (customer name, order date)

D4-11 Sales detail (ISBN, publisher's name)

D4-9 Past demand (ISBN)

Contents:

Order
- Order-identification
- Customer-details
- Book-details * (1-)

Immediate access analysis is to be found in Functional spec, Section 8.17

Physical organization:

Not yet specified

figure 5-18(d) Specimen form for recording data store

V|E|R|I|F|Y|-|C|R|E|D|I|T|-|I|S|-|O|K| | | | | | | | | | | | | | | | | | | | Process ref: 3

Description Decide whether orders without prepayment can be shipped, or whether prepayment should be demanded from customer.

| Inputs | Logic summary | Outputs |
|---|---|---|
| 1-3 ORDERS<br>D3-3 PAYMENT HISTORY<br>DATE ACCT. OPENED<br>INVOICE *<br>PAYMENT *<br>BALANCE ON ORDER | Retrieve payment history<br>If new customer, send prepayment request<br>If regular customer (average of two orders per month) OK the order unless balance overdue is more than two months old<br>For previous customers who are not regular, OK the order, unless they have any balance overdue | 3-C Prepayment request [Reminder of balance]<br>3-D3 New balance on order<br>3-6 Orders with credit OK |

Physical ref Part of on-line order entry, 31 717

Full details of this logic can be found in Functional specification, Section 7.2

figure 5-18(e) Specimen form for recording process

Figure 5.19: example of automated dictionary.
Source: GANE and Sarson (1979).

LIST OF MEMBERS

| MEMBER NAME | TYPE | USAGE | CONDITION | AC | ALT | REM | OWNER |
|---|---|---|---|---|---|---|---|
| ACTION-CODE | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| ADDRESS | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| ADDRESS-UPDATE | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| BASIC-UPDATE | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| DEDUCT-CODE | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| DELETE | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| DEPARTMENT | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-HISTORY-LIST | FILE | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-HISTORY-MASTER | FILE | 3 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-HISTORY-REPORT | PROGRAM | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-HISTORY-UPDATE | PROGRAM | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-LIST | FILE | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-MASTER | FILE | 4 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-MASTER-UPDATE | PROGRAM | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-NUMBER | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-RECORD | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-REPORT | PROGRAM | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-TRANSACTIONS | FILE | 1 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-TRANSACTIONS-SORTED | FILE | 3 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-VET | PROGRAM | 1 | SCE ENC | 0 | 0 | 0 | |
| FILLER00002 | ITEM | 13 | SCE ENC | 0 | 0 | 0 | |
| HISTORY-RECORD | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| HISTORY-REPORT-RECORD | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| JOB-COUNT | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| JOB-ENTRY | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| JOB-STATUS | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| JOB-TITLE | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| MAINTAIN-EMPLOYEE-DATA | SYSTEM | 0 | SCE ENC | 0 | 0 | 0 | |
| MAINTAIN-EMPLOYEE-HISTORY | SYSTEM | 0 | SCE ENC | 0 | 0 | 0 | |
| NAME | ITEM | 4 | SCE ENC | 0 | 0 | 0 | |
| REPORT-COUNT | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| REPORT-RECORD | GROUP | 1 | SCE ENC | 0 | 0 | 0 | |
| SALARY | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| SOCIAL-SECURITY-NUMBER | ITEM | 3 | SCE ENC | 0 | 0 | 0 | |
| TAXCODE | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| TRANSACTION-RECORD | GROUP | 2 | SCE ENC | 0 | 0 | 0 | |
| TYPE | ITEM | 6 | SCE ENC | 0 | 0 | 0 | |

LIST CONTAINS
14 ITEMS
10 GROUPS
6 FILES
5 PROGRAMS
2 SYSTEMS
37 MEMBERS IN TOTAL

figure 5-19 (a) DATAMANAGER output

LIST OF ITEMS

| MEMBER NAME | TYPE | USAGE | CONDITION | AC | ALT | REM | OWNER |
|---|---|---|---|---|---|---|---|
| ACTION-CODE | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| ADDRESS | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| DEDUCT-CODE | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| DEPARTMENT | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| EMPLOYEE-NUMBER | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| FILLER00002 | ITEM | 13 | SCE ENC | 0 | 0 | 0 | |
| JOB-COUNT | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| JOB-STATUS | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| JOB-TITLE | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| NAME | ITEM | 4 | SCE ENC | 0 | 0 | 0 | |
| SALARY | ITEM | 5 | SCE ENC | 0 | 0 | 0 | |
| SOCIAL-SECURITY-NUMBER | ITEM | 3 | SCE ENC | 0 | 0 | 0 | |
| TAXCODE | ITEM | 2 | SCE ENC | 0 | 0 | 0 | |
| TYPE | ITEM | 6 | SCE ENC | 0 | 0 | 0 | |

LIST CONTAINS 14 ITEMS

figure 5-19.(b) LIST ITEMS output

```
WHICH FILES USE DEPARTMENT.

THE FOLLOWING USE ITEM DEPARTMENT
 FILES                                  EMPLOYEE-MASTER
                                        EMPLOYEE-HISTORY-MASTER
                                        EMPLOYEE-HISTORY-LIST
                                        EMPLOYEE-LIST
                                        EMPLOYEE-TRANSACTIONS
                                        EMPLOYEE-TRANSACTIONS-SORTED

        00019       WHICH PROGRAMS USE DEPARTMENT.

THE FOLLOWING USE ITEM DEPARTMENT
 PROGRAMS                               EMPLOYEE-HISTORY-REPORT
                                        EMPLOYEE-MASTER-UPDATE
                                        EMPLOYEE-REPORT
                                        EMPLOYEE-HISTORY-UPDATE
                                        EMPLOYEE-VET
```

figure 5-19(c)

```
WHAT USES DEPARTMENT.

ITEM                                    DEPARTMENT IS USED BY
  GROUP                                   EMPLOYEE-RECORD
  GROUP                                   HISTORY-RECORD
  GROUP                                   HISTORY-REPORT-RECORD
  GROUP                                   REPORT-RECORD
  GROUP                                   TRANSACTION-RECORD

GROUP                                   EMPLOYEE-RECORD IS USED BY
  FILE                                    EMPLOYEE-MASTER

GROUP                                   HISTORY-RECORD IS USED BY
  FILE                                    EMPLOYEE-HISTORY-MASTER

GROUP                                   HISTORY-REPORT-RECORD IS USED BY
  FILE                                    EMPLOYEE-HISTORY-LIST

GROUP                                   REPORT-RECORD IS USED BY
  FILE                                    EMPLOYEE-LIST

GROUP                                   TRANSACTION-RECORD IS USED BY
  FILE                                    EMPLOYEE-TRANSACTIONS
  FILE                                    EMPLOYEE-TRANSACTIONS-SORTED

FILE                                    EMPLOYEE-MASTER IS USED BY
  PROGRAM                                 EMPLOYEE-HISTORY-REPORT
  PROGRAM                                 EMPLOYEE-MASTER-UPDATE
  PROGRAM                                 EMPLOYEE-MASTER-UPDATE
  PROGRAM                                 EMPLOYEE-REPORT

FILE                                    EMPLOYEE-HISTORY-MASTER IS USED BY
  PROGRAM                                 EMPLOYEE-HISTORY-REPORT
  PROGRAM                                 EMPLOYEE-HISTORY-UPDATE
  PROGRAM                                 EMPLOYEE-HISTORY-UPDATE

FILE                                    EMPLOYEE-HISTORY-LIST IS USED BY
  PROGRAM                                 EMPLOYEE-HISTORY-REPORT

FILE                                    EMPLOYEE-LIST IS USED BY
  PROGRAM                                 EMPLOYEE-REPORT

FILE                                    EMPLOYEE-TRANSACTIONS IS USED BY
  PROGRAM                                 EMPLOYEE-VET

FILE                                    EMPLOYEE-TRANSACTIONS-SORTED IS USED BY
  PROGRAM                                 EMPLOYEE-HISTORY-UPDATE
  PROGRAM                                 EMPLOYEE-MASTER-UPDATE
  PROGRAM                                 EMPLOYEE-VET

PROGRAM                                 EMPLOYEE-HISTORY-REPORT IS USED BY
  SYSTEM                                  MAINTAIN-EMPLOYEE-HISTORY

PROGRAM                                 EMPLOYEE-MASTER-UPDATE IS USED BY
  SYSTEM                                  MAINTAIN-EMPLOYEE-DATA
```

figure 5-19(d) WHAT USES output

## 5.5 CONCLUSION

In reviewing the techniques covered by the above classification, one is struck by the following points.

(1) A technique may be adapted for use in various activities in the system development process.

(2) There is a small number of basic types of notation among which diagrammatic notations predominate.

(3) The duality between process and data is implicit in most techniques; but most techniques are strongly oriented towards one or the other.

(4) Any individual technique is likely to be closely related to, and overlap with, one or more other techniques.

(5) Taken together, this network of overlapping techniques, and the concepts (arising from various different view points) which underlie them, can constitute to a coherent conceptual model relevant to information system development.

# CHAPTER 6

# REQUIREMENTS DESCRIPTION FOR A SYSTEM DEVELOPMENT METHODOLOGY

## CONTENTS

6.1 Introduction

6.2 Principles which should underlie a development methodology

6.3 Subset of methodology addressed in this thesis

6.4 Conclusion

## 6.1 INTRODUCTION

This chapter presents an informal description (based on the review of the previous chapters) of the ideals of a system development methodology. It introduces the main part of research covered in this thesis, and concludes by identifying the need for a new development methodology.

## 6.2 PRINCIPLES WHICH SHOULD UNDERLIE A DEVELOPMENT METHODOLOGY

For a whole system, or for components at any level, it is necessary to understand:

- its function, and

- how its structure enables it to perform that function

and to integrate these understandings across all levels. POLANYI (1969) elaborates at some length on this basic notion.

In order to achieve such understanding, it is important that our knowledge of complex systems is well-structured; and that in turn relies on (a) good models of systems of the type under consideration, (b) information being presented to us in a well-structured way which accords with those models.

Langefors (1973) discusses the design of complex systems. In his eighth theorem he concludes:

" A system can only be designed to specified properties through a hierarchical system of design processes, in each of which every subsystem specified in a previous process is designed by organising a workable subsystem structure for it; and the system so designed will itself have a hierarchical structure".

A workable subsystem structure is previously defined as a subsystem structure such that the properties of the subsystems together with the iterations between them result in the properties specified for the

system as a whole.

This theorem stresses two key points.

(i) the importance of the design process being structured;

(ii) the importance of verification that each design step is consistent with the previous ones.

The activity of system development may be classified into four classes of intellectual activity, as follows.

Conjecture

This is the traditional "design" approach; the specialist thinks about the problem and searches for a solution.

Observation

The developer must discover a great deal of information about the problem domain in which he is working, and build up a rich mental model.

Analysis

Observations or conjectures are submitted to analysis, to deduce further information or to uncover errors.

Experiment

In this approach, for which the term "prototyping" or "piloting" maybe used, designs are implemented as rapidly as possible, may be with little initial concern for efficiency, and tested operationally. This approach stresses user participation and the learning nature of the development process. It reflects the diminishing distinctions between specialists and non-specialists and between development and operation.

As in science, in other branches of engineering, and in many other human activities, these approaches are complementary and difficult to separate in practice.

CHECKLAND (1976) distinguishes between "human activity systems" and "designed systems". For a designer it is essential to distinguish these two types of systems.

Designed systems behave predictably, can be described formally and are used to tackle "hard" or convergent problems. Human activity systems, by contrast, do not behave predictably, cannot be described precisely, and have diverse and conflicting aims. There do not exist unique (testable) accounts of human activity systems. These systems are faced with "soft" or divergent problems. They may incorporate designed systems as components. They may to some extent themselves be designed; but such design as they display does not fully reflect their behaviour.

The job of the designer is to understand the human activity system, to improve its effectiveness by embedding an appropriate number of

designed systems in it, and then to develop, install and maintain such systems.

A system developer, when designing an information system, can be viewed as inhabiting three distinct worlds, which are summarised as follows.

World 1 is the product system or target system (TS), the system which he is developing. It is a world of data and functions, programs and schemas, volumes and frequencies, discs, processors, terminals and lines, reliability calculations, and so on. It is a precise and measurable world. The focus is on information, and the physical objects (eg. people and machines) are there because of their information handling roles.

World 2 is the world of the environment host (HS), the human activity system within which the target system will be embedded. World 2 is the world which produces and consumes the information handled by the target system. It is a world of multiple, competing, imprecise, unagreed and changing objectives; a social, political and economic world; a world of people, each with a set of roles; a world of multitudinous objects (factories, products, orders, weapons,..) and events. Information, however important, is likely to be mostly of secondary importance in this world; and the most important information is often informal and unpredictable.

World 3 is the world of the development system (DS), the system of which the developer's work forms a part. Like the environment system,

the DS is a human activity system; and, just as the TS is embedded within the HS, so it is embedded within the DS, but in a different way. World 3 is a world of project budgets and schedules, project teams and objectives, documentation and standards, and so on.

While inhabiting world 1, system developers collaborate with non-technical colleagues in the activity of requirements analysis, with the purpose of producing a document which we may call a requirements description for a new target system. It has the following characteristics.

1. It records only a subset of knowledge acquired during requirements analysis.
2. It is described in natural language, understandable by all categories of users.
3. The requirements expressed differ in nature and precision, from the ambiguous and organisationally directed to the accurate and technically directed.
4. The document will contain much background information and argument to justify the requirements.
5. Its function will be as much political as technical.
6. It will be unsuitable for completeness, consistency and ambiguity checking.

System developers then extract from the above informal document a set of formal statements (called a system specification) which serves as the starting point for the task of developing the "designed information system".

A very clear recognition of the difference between the informal

requirements description and the formal system specification is vital. In most of the current literature of methodologies this distinction is insufficiently recognised.

It can be further argued that a system specification, and any subsequent subsystem specification, and should pass through three stages of refinement:

- outline specification (initial, or "key features", specification, in which the developer's first ideas about the object are expressed);

- complete specification (in which the required characteristics of the object are completely expressed); and

- verified specification (in which inconsistency, ambiguity and incompleteness have been detected and eliminated).

The concept of specification is crucial in thinking about system development. A specification is the description of what an object is to do (or does), as opposed to a design, which is a description of how it does it. Design involves selection between alternatives. Many methodologies see specification as an activity which occurs early in the "life cycle" and does not occur thereafter. On the contrary, it is not only the system as a whole that needs to be specified; if it is decomposed into subsystems, and into components (such as databases, interfaces and programs) then each of these needs to be specified at an appropriate point of time during system development.

As just indicated, corresponding to the specification of any object is the design of that object. Except at a very lowest level of decomposition, a design is expressed as a set of specifications for objects at the next level of detail. Specification, verification and design can thus be seen as activities which go hand in hand throughout the development process.

A logical specification may be expressed as a set of functions of various types, which we call (they will be explained later) dependency functions, derivation functions, composition functions, deletion functions, selection functions and trigger functions.

A system or (subsystem) specification consists of the following:

- logical specification, and
- performance specification (time and space constraints).

(Performance specification, estimating and monitoring are not considered further in this thesis.)

Design - decomposition into subsystems - results in a boundary specification for each subsystem. Where a subsystem boundary coextends with part of the whole system boundary, the specifications must match; and where one subsystem interfaces with another subsystem, their boundary specifications must again match.

Verification is of two main types.

(i) Verification of specification (verification that boundary and

functional specifications taken together are complete and consistent: "horizontal verification").

At any level of development, one may either perform a static analysis of a specification or perform a dynamic execution of it. At all stages prior to final execution on an actual machine, operational verification requires the provision of an appropriate virtual machine.

(ii) Verification of design (verification that a set of subsystems meets its higher-level specification in terms of (a) function, (b) performance: "vertical verification").

The final outcome of the process of decomposition is a complete and consistent set of low-level specifications of the following types.

1. Database specification
2. Interface specifications
3. Program specifications

To this point, development has been in abstract terms; now the abstract specifications must be made concrete. For programs, this is relatively straightforward; the function networks can be transformed into code (Jackson-like formalism may be used). The more difficult problems lie with the database and the interfaces; each of these must be physically designed, independently of the programs which use them. Each requires a specialist development method.

Another very important feature of a methodology is that it should permit separation of concerns. The discussion so far has concentrated on what might be called the pure capabilities of a system - those functions which it would contain if it were to operate in a perfect world. In fact, it will operate in a world where things go wrong; and, as we all know, a very large and important part of system development is concerned with capabilities for dealing with things that go wrong. These may be classified as (a) error handling, (b) recovery from breakdown, (c) access control, (d) measures to increase reliability by building in redundancy. As a group, they may be referred to as "error, failure and misuse". Other separate concerns include performance (specification, estimating, maintaining: referred to earlier in this section), and both project and product management. Fully recognising their importance, it is desirable that a methodology should nevertheless enable and encourage designers to deal with these issues separately from the pure logic and from each other. These separate concerns are not further addressed in this thesis.

What we may call the style of a methodology may vary on a spectrum from authoritarian to liberal. In the absence of any strong reason otherwise, methodologies should seek to be liberal. The following comments relate to this observation.

(i) To prescribe system development work exactly, in terms of tasks involved and their sequence, is tempting; but it is neither desirable (given the varying characteristics both of the system and project teams) nor likely to succeed (given human nature).

(ii) There are varying views about the starting point of system development. To some it is a statement of the system inputs and outputs; to some it is the analysis of entities and/or events in the system environment; to some it is an analysis or model of the internal data objects that correspond to those entities and events; to some it is the statement of the system's functions. A methodology which took the authoritarian view that the starting point can only ever be one of those would have severely restricted the probabilities of its acceptance; worse, since the reasons for such diversity of starting points are as much objective (in the nature of the object systems) as subjective (in the prejudice of the developer), an authoritarian methodology applied to some projects could lead to counter-productive distortion of the development process.

(iii) Nevertheless, there is one aspect in which a methodology should be clear, unambiguous, dogmatic - and thus authoritarian. It is the aspect of the conceptual framework, or model, which it embodies, both of systems and of the system development process. This is the most important aspect of any methodology; and liberalism or fuzziness will lead to uncertainty and confusion. Most methodologies are liberal where they should be authoritarian in their conceptual framework; and authoritarian where they should be liberal in the extent to which they prescribe in detail the developer's tasks.

A methodology may be described at three "levels of abstraction".

The model is primary, a conceptual structure on the basis of which a system is viewed, and determines "what can be said about the system".

The language defines how that will be said; and there can be a choice of language forms to support any particular model. The chosen language in turn defines the primary inputs to a variety of software tools to meet particular requirements of developers. This implies that the better the initial model, the more likely it is that tools can in fact be specified to meet actual needs.

It is a fair comment on methodologies as a whole that their conceptual basis is either ignored or inadequately defined, that their notations are often informal, and that tools are absent, or inadequately powerful.

The more that effort and precision can be shifted towards the early phases of system development, the less likely it is that problems will occur later. Thus a formal and precise notation should be provided, in which precise statements can be made from the earliest stages of thinking, as well as software tools which can process those statements to the maximum benefit to the designer. This permits the early detection of errors and leads to a saving in cost and time in correcting these errors in the subsequent phases of system development.

However fertile the human mind may be, in dealing with large complex problems it needs strong frameworks. This constitutes the justification for the authoritarian element in methodologies. Man is, and needs to be, a classifier, a model builder; it enables him to make sense of natural or artificial reality when it would otherwise be too hard to grasp. That is what science does; science underlies all

engineering, and in talking about conceptual frameworks for systems methodologies we are talking about the scientific basis for the better construction of better systems.

## 6.3 SUBSET OF METHODOLOGY ADDRESSED IN THIS THESIS

The methodology proposed in this thesis is concerned with logical specification and design, within the overall system development process. It does not deal with physical design, with other subsequent development stages, or with the separate concerns identified in section 6.2. The aim is to provide a complete, consistent and coherent framework, which guides and supports a developer in his task of managing the development of product (or target) systems.

Just as a system developer investigates the activities of people in a particular organisation, generalises them, and specifies and designs target systems to be embedded in that organisation, so in this thesis we seek to investigate the activities of system developers, generalise them, and specify a conceptual model for a special kind of target system (a development support system).

## 6.4 CONCLUSION

It has been clear from the previous chapters that an ideal methodology does not exist at present.

It is argued that a methodology can be completely described on three levels: its conceptual framework or model; the languages or notations which it offers to developers; and the tools (software or intellectual techniques) which are provided to assist the developers. In one sense it is the tools that count; without them, the developer has nothing to enable him to do his job better. In another sense, it is the frameworks or models that count, for without good models there will not be good tools.

The essence of what is attempted is the provision of a comprehensive, robust and flexible framework for the development of an information system, founded on an evolutionary notion, which will support an appropriate notation and a set of software tools.

The previous chapters provided a foundation for the requirements of a methodology. The following chapters describe the methodology which has been developed along these lines.

# CHAPTER 7

# GENERAL CONCEPTUAL MODEL OF THE METHODOLOGY

## CONTENTS

7.1 Introduction

7.2 Context model

7.3 Informal models of product system

7.4 Formal model of product system

7.5 Development model

7.6 General model of system evolution

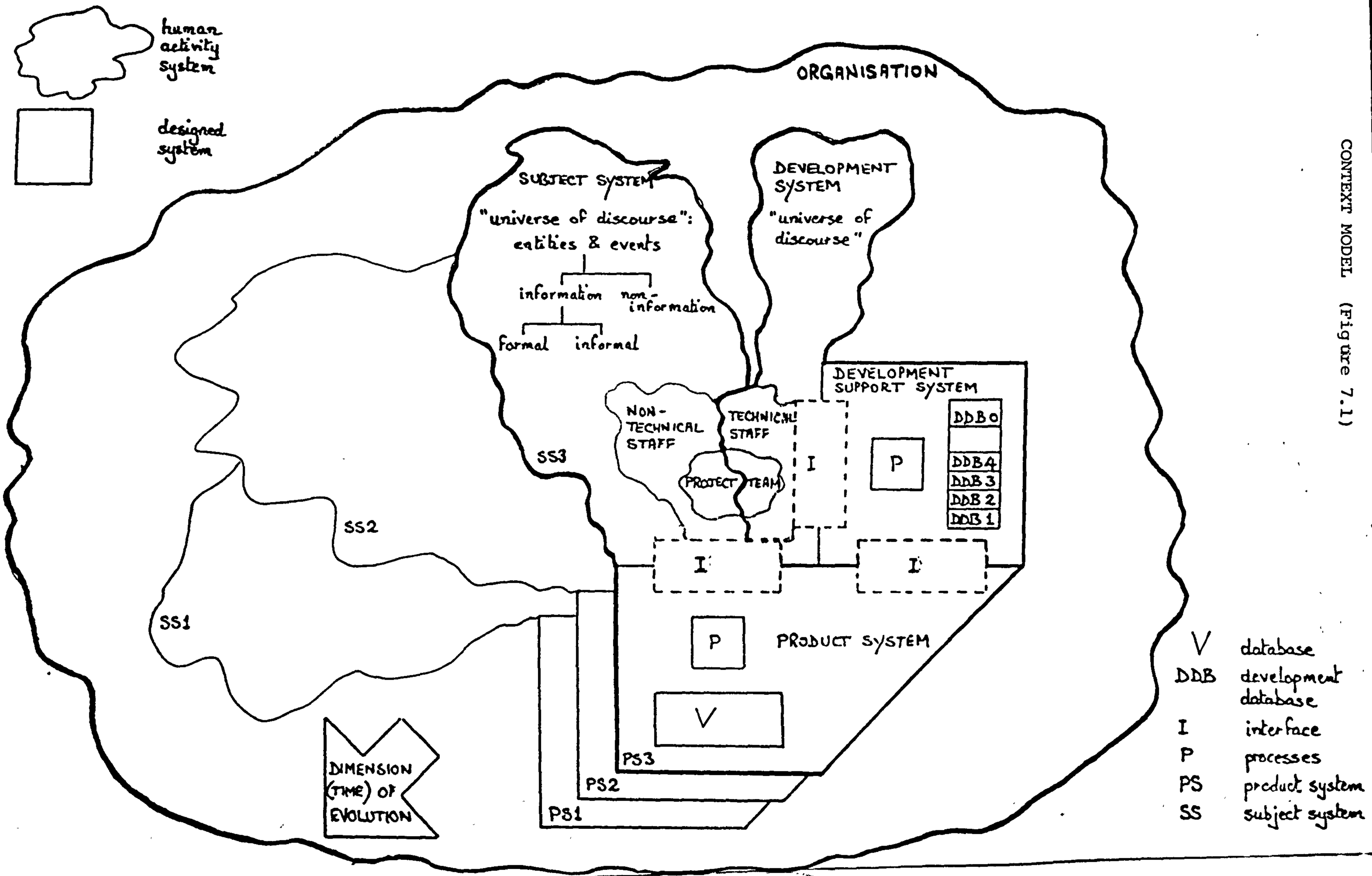
7.7 Conclusion

## 7.1 INTRODUCTION

According to KENT (1978), "A model is a basic system of constructs used in describing a reality. It reflects a person's deepest assumptions regarding the elementary essence of things. It may be called a world view. It provides the building blocks, the vocabulary that pervades all of a person's descriptions. In the broad arena of human thought, some alternative models might be composed of physical objects and motion, or of events seen statically in a time-space continuum, or of interactions of the mystical or spiritual forces, and so on".

A small set of models is proposed, in this chapter and the next, which it is suggested go a long way toward providing a coherent conceptual framework for system development. A key feature of these models is that they are intended to support methodologies which give system developers the maximum flexibility in using them for developing target systems taking account of the unavoidable differences of approach between developers, and of differences in requirements from one project to another.

TEXT BOUND INTO THE SPINE




CONTEXT MODEL (Figure 7.1)

## 7.2 CONTEXT MODEL

This model (figure 7-1) is designed to show the target (or product) system within its organisational context, and in relation to the development system which manages its evolution.

The model shows a number of information systems, each consisting of:

- a human activity system, and

- a designed system.

Each human activity system in turn comprises (a) a group of people and (b) a universe of discourse.

The three information systems on the left hand side of the figure (ie. SS1, SS2, SS3), each consists of:

- a subject system (which is the human activity system and is the subject of study for the purpose of system development);

- a product system (which is the designed or target system and is the product of the system development process).

The remaining information system, on the right hand side, consists of a development system (the human activity system within which technical system development is carried out) and a development support system (a designed system which constitutes the software tool set of a methodology).

Non-technical staff are shown interfacing to the subject system

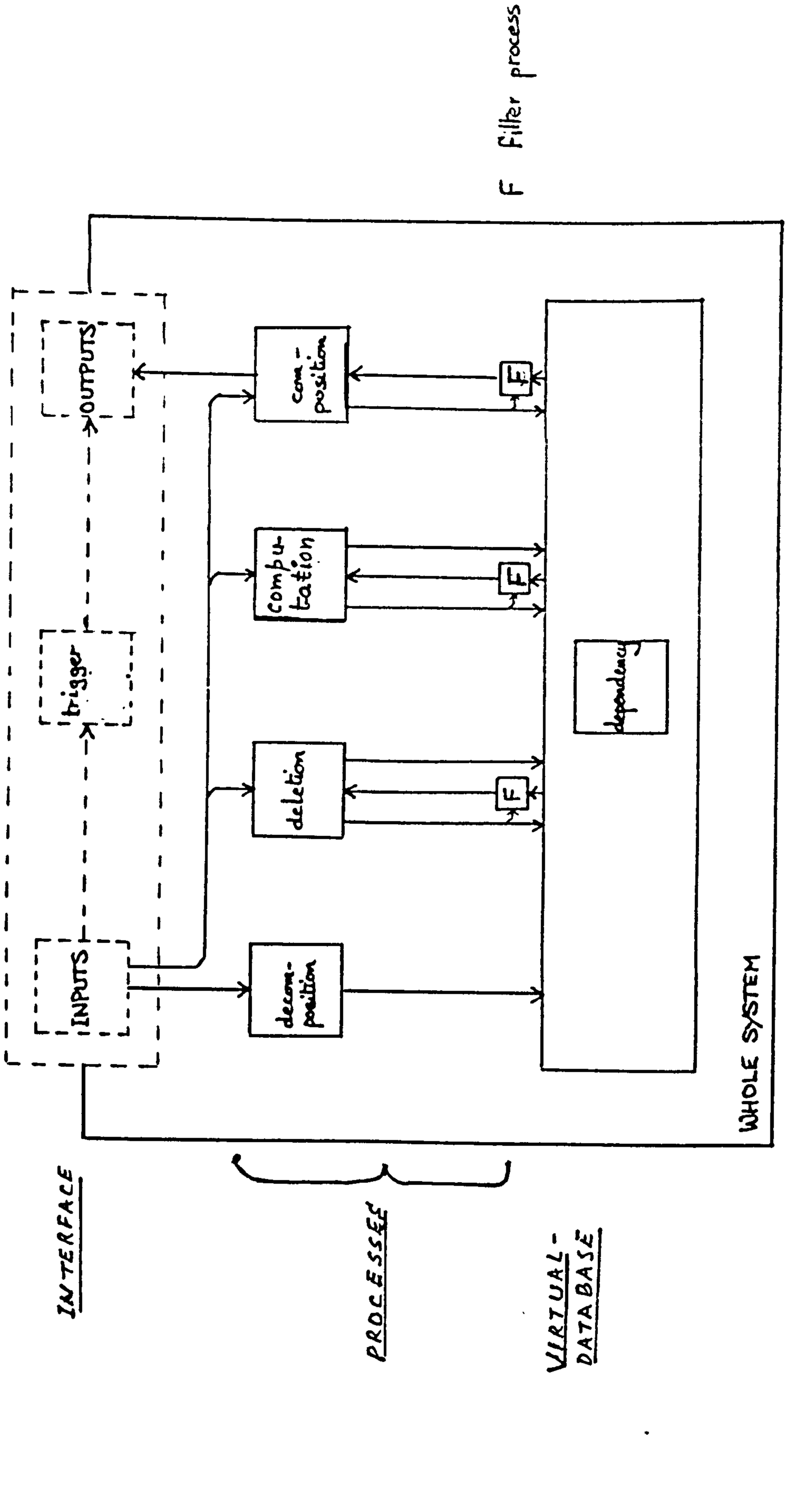
universe of discourse, to the technical staff, and to the product system; not all of them will be active on all interfaces. Technical staff are shown interfacing to the subject system and development system universe of discourse, to the non-technical staff, to the product system and to the development support system; again, not all of them will be active on all interfaces.

A subset of technical and non-technical staff constitutes the project team (which may change through time). It is also possible, though not shown, that non-technical project team members may interface with the development support system.

An evolutionary dimension is also shown in the figure, which indicates the development through time of all the systems shown.

INFORMAL MODEL OF PRODUCT SYSTEM SHOWING FUNCTIONAL STRUCTURE

Figure 7-2(a)



## 7.3 INFORMAL MODELS OF PRODUCT SYSTEM

The model (figure 7-2 (a)) showing the functional structure of a system is described as follows.

This is a partial refinement of a product system (ie. target system). It refines the three components shown in that higher-level model (ie. context model), which are: (i) I (interface), (ii) P (processes), (iii) V (virtual database) - and does so in terms of classes of functions.

The interface should be regarded as a line rather than a space, and inputs and outputs should be seen as having only an instantaneous existence as they cross it. Nevertheless, inputs and outputs, and the "trigger" relationships between them, are so important in specifying systems that it is useful to have a space within which they can be represented; for the reasons given, however, they are represented using broken lines. It should be added that trigger functions are abstract functional relationships, which are only actualised by sequences of other functions (of the classes shown in the lower part of the diagram).

The decomposition class of functions receives input messages and distributes their elementary components in the database. Note that these may be new values (insertions) or replacements for existing values (amendments).

The deletion class of functions are triggered by inputs, pass access arguments to the database, receive required data in return, and pass

the effected deletion back to the database.

The computation class of functions are triggered by inputs, pass access arguments to the database, receive required data in return, compute values of data items, and pass the computed values back to the database. Note that the item for which a value is computed may be different from any of the input items of the function, or it may be the same as one of them. In the latter case, we commonly call the computation an update.

The composition class of functions are triggered by inputs, pass access arguments to the database, and compose output messages from the elements supplied.

The filter class of functions apply selection criteria to tuples supplied by the database (this class of functions is often thought as being part of database management activity; in practice, however, it is often important to be explicit about selection criteria as part of the logical specification of a system and therefore, for the purpose of this model, they are shown as separate from the database (though close to it).

The dependency class of functions relate data items in the database, and specify what it is logically possible to retrieve from any given access argument.

There exist two main abstractions in the model, defined as follows.

(i) The database is considered to contain every data type

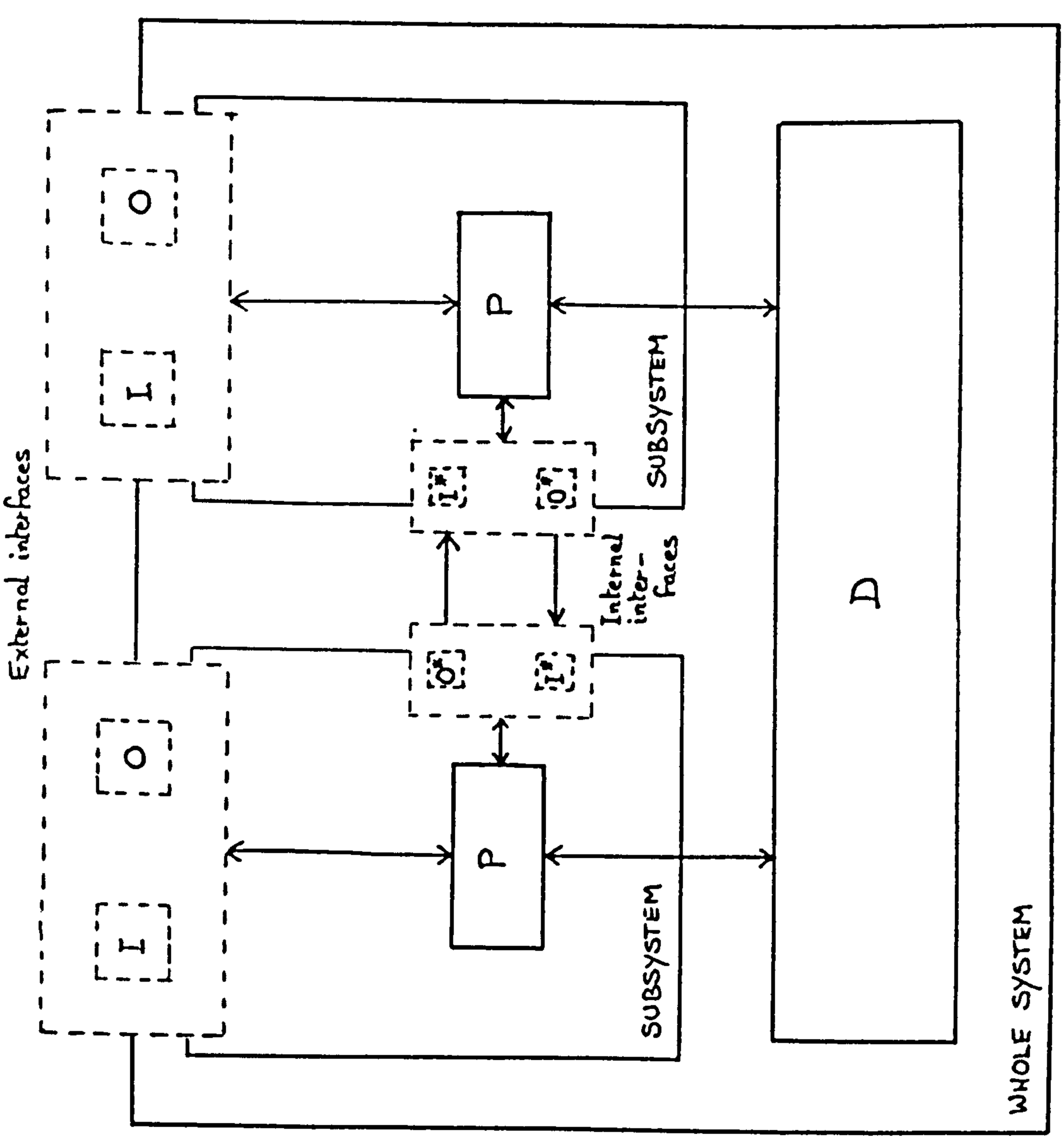
necessary for the definition of the system, irrespective of whether or not it will figure in the actual database as subsequently designed and implemented: it is therefore referred to as the virtual database. Every value, when input or computed, is considered as being immediately stored in the virtual database; and every value required for composition or output is considered as being retrieved from the virtual database. This proves to be a very useful simplification, and also enables the virtual database schema to incorporate all data types and their interrelationships, rather than just those in the actual database.

(ii) The model assumes that every separate deletion, derivation and output is separately triggered by some input; in practice, of course, a single input will probably trigger a cluster of such functions. That cluster is in fact a particularly ordered set of functions, of which the first function is triggered by an input, the second function is triggered by the first, and so on through the cluster.

This model provides the basis for horizontal verification checks.

INFORMAL MODEL OF PRODUCT SYSTEM SHOWING SUBSYSTEM STRUCTURE

Figure 7-2(b)



The model (figure 7-2 (b)) showing the subsystem structure is described as follows.

This is a different way of refining a product system, by decomposition into subsystems and the channels which connect them, which it would be difficult to combine with the detail shown in the previous diagram (ie the informal model showing functional structure).

The relationships between a parent (sub)system and its component subsystems is as follows.

- The input/output messages of the parent system are partitioned between the subsystems.

- The virtual database remains global to the set of subsystems.

- The processes of the parent (sub)systems are partitioned between the subsystems. In each subsystem there are additional decomposition and composition functions corresponding to the inputs and outputs crossing the internal interface.

This model provides the basis for vertical verification checks and additional horizontal checks.

## 7.4 FORMAL MODEL OF PRODUCT SYSTEM

This model expresses the relationships of the informal models of section 7.3 with greater clarity and precision, using straightforward notions of sets and functions. The notation employed is as follows.

### Symbols

| | |
|---|---|
| = | consists of |
| :: | is defined in terms of |
| { } | set of |
| / | or |
| A | algorithm |
| C | computation process |
| D | deletion process |
| F | filter process |
| I | input (external interface) |
| I* | input (internal interface) |

O output (external interface)

O* output (internal interface)

P process

R relation

S system (whole)

S′ subsystem (intermediate)

S′′ subsystem (elementary)

T trigger

V virtual database

c bijective (candidate key) element

e element

k key element

n non-key element

r row (in relation)

s statement

## Formal model

S = V, {P}, {S'}, {S''} :: {I}, {O}, {I*}, {O*} ....(1)

S' = {P}, {S'}, {S''} :: V, {I}, {O}, {I*}, {O*} .... (2)

S'' = {P} :: V, {I}, {O}, {I*}, {O*} .... (3)

P = C / D / F .... (4)

C = A :: T, {R}, F .... (5)

D :: T, {R}, F .... (6)

F = A :: {R}, (C / D / O / O*) .... (7)

O = {R} :: T, F .... (8)

Note: the composition function is implicit in the statement O = {R}.

O* = {R} :: T, F .... (9)

Note: as for O.

I = {R} .... (10)

Note: the decomposition function is implicit in the statement I = {R}.

I* = {R} .... (11)

Note: as for I.

T = {R} .... (12)

V = {R} .... (13)

A = {s} (note: s is primitive) .... (14)

R = {r} .... (15)

r = {k}, {b}, {n}, {r} .... (16)

k = {e} .... (17)

b = {e} (note: e is primitive) .... (18)

$$n = \{e\} \quad \ldots\ldots (19)$$

Notes on the equations

(1) A whole system consists of its virtual database V, a set of processes {P}, and a set of subsystems both intermediate {S'} and elementary {S''}.

It is defined in terms of a set of inputs {I} and outputs {O} which cross the external interface, and a set of inputs {I*} and outputs {O*} which are linked by internal interfaces.

{I*}, {O*}, {S'}, {S"} may each be null.

(2), (3) The virtual database V is not partitioned among subsystems {S'}, {S''}, but remains global to them; it therefore appears in the right-hand parts of (2) and (3).

S" is shown as consisting purely of a set of processes {P}, in which form it maps directly onto an individual (logical) program.

(2) is recursive, in that S' appears in the left-hand part and in the middle part. This allows for an indefinite number of levels of decomposition.

(4) - (7) A process may be a computation process C, a deletion process D, or a filter process F.

A computation process consists of an algorithm A and is defined in terms of a trigger T, a set of relations serving as its input or argument {R}, and a filter process F which may select the particular rows from the virtual database which are to enter into the computation.

A filter process consists of an algorithm A and is defined in terms of a set of relations {R} retrieved from the virtual database, from which it will deliver selected rows to C, D, O or O*.

The algorithm for a deletion process is standard in all cases (ie. "delete") and can therefore remain implicit or unspecified. A deletion process is defined in terms of a trigger T, a set of relations {R} which are to be deleted, and a filter process F which may select the particular rows from the virtual database which are to be deleted.

(8), (9) Outputs, both across external and internal interfaces, consist of sets of relations {R}, and are defined in terms of triggers T and filter processes F.

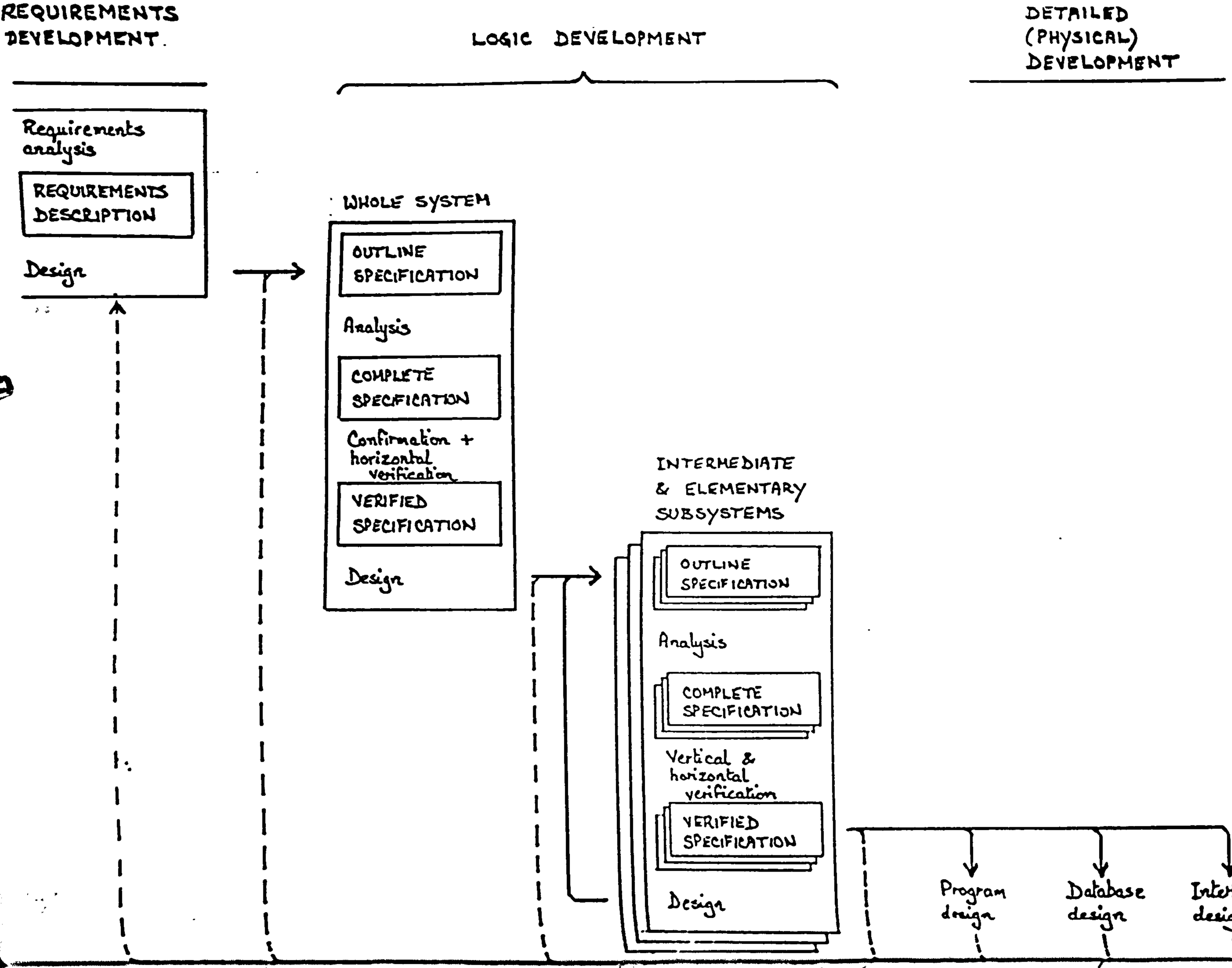
(10) - (13) Inputs, both across external interfaces and internal interfaces, consist of sets of relations {R}, as do triggers and the virtual database.

A trigger may be a named input, or an unnamed collection of elements, or an output from the "system clock".

The virtual database is considered to contain every data type necessary for the definition of the system, in a single level of storage, irrespective of whether or not it will be part of the actual database as subsequently designed and implemented. Every input or computed value is considered as being immediately stored in the virtual database; and every value required for computation, deletion or output is considered as being retrieved from the virtual database. This a very useful simplification, which enables the virtual database schema to incorporate all data types and their interrelationships, rather than (conventionally) those which are just in the actual database.

(14) An algorithm is a set of statements {s} in any convenient notation.

(15) - (19) A relation is a set of rows {r}, which break down into key elements {k}, bijective candidate key elements {c}, non-key elements {n}, and nested rows {r}: unnormalised relations are thus permitted.




DEVELOPMENT MODEL

Note Dotted lines indicate possible iteration paths.

Figure 7-3

## 7.5 DEVELOPMENT MODEL

The development model (figure 7-3) shows the activities by which a specification is produced for the whole system and then decomposed, via intermediate subsystems, to elementary subsystems. That is the point at which detailed (physical) development, which is outside the scope of this thesis, commences.

The purpose of this model is to provide a context for the requirements description for a development support system.

The model is summarised as follows.

Requirements development involves non-technical as well as technical staff. The requirements description is an informal document, in natural language, ranging over many issues concerning the proposed product system in addition to its technical characteristics. It may address issues of corporate objectives and strategy, organisational structure, motivation, and so on, and thus to a considerable extent be a political document.

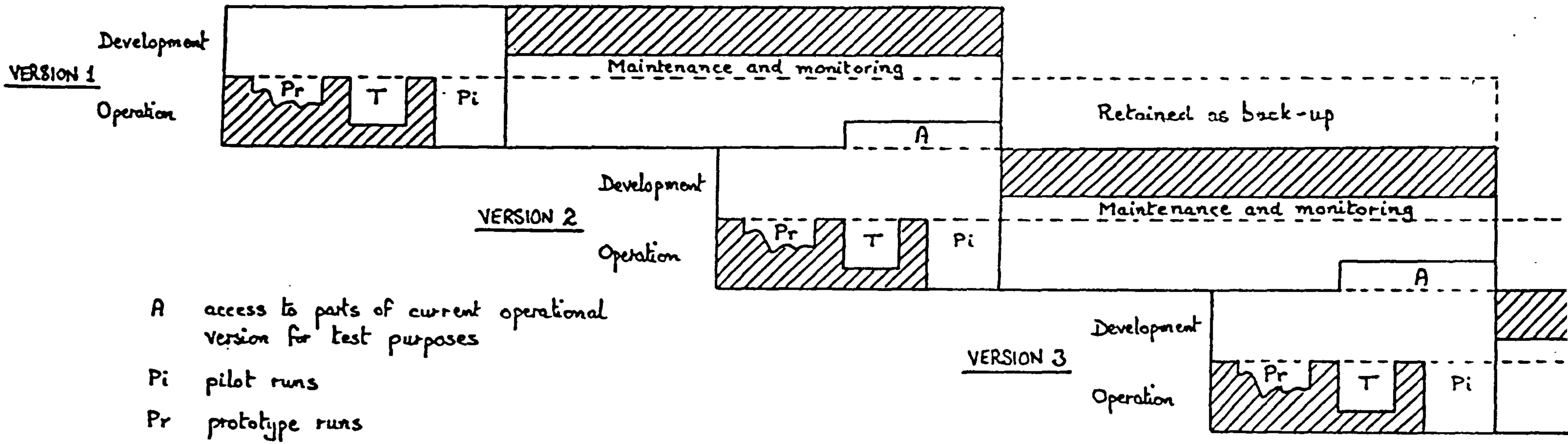
The first technical task is to extract an outline specification for the whole proposed system from the requirements description: this is a design task, in the sense that alternative solutions will usually present themselves, from which the "best" must be chosen. Working from now on in a formal notation, the developer says as much as he can, on the basis of what he has been given, about what the system is to do. It is unlikely, however, that this can amount to a complete specification: so he must embark on the analytical task of elucidating

the missing components and/or the contradictions in the outline specification. This complete specification is then ready to be verified: (i) informally "confirmed" against the requirements description, (ii) submitted to a formal "horizontal" (internal) verification for completeness and consistency.

A further design task may be then to decompose the system into subsystems, each one of which will then pass through the same sequence of steps, with the exception that formal "vertical" verification (ie. verifying the set of complete specifications against the parent specification from which they derive) replaces informal confirmation.

When decomposition has reached the point when subsystems can be equated with programs (a subjective decision), detailed (physical) development starts. Specialist sub-methodologies are required, including an optimisation capability where necessary, for the detailed development of programs, interfaces and the database.

The model shows the processes through which an individual version of the product system is developed. It applies not only to the "pure logic" of the product system, as modelled in sections 7.3 and 7.4, but also to the various "separate concerns" identified earlier, such as performance, errors, faults and misuse, etc. It is significantly different from the conventional life cycle "water fall" model, in that its foundation is a "canonical step" which integrates specification, analysis, design and verification.




A access to parts of current operational version for test purposes

Pi pilot runs

Pr prototype runs

T test runs

GENERAL MODEL OF SYSTEM EVOLUTION
(Figure 7-4)

## 7.6 GENERAL MODEL OF SYSTEM EVOLUTION

Section 7.5 offered a model of the development process for one version of a product system. This section presents a very simple model showing the relationships between the development and operational stages of a version, and between successive versions.

The model (figure 7-4) is summarised as follows.

The main purpose of the model is to show that, although there is (usually) a fairly clear cutover point for a version of a system from development to operational status, it is not the case that all activities preceding the cutover are wholly non-operational in nature, nor that all succeeding activities are exclusively operational.

Operation is loosely defined as computer processes carried out on data which the product system is designed to process (or dummy versions of such data), irrespective of whether the processes concerned reside within the product system itself or in the development support system.

Development is defined as the activities and computer processes carried out on information about the product system and its development.

A version is loosely defined as a product system which embodies significant function differences from its immediate predecessor version. (Terms like release or issue may be used for small-scale variations within versions.)

A prototype run involves "animating" some model of the product

system - ie. interpreting a representation of the system in a higher-level notation than the one in which it will ultimately go operational. It is usually primarily concerned with enabling users to verify requirements and early specifications.

A test run involves running an individual component, or a set of components, and is primarily concerned with verification at the detailed program level.

A pilot run involves running a cut-down version of the product system - ie. either with reduced functionality or with reduced data volumes. A pilot run is usually primarily concerned with verifying overall coherence and usability. Pilot running can be a much more significant overall strategy - ie. the whole of version x might be a pilot for version x+1, which itself might be an extended pilot, and so on.

The model achieves its maximum "depth" toward the right-hand side of the page, with versions 1, 2 and 3 playing concurrent roles.

## 7.7 CONCLUSION

The proposed set of models is a representation of what are felt to be the important features of the methodology (SSDM) under development.

Models of product systems and of the system development process may be classified on (at least) the following four dimensions.

- subject (what is it a model of - a static system or process, and of what broad category of system/process?)

- degree of generality (ranging from models general to all systems of a certain type, through those general to a class of systems of a certain type, to those of an individual system, and those applicable to parts of individual systems which have their own specific characteristics)

- degree of formality (from the more formal, using, say, mathematical notation, to the less formal, using, say, graphic notation)

- degree of authoritarianism (from the more authoritarian, constraining the developer's freedom, to the more liberal)

The models occupy a variety of points in that four-dimensional space. There is certainly room for others to be developed, but it is believed that the present set is sufficient for the limited purposes of this thesis.

In order to test the validity of the models (which are in effect

hypotheses), prototype development of notations and of software tools is being undertaken (by other members of the research team); but the expectation, realistically, is not that the outcome will be a complete methodology in practical use (though if it is, so much the better). The result of this research should permit (a) the evaluation, comparison and classification of existing methodologies, (b) the development of new methodologies based on sound and formal principles.

Science underlies all engineering; and, in talking about models for development of methodologies, we are talking about the scientific basis for the better construction of better systems. It is in studying these problems (with a keen appreciation of practical realities) that the academic community can best serve their practitioner colleagues.

# Chapter 8

# APPLICATION OF THE SYSTEM MODEL TO AN EXAMPLE INDIVIDUAL SYSTEM

## CONTENTS


8.1 Introduction

8.2 Requirements description for example individual system

8.3 System model (DDIR) of example individual system

8.4 Sample verifications from system model of example system

8.5 SSDL representation of the example individual system

8.6 Comments on SSDL

8.7 Conclusion

## 8.1 INTRODUCTION

This chapter presents a requirements description for an "example individual system" (ie. students' continuous assessment mark system), a formal specification of the "example system" in the form of a matrix. This matrix is a proposed structure for part of the development database internal representation (DDIR), which is in a highly structured form and supports several types of verification; examples of some types of verification are shown. In the case of any particular target system, the DDIR is set up from a set of statements in a System Specification and Design Language (SSDL); the SSDL statements for the example system are shown next (section 8.5). The chapter concludes with some brief general comments about SSDL.

## 8.2 REQUIREMENTS DESCRIPTION FOR EXAMPLE INDIVIDUAL SYSTEM

Title of the system: students' continuous assessment mark system

(Note that this example is a small part of a small system to handle students, continuous assessment marks over a two-year period. It has been selected because it permits the presentation of a reasonable selection of features at the level of the individual system model and its equivalent representation in SSDL. It is obvious that not all features which it would be necessary to model for a representatively large and complex system will appear in this example. It is claimed, however, that within the one side of A4 occupied by the requirements description it effectively illustrates the power and variety of the modelling approach.)

1. For each degree course covered by the system, a list of constituent course units is held; corresponding to each course unit is a weighting factor which is used when combining the mark for that unit with marks for other units in the degree course. (Any given course unit may have different weightings in different degree courses.) These degree details may need to be updated at any time.

2. At the start of each academic year, basic details are input (from a terminal keyboard) to the system for all new first-year students taking any of the relevant degree courses, and a hard-copy listing is produced. At the same time, previous first-year students are automatically changed to being second-year, and all previous second-year students are deleted. Information corresponding to an individual student can be deleted at any time.

3. A set of marks for the students taking a given course unit can be input to the system at any time. These marks will update the mean mark and the number of fails for each student concerned, which are maintained by the system. The fail mark is standard for all course units, but may change from time to time.

4. At any time, an authorised user may request a borderline list. This will list students who have a weighted mean mark to date which is equal to or less than a given value (to be specified by the enquirer), and/or who have a number of fails which is equal to or greater than a given number (again to be specified by the enquirer). The enquirer must also state whether he is concerned with first-year or second-year students.

5. Students are identified by student numbers; it is assumed that student names are not unique. Courses are identified by course numbers only. Degree courses are identified both by degree codes (e.g. BCS for Biology and Computer Science) and by full degree titles; there is a one-to-one correspondence between degree code and degree title.

SYSTEM MODEL Figure 8-1

| | Function label | Function name | statement parameter | student number | name | degree code | course number | mark | degree title | weight | fail mark | student year | student mean | student no of fails | year number | borderline mean | borderline no of fails | student no for deletion | comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | I1 | student details | a | k | e | e | | | | | | | | | | | | | |
| 2 | I2 | mark input | 1 | | | | k | | | | | | | | | | | | |
| 3 | | | 1a | k | | | | e | | | | | | | | | | | |
| 4 | I3 | degree details | 1 | | | k | | | b | | | | | | | | | | |
| 5 | | | 1a | | | | k | | | e | | | | | | | | | |
| 6 | 01 | borderline list | 1 | | | | | | | | | | | | | e | e | e | |
| 7 | | | a | | e | | | | e | | | | e | e | | | | | |
| 8 | | | | | | | | | | | | c | c | c | a | a | a | | |
| 9 | | | | | | | | | | | | | | | t | t | t | | |
| 10 | 02 | new student list | a | | e | | | | e | | | | | | | | | | |
| 11 | | | I1 | t | t | t | | | | | | | | | | | | | |
| 12 | P1 | student mean | a | k | | | | | | | | | e | | | | | | |
| 13 | | | ab | | | | | i | | i | | | | | | | | | |
| 14 | | | I2 | t | | | t | t | | | | | | | | | | | |
| 15 | P2 | student no of fails | a | k | | | | | | | | | | e | | | | | |
| 16 | | | ab | | | | | i | | | | | | | | | | | |
| 17 | | | 1 | | | | | | | | i | | | | | | | | |
| 18 | | | I2 | t | | | t | t | | | | | | | | | | | |
| 19 | P3 | student year | a | k | | | | | | | | e | | | | | | | |
| 20 | | | a | | | | | | | | | i | | | | | | | |
| 21 | | | | | | | | | | | | c | | | | | | | argument is int constant (null |
| 22 | | | (I1) | | | | | | | | | | | | | | | | |
| 23 | D1 | third-year students | a | e | e | e | | | | | | e | e | e | | | | | |
| 24 | | | ab | | | | e | e | | | | | | | | | | | |
| 25 | | | | | | | | | | | | c | | | | | | | argument is int constant (2) |
| 26 | | | (I1) | | | | | | | | | | | | | | | | |
| 27 | D2 | individual student | 1 | e | e | e | | | | | | e | e | e | | | | | |
| 28 | | | 1a | | | | e | e | | | | | | | | | | | |
| 29 | | | | c | | | | | | | | | | | | | | a | |
| 30 | | | | | | | | | | | | | | | | | | t | |

## 8.3 SYSTEM MODEL (DDIR) OF EXAMPLE INDIVIDUAL SYSTEM (figure 8-1)

The above system is now formally described in the form of a structured matrix called the "Development Database Internal Representation" (DDIR). As its name suggests, such a matrix would only be used internally by the software tools associated with the methodology; it would not be visible to the developer, who would operate at the level of the System Specification and Design Language (SSDL). For the purposes of this thesis only, the DDIR precedes the SSDL representation in this chapter, in order to demonstrate its direct relationship to the general formal model in chapter 7.

(Note that the example system is regarded as a complete system, with no decomposition into subsystems and therefore no internal interfaces.)

There are four function types, having function labels Ix (input functions, corresponding to decomposition functions, rows 1 to 5), Ox (output functions, corresponding to composition functions, rows 6 to 11), Cx (computation functions, rows 12 to 22), and Dx (deletion functions, rows 23 to 30). Filter functions are treated as subsidiary; where they occur, they are included as part of a major function definition.

The item names from the fourth column onwards (from student-number to student-number-for-deletion) are the data types in the virtual database. For the purpose of this level of abstraction, the virtual database contains all variables used in the system - many of which

will not be part of the actual database. As soon as the value of a variable is supplied (unless it is being used for triggering purposes only), whether via an input function or via a processing function, it is considered to be stored in the virtual database. Similarly, whenever the value of a variable is to be retrieved, whether for an output function, for a processing function or for a deletion function, it is considered as being retrieved from the virtual database. There is, in other words, no communication of data except via the database. This is a useful simplifying assumption.

Definition of the terms used in DDIR: statement parameter column

'a' denotes 'a' occurrences of one or more attributes.

'ab' denotes a set of 'b' occurrences of one or more attributes for each of the 'a' occurrences of one or more "higher-level" attributes.

Example (rows 12 and 13): process C1 computes 'a' occurrences of student mean; each occurrence is computed from 'b' occurrences of the pair (mark, weight).

'1' denotes a single occurrence of one or more attributes.

'1a' denotes a set of 'a' occurrences of one or more attributes corresponding to a single occurrence of one or more "higher-level" attributes.

Example (rows 2 and 3): input I2 consists of a single occurrence of course number, together with 'a' occurrences of the pair (student number, mark).

An input label indicates that the specified input triggers a particular output, process or deletion. If the label is not in parentheses (example: row 11), then the contents of the input are significant for triggering purposes; if the label is in parentheses (example: row 22), then it is not the contents of the input that are significant for triggering purposes but simply its arrival.

Definitions of the terms used in DDIR: remaining columns

'k' denotes a key attribute (in normal relational database terminology).

'b' denotes a candidate key attribute: a candidate key has a bijective relationship with a key.

'a' denotes an argument for a filter function.

'c' denotes a comparand for a filter function.

't' denotes a component of a trigger.

'i' denotes an input to a computation function.

'e' denotes an element (where its role can be inferred from its context and does not need to be explicitly defined).

Interpretation of the matrix

I1 (student details) consists of 'a' occurrences of: student number (key), name, degree code.

I2 (mark input) consists of 'one' occurrence of: course number (key); and 'a' occurrences of: student number (key), mark.

I3 (degree details) consists of 'one' occurrence of: degree code (key), degree title (candidate key); and 'a' occurrences of course number (key), weight.

O1 (borderline list) consists of 'one' occurrence of: year number, borderline mean, borderline number of fails; and 'a' occurrence of: name, degree title, student mean, student number of fails. A filter function is necessary to select these 'a' occurrences; the arguments for this function are: year number, borderline mean, borderline number of fails. These arguments are compared against elements: student year, student mean, student number of fails. The trigger for the borderline list is: year number, borderline mean, borderline number of fails; these trigger elements are not named as a predefined input.

O2 (new student list) consists of 'a' occurrences of: name, degree title. It is triggered by the arrival of an occurrence of I1, from which one element (student number) is significant for selection purposes.

C1 computes 'a' occurrences of student mean, for which the key is student number. Each occurrence is computed from 'b' occurrences of: mark, weight. The computation is triggered by the arrival of an

occurrence of I2, from which one element (student number) is significant for selection purposes.

C2 computes 'a' occurrences of student number of fails, for which the key is student number. Each occurrence is computed from 'b' occurrences of mark and 'one' occurrence of fail mark. The computation is triggered by the arrival of an occurrence of I2, from which one element (student number) is significant for selection purposes.

C3 computes 'a' occurrences of student year, for which the key is student number. Each occurrence is computed from one (= a/a) occurrence of student year. A filter function is necessary to select these 'a' occurrences; the argument for this function is a pair of constants (0, 1) to be compared against student year - ie. students of year 0 (just input) will be assigned year 1, and students with year 1 will be assigned year 2. The computation is triggered by the arrival of an occurrence of I1, though no elements of I1 are significant for selection purposes.

D1 deletes 'a' occurrences of the tuple (student number, name, degree code, student year, student mean, student number of fails); for each of these occurrences it deletes 'b' occurrences of the tuple (course number, mark). A filter function is necessary to select these 'a' occurrences; the argument for this function is a constant (2) to be compared against student year - ie. students with year 2 will be

deleted. The deletion is triggered by the arrival of an occurrence of I1, though no elements of I1 are significant for selection purposes.

D2 deletes 'one' occurrence of the tuple (student number, name, degree code, student year, student mean, student number of fails) and 'a' occurrences of the tuple (course number, mark). A filter function is necessary to select this one occurrence; the argument for this function is student number for deletion, to be compared against student number. The deletion is triggered by the submission of a value of student number for deletion, which is done via a non-predefined input.

## 8.4 SAMPLE VERIFICATIONS FROM SYSTEM MODEL OF EXAMPLE SYSTEM

A number of types of verifications are described below. It is not claimed to be an exhaustive enumeration of verification types.

### (1) Derivation dependency checks

The basis for this check is the hypothesis that the key of a derived item must be the same as the key of the set of arguments from it is derived. No proof of this hypothesis is offered. However, (1) it is intuitively convincing, (2) it has been found to be true experimentally. Two alternative methods for the derivation function "student mean" are presented respectively in the following. Note that "F" denotes "function of" and "K" denotes "key of".

(i) Derivation function student mean = F({mark, weight}) (ie. the function as specified in section 8.3)

LHS: K(student mean) = student number (given)

RHS: K (mark) = student number, course number (given)

K (weight) = degree code, course number (given)

K (degree code) = student number (given)

K (weight) = student number, course number

K (weight) = student number, course number

K (mark, weight) = student number, course number

K ({course number, mark, weight}) = student number

K ({mark, weight}) = student number

K (LHS) = K (RHS)

(ii) Derivation function student mean = F(weighted sum, sum of weights, {mark, weight})

(ie. an alternative specification of the same function)

LHS: K (student mean) = student number (given)

RHS: K (weighted sum) = student number (given)

K (sum of weights) = student number (given)

K (mark) = student number, course number (given)

K (weight) = degree code, course number (given)

K (degree code) = student number (given)

K (weight) = student number, course number

K (mark, weight) = student number, course number

K ({course number, mark, weight}) = student number

K ({mark, weight}) = student number

K (weighted sum, sum of weights, {mark, weight}) = student number

K (LHS) = K (RHS)

(2) Trigger consistency checks

An overall trigger function specifies the triggering input corresponding to each individual output. The composition function specifies the items constituting an individual output. Each item is either given or derived; and each derived item is the root of a derivation tree, the leaves of which are all given items. The root, and each intermediate node between the root and the leaves, represents a derivation process, for which a trigger may or may not be specified (at the developer's discretion). A property of this tree is that the overall output trigger propagates backwards from the root to all nodes, unless and until a node is encountered with a different specified trigger; that trigger then propagates backwards similarly within the remaining subtree for which that node is a root. In this way, triggers can be associated (by specification or by inference) with every derivation process in the system being specified.

Consistency checking can then be carried out for each derivation process separately, and follows the derivation dependency check. The derivation dependency check says that the key of the variable computed by the process must be the same as the key of the arguments(s); the trigger check says that the key of the argument(s) must be present in, or reachable from, the trigger of the process. Computation C1, as shown in (1) above, has arguments with the key student number. Values of this key are necessary to select the particular students for whom student mean is to be computed; and a set of values of student number are indeed present in the trigger (I2).

(3) Dependency check

If a non-key attribute occurs in more than one input, and is shown with different keys then it must be possible to account for that difference. This can be done if a "bijection" is known to exist between the different keys, whether that bijection is specified as part of an input definition or as part of a computation definition. For example, if students are numbered within each year, then student-year is part of the key for all attributes of student, but year-of-entry has a bijective relation with student-year and can be computed from it, and therefore could be used in place of student-year as the key.

(4) Derivation completeness check

As described in (2) above, the items comprising each output are either given or derived. Each derived item is the root of a derivation tree. This check simply says that all leaves of all derivation trees must be given items.

(5) Domain consistency check

Filter functions are defined in terms of arguments and of elements (in the virtual database) against which arguments are to be compared. This check ensures that pairs of arguments and comparands have consistent domains. (For domain definitions, see section 8.5 below.)

(6) Internal interface consistency check

This check applies to all non-elementary subsystems once they have been decomposed, and ensures that the output from one system is consistent with the input(s) to one or more other subsystems.

## 8.5 SSDL REPRESENTATION OF THE EXAMPLE INDIVIDUAL SYSTEM

```
students' continuous assessment marks

        system

        outputs are          borderline list
                             new student list

        inputs are           student details
                             degree details
                             mark input

        end


borderline list     output

                    triggered by     year number, borderline mean
                                     borderline number of fails

                    consists of      year number, borderline mean,
                                     borderline number of fails,
                                     set of (name, degree title,
                                     student mean, student no. of
                                     fails

                    filter           option clause 1

                    cardinality      1 (0 .. 120)

                    frequency        occasional

                    ordering         student mean ascending

                    end


• option clause 1   filter    if student number = year number and
                              (if student mean < borderline mean
                               or student number of fails >
                               borderline number of fails)
                               then select
```

end

| | | |
|---|---|---|
| new student list | output | |
| | triggered by | student details |
| | consists of | set of (name, degree title) |
| | cardinality | 1 .. 120 |
| | ordering | name ascending |
| | end | |

Note: frequency (1 per year) can be inferred from the frequency of the trigger (student details).

| | | |
|---|---|---|
| student details | input | |
| | consists of | set of (student number; name, degree code) |
| | cardinality | 1 ..120 |
| | frequency | 1 per year |
| | ordering | random |
| | end | |

| | | |
|---|---|---|
| mark input | input | |
| | consists of | course no., set of (student-number; mark) |
| | cardinality | 1 (1 .. 120) |
| | frequency | max. 36 per year |
| | ordering | student number ascending |
| | end | |

```
degree details      input
                    consists of     degree code: degree title;
                                    set of (course number;
                                            weight)
                    cardinality     1 (1 .. 60)
                    frequency       occasional
                    ordering        course number ascending
                    end

delete third year students
                    deletion
                    consists of     set of (student number,
                                    name, degree code,
                                    student year, student mean,
                                    student number of fails,
                                    set of (course number, mark))
                    triggered by    student details
                    filter          option clause 2
                    end
```

Note: frequency (1 per year) can be inferred from the frequency of the trigger (student details).

```
option clause 2     filter          if student year = 2
                                    then delete
                    end

delete individual student
                    deletion
                    consists of     student number, name,
                                    degree code, student year,
                                    student mean, student num-
                                    of fails, set of (course
                                            number, mark)
```

```
                    triggered by        student number for deletion
                    filter              option clause 3
                    frequency           occasional
                    end

option clause 3     filter              if student number =
                                        student number for deletion
                                        then delete
                    end

borderline mean     item
                    domain              real (1 .. 10)
                    function of         given
                    comment             this scalar is used to define the
                                        criteria for selection of students
                                        who are the members of borderline
                                        list
                    end

borderline number of fails
                    item
                    domain              real (1 .. 5)
                    function of         given
                    comment             this scalar is used to define
                                        the criteria for selection of
                                        students who are members of
                                        borderline list
                    end
```

```
course number       item
                    domain          integer (101 .. 512)
                    function of     given
                    comment         courses are numbered so that
                                    first digit represents term
                                    within the year,
                                    other two digits stand for course
                                    within the term
                    end

weight              item
                    domain          real (0 .. 1)
                    function of     given
                    comment         a student may offer 12 courses
                                    in a term, where each course has
                                    a corresponding weight,
                                    depending on the degree offered
                    end

degree code         item
                    domain          string
                    function of     given
                    comment         department offers 13 degrees and
                                    each degree has a unique code
                                    for its identification
                    end
```

| | | |
|---|---|---|
| degree title | item | |
| | domain | string |
| | function of | given |
| | comment | for each degree code there exists a corresponding degree title |
| | end | |
| mark | item | |
| | domain | integer (0 .. 25) |
| | subdomain | fail (0 .. 4), pass (5 .. 25) |
| | function of | given |
| | end | |
| student number for deletion | | |
| | item | |
| | domain | integer (0 .. 999) |
| | function of | given |
| | end | |
| student mean | item | |
| | domain | real (0 .. 25) |
| | function of | set of (mark, weight) |
| | depends on | student number |
| | triggered by | I2 |
| | end | |

```
name                    item
                        domain              string
                        function of         given
                        end

student number          item
                        domain              integer (0 .. 999)
                        function of         given
                        end

student number of fails
                        item
                        domain              integer (0 .. 60)
                        function of         fail mark, set of (mark)
                        depends on          student number
                        triggered by        I2
                        end

year number             item
                        domain              integer (1 .. 2)
                        function of         given
                        comment             year number is used as a
                                            selection scalar to
                                            determine the members of
                                            borderline list

                        end
```

```
student year        item
                    domain              integer (0 .. 2)
                    function of         student year
                    depends on          student number
                    filter              option clause 4
                    triggered by        Il
                    end

option clause 4     filter              if student year = 0 or
                                        if student year = 1
                                        then select
                    end

        fail mark   item
                    domain              integer (0 .. 25)
                                        current value is 4
                    function of         given
                    comment             each year a prescribed
                                        integer is set to serve
                                        as fail mark to select
                                        marks obtained
                                        by students in courses
                    end
```

## 8.6 COMMENTS ON SSDL

The previous section gave an example of SSDL (system specification and design language). The purpose of this section is to make some brief comments about the language. In an earlier draft of the thesis, a formal definition of the syntax of part of SSDL was provided; that was such a straightforward exercise, however, that it has been omitted in the interests of space.

SSDL is the second component of the SSDM methodology (models, language, tools). It is indeed to be capable of expressing all formally-expressible information generated during system development: information about "separate concerns" such as performance, error handling and project management, as well as about the "pure logic" of the system; and information about detailed (physical) development as well as about logic development. The example in section 8.5 showed some fairly straightforward pure logic, with the addition of some volume and frequency metrics.

The expressive requirements for SSDL are given from the model level: every relationship identified in a model must be capable of being expressed in the language. The other requirement is that the language should have a convenient and user-friendly syntactic form. This is met by adopting the following general form.

paragraph::=

'object name' 'object type'

'relationship 1' 'clause 1'

'relationship 2' 'clause 2' etc.

All objects named in clauses must appear on the left-hand side of a paragraph (except terminal objects). Relationships are named by reserved terms such as "triggered by", "consists of", "cardinality", "function of". Object types include "system", "output", "filter", "deletion", "item", etc.

The language thus follows the general structure of BNF, in which everything occurring on the right-hand side of a statement must appear on the left-hand side of another statement (except for terminal objects). The difference is that in BNF there is only one type of relationship identifier ("::="), whereas in SSDL there are many. The language also follows the general structure of a data dictionary. Both these structures are known to be easy to work with.

## 8.7 CONCLUSION

The model presented in this chapter is the application of the general model (chapter 7) to a particular example target (or product) system, providing an internal representation in matrix form of the set of statements that might be made about it by a system developer. This matrix is shown to be in a highly structured form, consisting of all the necessary and sufficient information about the product system, and to be verifiable for completeness and consistency. It is derived from a formally expressible language SSDL.

# CHAPTER 9

# SOFTWARE TOOLS REQUIREMENTS

## CONTENTS


9.1 Introduction

9.2 Overview of the tools

9.3 Development dialogue processor

9.4 Analyser

9.5 Logical simulator

9.6 Development database decomposer

9.7 Conclusion

## 9.1 INTRODUCTION

Software support is a vital aspect of the proposed methodology, providing a level of automation for the non-trivial development activities of specification, design and verification.

A specification language can be viewed as the expression of an underlying system model. In a similar sense, software tools can be viewed as being the expression of a development model.

The process of system design is argued to be largely heuristic in nature, involving:

(a) creation of tentative versions,

(b) verification and testing of proposed versions,

(c) selecting the best version,

(d) documenting design decisions.

In particular processes (a) and (c) require creativity, inventiveness and the capability to make value judgements and, consequently, these are best performed by human developers. Tasks (b) and (d) on the other hand are usually algorithmic and are, therefore, best automated. It follows therefore, that the design of an information system is best performed interactively by the pair (man, machine).

The provision of as much automation as possible makes the specification of a system, as a normal evolutionary process, safer and faster. Such support can be viewed as an extension of the modelling

and linguistic aspects of the proposed methodology.

This chapter presents an outline requirements description of the main software tools necessary to support the developer. The tools together constitute an integrated information system development support environment.

KEY

$U_1$ $U_2$ = specification and dialogue and reports (designer)

$U_3$ $U_4$ = analysis dialogue and reports (designer)

$U_5$ $U_6$ = simulation dialogue and reports (user and designer)

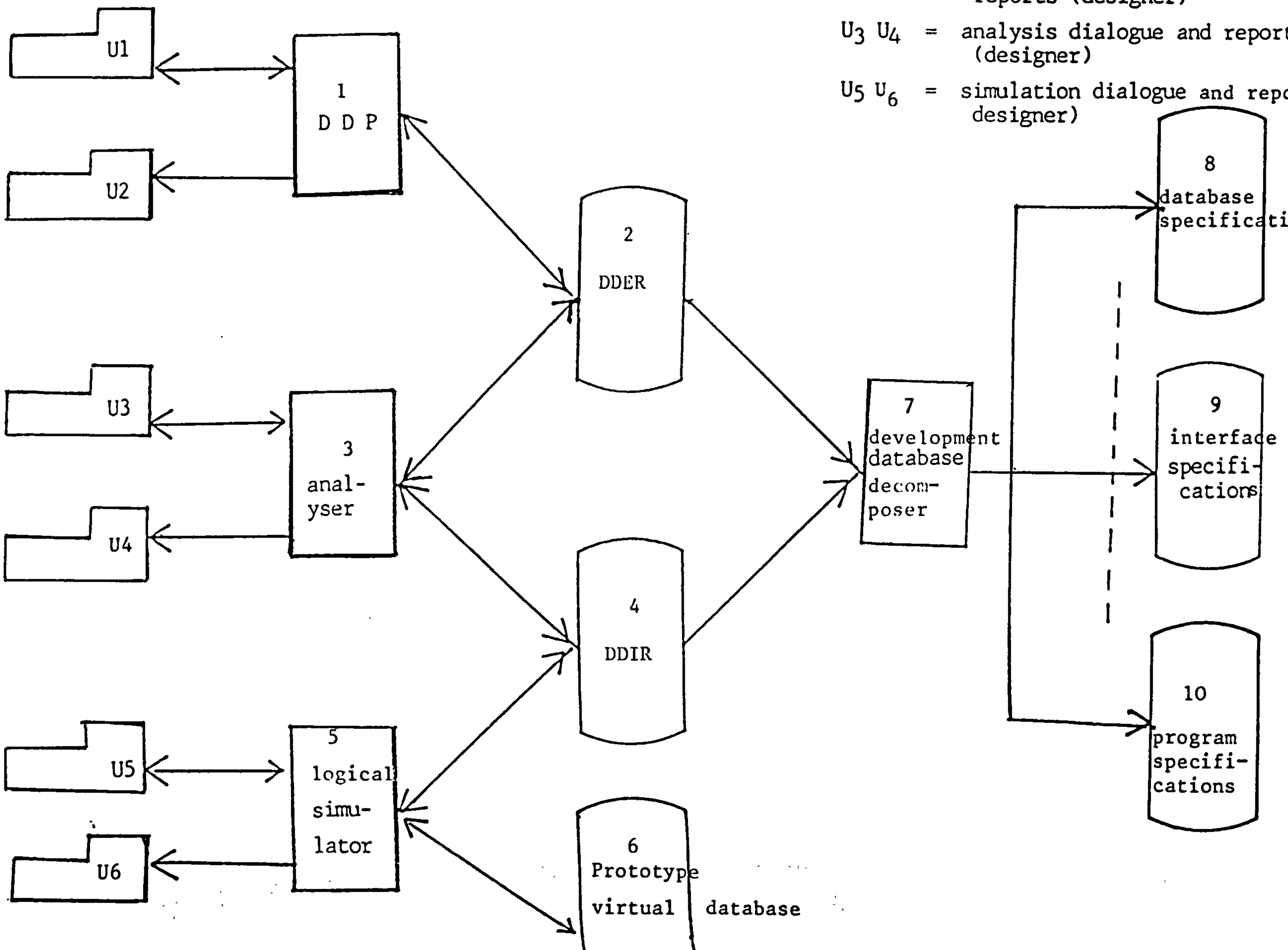



FIGURE 9_1
SOFTWARE TOOLS MODEL

## 9.2 OVERVIEW OF THE TOOLS

Figure 9-1 shows the architecture of the software "environment" offered by SSDM to the developer to support work in the logic development phase. It consists of three tools: a development dialogue processor (DDP), an analyser, and a logical simulator. Three databases are used: a development database external representation (DDER), a development database internal representation (DDIR), and a prototype virtual database. The figure shows the relationships between the tools and the databases.

A further program, the development database decomposer, is used to extract information from the DDER and DDIR and set it up for each of the three successive stages of database development, interface development and program development (see figure 7-3).

The DDP, the analyser and the logical simulator each have an interactive interface with the developer (U1, U3 and U5 respectively) as well as hard-copy output capability (U2, U4 and U6 respectively).

## 9.3 DEVELOPMENT DIALOGUE PROCESSOR (DDP)

Throughout the development process, the developer makes specification and design decisions which can be recorded in SSDL. DDP (figure 9-1 component 1) is an intelligent editor which receives SSDL statements, edits them individually for syntactic correctness, and enters them into a database called the DDER (development database external representation) (figure 9-1 component 2). If a statement appears to duplicate or contradict a statement already present in the database, DDP will report the fact.

From time to time, on request, another tool (the analyser: see section 9.4) will "compile" the DDER into a compact form suitable for analysis and verification. DDP will keep statements made since the last compilation in a separate section of the DDER, in order to minimise the analyser's recompilation task.

The DDP will offer selective display facilities. That is to say that, on request from the developer, it will display (say) all inputs, or all items, or all derived items, or all items having a given domain, or all functions triggered by a given input.

## 9.4 ANALYSER

As indicated in the previous section, the first task of the analyser (figure 9-1 component 3) is to "compile" a set of statements from the DDER into a format in which they can be added to the DDIR (development database internal representation) (figure 9-1 component 4). Figure 8-1 provides a small example of the style of the DDIR. Separate compilation of different sections of a developer's work is an important feature to eliminate redundant processing.

On completing a compilation, the analyser will on request subject the current schema to a complete verification procedure. This will include the checks discussed in section 7.3, as well as more mundane checks. Reports will be fed back to the developer, indicating (a) incompleteness - where further SSDL statements need to be made, (b) inconsistency - where apparent errors exist. These reports must be made in "source - language - compatible" form; in other words, the analyser must have access to the DDER. Further, the reports themselves need to be added to the DDER, since the developer is likely to want to refer to them frequently in subsequent periods of work. Again, the developer will be able to request selective displays from the complete set of reports.

## 9.5 LOGICAL SIMULATOR

The logical simulator (figure 9-1 component 5) offers a powerful prototyping capability. The purpose of prototyping is to show one or more users how the system will behave if it is implemented according to its current specification. At the level with which this thesis is concerned, it is appropriate to use the term logical simulation, since only the logic of the system exists to be simulated. Other forms of prototyping - eg. performance simulation - could be provided to correspond to separate concerns during development.

For a system of any size, it is likely that different users will know about different aspects of the system's required function. A prototyping session, therefore, needs to animate a part of the system corresponding to the interests and knowledge of the particular users who are observing it; and the logical simulator must be able to accept parameters which delimit the part of the system to be animated in any one run. Animation may be at two levels of detail: without data, and with data. Animation without data simply displays the sequence of steps to be performed by the system on receipt of each of the inputs which fall inside the simulation boundary. Animation with data handles sample data values and carries out computations; for this to happen, appropriate data values must be submitted to a prototype virtual database (figure 9-1 component 6), and the logical simulator will prompt for values of data items which will be required.

## 9.6 DEVELOPMENT DATABASE DECOMPOSER

When the developer believes that logic development is complete, and that detailed (physical) development is ready to begin, the development database decomposer (figure 9-1 component 7) is invoked. This simply takes both the DDER and DDIR and extracts from them the information that is required for each of three subsequent parallel activities (database development, interface development and program development) and sets up the appropriate databases (figure 9-1 components 8, 9, 10) in the format required by the different software tools.

## 9.7 CONCLUSION

The tools outlined in this chapter are regarded as the essential strategic tools to give strong support to the developer when working on the "kernel capabilities" of a system. Undoubtedly other tools will be necessary within this area, to provide an effective development environment; and further tools will be necessary to support other aspects of development, such as performance estimating and monitoring, and project management.

# CHAPTER 10

# FUTURE AND RELATED WORK

## CONTENTS

10.1 Introduction

10.2 Characteristic features for comparing methodologies

10.3 Models

10.4 Systems specification and design language

10.5 Software tools

10.6 Separate concerns

10.7 Other issues

## 10.1 INTRODUCTION

The study of existing methodologies, and the proposed model, language and software tools, are claimed to offer a necessary and sufficient basis for an improved methodology. However, due to the very scale of the topic, only the kernel of the methodology has been presented here, and there are many relevant problems which require further study. This chapter presents a summary of such problems and other related work together with some suggestions.

## 10.2 CHARACTERISTIC FEATURES FOR COMPARING METHODOLOGIES

As seen in the study of methodologies in the appendices and in chapters 3 and 4, the task of evaluating methodologies is extremely difficult. One cannot provide rigid accurate and sufficient "characteristic features" for an ideal methodology in all environments.

The description, evaluation and comparison of existing (and future) methodologies is, however, a task of great importance. Users will need to evaluate them, and to choose (and perhaps adapt) one or more of them to fit their particular needs, style of work and perceptions of problems; or alternatively consultants and academics may carry out such evaluations on their behalf. This thesis maybe seen as making two contribution in this direction, not only by carrying out forms of comparison but also in proposing desirable characteristics for models on which good methodologies might be based.

Certainly, though, this task needs to be carried further, using empirical methods to the greatest extent possible. That is to say, representative test cases, of a size and complexity which are at once manageable and challenging, should be defined, and then a range of methodologies should be applied to them. This approach would surely lead to continuing refinement of the features list, and perhaps lead to some capability for ranking features in order of significance, for identifying inter-feature conflicts, and for associating metrics with some features. It might also lead to the recognition that features differ in their relative importance for different classes of target

systems.

A further important task is to study the possible partitioning of methodologies into separate techniques or methods, which might be recombined with others to provide a good "fit" with users' needs. In this activity, the definition of the interfaces of a technique/method becomes extremely important.

## 10.3 MODELS

A classification scheme for models, and a number of models within it, have been proposed. There is a need to investigate the role and usefulness of further models (maybe in particular making use of graphic notations).

The existing model set, plus any extensions as indicated above, needs auditing by application to the same representative set of test cases suggested in section 10.2. This activity should lead to the identification and elimination of inadequacies in individual models.

Although the models proposed all relate to the domain of target systems and their development, the idea of the application of some of them, maybe with modifications, to the wider domain of the organisational environment, and to the more specialist domain of the development support system, should be investigated.

## 10.4 SYSTEMS SPECIFICATION AND DESIGN LANGUAGE

Further study of the language (SSDL) is necessary in two directions. First, extensions and changes in the models resulting from further work proposed in section 10.3 need to be incorporated. Second, only the minimum attention has been paid to the syntax of the language, which could be improved to make it more accessible to system developers.

## 10.5 SOFTWARE TOOLS

In the thesis, only a basic set of tools has been proposed, and each has been specified only in bare outline. A large amount of work needs to be done to develop each of these, at least in prototype form, and then to investigate empirically their usefulness. This will represent a major feedback loop, and can be expected to lead to revisions of models and the language, as well as to recognition of the need for additional software tools.

Eventually a major design task will be to engineer the software tools into a coherent systems development environment, with an integrated control language, to be used in an evolutionary manner for yet further empirically-based experimentation.

One particular central problem to be solved at an early stage in the development of software tools is the content and structure of the development database, in which all language statements and associated information will be stored and which will serve as the chief means of communication between individual software tools. This task is akin to the problem of IPSE database design which is currently receiving much attention.

## 10.6 SEPARATE CONCERNS

This thesis has been concerned with what might be called the "kernel capabilities" of target systems - ie. the pure logic of how they would behave in a perfect world. This aspect of a system must always be an important concern of the developer, but as a practical engineer in an imperfect world he must also pay careful attention to issues such as error detection and correction, access control, recovery from failure, concurrency control, etc. Further he must be deeply concerned with questions of performance, from the initial specification of the requirements, through progressively more detailed estimates as development proceeds, to eventual operational monitoring. Finally, any methodology, to be useful, must incorporate configuration management capabilities.

It is the contention of this thesis that these matters are properly to be regarded as separate concerns; but clearly they each represent an area of major further study. (It has been noted in chapter 3 that they receive little or no attention in existing methodologies.)

## 10.7 OTHER ISSUES

The topic of this thesis is the engineering of useful systems in the real world. Some mention has already been made in this chapter of the need for empirical research in relation to specific matters. In general, however, far too little is known of current practice, experience, intentions and problems in information systems development; and one might say that any attempt to develop methodologies in such circumstances is at best foolish. Nevertheless, the work here reported at least represents a coherent set of hypotheses, and it is hoped that anyone conducting empirical studies in this field would benefit from them as a basis for investigation.

Reference has also been made several times to the need for test-case systems to serve as experimental material, as well as to the need to develop a prototype toolset. The ideal objective would be to develop a demonstration development environment together with at least one fully developed and operational target system which would be subjected to continuing evolution.

Finally, there are two specific issues which need to be addressed. The first is prototyping, about which much is said that is glib: we need to clarify our ideas and develop our practices in this area. The second is expert systems: how do we extend our development support systems so as to take account of expert systems techniques both within target systems and within development support systems themselves?

# CHAPTER 11

# CONCLUSIONS

## CONTENTS

11.1 Review of past work

11.2 Proposals for a new approach

11.3 Final remarks

## 11.1 REVIEW OF PAST WORK

There has been a notable absence of any serious recent attempt to study the large and continually growing field of information systems design methodologies. Published studies are invariably limited in their coverage and lacking any effective systematic approach. While such a review was not the primary objective of this research project, it was seen to be an important preparatory stage; and in addition to its value in influencing the proposals for a new approach, it seemed to be an academic task worth carrying out for its own sake, and worth the effort of seeking some improved descriptive framework.

The original elements claimed in this first part of the thesis are as follows.

(1) The historical review is simple and straightforward. The causality implied in the idea of the two-generation "sophistication lag" from hardware to methodologies is intuitively appealing.

(2) The "detailed summary" of six leading methodologies is a means of comparison which has not been attempted elsewhere, and which has proved illuminating. It would be valuable to extend its scope both in terms of the individual summaries and of the number of methodologies covered.

(3) The comparative survey of methodologies breaks new ground both in the number of methodologies included and in the feature set used for their description. It is far more comprehensive than

any other survey, and concentrates more on essential features.

(4) The classification of the various approaches which have been adopted by methodology originators, while not claiming completeness, casts a new light on the diversity of view points, both technical and organisational, that a serious worker in this field has to encompass.

(5) The grouping of individual techniques is also regarded as an original and illuminating contribution.

## 11.2 PROPOSALS FOR A NEW APPROACH

It should be apparent from the review of past work that the large majority of proposed methodologies have not been well-founded on a sound theoretical basis, itself derived from a thoughtful analysis of requirements. While methodologies are certainly practical tools, intended for use by practical people to solve practical problems, it is a mistake to suppose that they can be successfully designed in an ad hoc manner. The size and complexity of the problems with which methodologies are supposed to be of assistance is such that good theoretical foundations are essential.

The original elements in the second part of the thesis are as follows.

(1) The proposals are based on a comprehensive but simple architecture, involving (a) levels of abstraction (models, notations, tools), (b) separation of concerns (pure logic, performance, error handling, etc.). The very large scale of the proposed enterprise inevitably dictated that work should be confined to a small part of the whole - primarily to the level of models and to the pure logic of target systems. Enough is said, however, about other levels and concerns to indicate the viability and power of the architecture.

(2) The highest-level model presents a picture of broad categories of systems within an organisation, and of the "worlds" inhabited by systems developers and others, which is richer and more realistic than other models of this kind - which are in

any case rarely offered. A deep understanding of this organisational context is an essential starting-point for the development of methodologies.

(3) The models of the development process represent a major step forward from the traditional life-cycle ("waterfall") family of models, which is becomming increasingly discredited. Their most notable features are (a) the emphasis on the interrelatedness of development and operational activities; (b) the success of the more detailed model in integrating analysis, specification, design and verification within a single "canonical step"; (c) the definition of clear interfaces between requirements development, logic development and detailed physical development; (d) the idea of developing the logic of the whole system, and verifying it, before embarking on subsystem decomposition.

(4) The product system models are original in their identification of a small number of classes of functions, and their expression of the structure of systems using set notation. The translation from these models into a corresponding language (SSDL) is easy and straightforward. The models have received some (albeit very limited) empirical testing.

(5) The proposed classes of verification checks are also an original contribution. In particular, the so-called "derivation dependency check" has as far as is known never been proposed before.

(6) The proposals obviously owe a primary debt to the work of GRINDLEY in Systematics. Major improvements on his work are (a) the virtual database, (b) the technique of propagating triggers to individual processes to facilitate trigger consistency checking, (c) the introduction of filter functions.

## 11.3 FINAL REMARKS

In sum, it is claimed that this thesis, while leaving large areas of its subject matter unaddressed, presents an original and successful overall approach, and many original and successful ideas within the limited area which it is has been possible to develop in detail. An extensive programme of further work has been mapped out. It is worth noting that one of the areas of detailed physical development, which is designed to interface to the logic development phase - that of database design - is the subject of a parallel piece of work carried out by my colleague R P Whittington, whose thesis was recently successfully presented. I very much hope that other workers will now take up more of the problems identified earlier in this thesis, and that they will find the work as challenging and rewarding as I have done.

## REFERENCES


ACKOFF R L, "Towards a System of Systems Concepts", in Couger and Knapp (op cit, 1974).

AFIPS (American Federation of Information Processing Societies), "Proceedings of Fall Joint Computer Conference 1972" (1972).

ALFORD M W, "A Requirement Engineering Methodology for Real Time Processing Requirements" IEEE Transactions on Software Engineering Vol.SE-3 No.1 (January 1977).

ALLOWAY R M and Quillard, "User Managers Systems Needs", CISR WP 86, Massachusetts Institute of Technology (1982).

ARNON J and others, "Software Documentation - an Automated Approach, Trends and Applications", IEEE (May.1982).

ASCHIM F and Mostue, "Sysdoc and Systemater",in Olle (op cit, 1982).

ASPROTH V and Hakansson, "A Declaration as a means to compare Information Systems Design Methodologies", unpublished paper (1983).

ASWE (Admiralty Surface Weapons Establishment), "MASCOT Operating System Manual",(1979).

ATWOOD J W, "The Systems Analyst", Hayden (1977).

BATINI C and Lenzerini, "A Computer Aided Methodology for Conceptual Data Base Design", Information Systems Vol.7, No.1, (January 1982).

BCS (British Computer Society), "Data Dictionary Systems Working Party Report" (1977).

BENCI E and others, "Concepts for the Design of Conceptual Schema", in Nijssen (op cit, 1976).

BENN R and others, "Data Base Driven System Design", in Cotterman and others (op cit, 1981).

BERNSTEIN P A, "Synthesizing Third Normal Form Relations from Functional Dependencies", ACM Transactions on Database Systems, Vol.1, No 4 (December 1976).

BILLER H, "Concepts of Conceptual Schema", in Nijssen (op cit, 1977).

BIRKS E, "Requirements Analysis and Specifications", in Cotterman and others (op cit, 1981).

BODART F and others, "Evaluation of CRIS 1 Methodologies Using Three Cycles Framework", in Olle (op cit, 1983).

BOEHM B, "Software Engineering", IEEE Transactions on Computers, Vol c-25, No.12 (1976).

BOEHM B W, "Software Engineering Economics", Prentice-Hall (1981).

BOOT R (editor), "Computers and the Professionals" (Proceedings of the Workshop on Approaches to Systems Design), NCC (1973).

BOSMAN A and others, "Evolutionary Development of Information Systems", in Hawgood (op cit, 1982).

BOWERS R J, "Structure and Flowcharts : a Suggested Notation ", Computer Bulletin (December 1981).

BRACCHI G and others, "Constraints specification in Evolutionary Database Design",in Schneider (op cit, 1979).

BRADLEY J, "File and Data Base Techniques", Holt, Rinehart and Winston (1981).

BRAMER M A, "A Survey and Critical Review of Expert Systems Research", BCS (1981).

BRANDT I, "A Comparative Study of Information System Design Methodologies", in Olle (op cit, 1983).

BREWER T, "Improving Systems Productivity", Butler Cox Foundation Report No.11 (1979).

BRITTAN J N G, "Design for Changing Environment", Computer Journal, Vol.23 No.1 (Feburary 1980).

BRODIE M L and Silva, "Active and Passive Component Modelling - ACM/PCM",in Olle (op cit, 1982).

BROWN J L, "Logical Design of Computer Based Information Systems", in Cotterman and others (op cit, 1981).

BUBENKO J A, Langefors, Solvberg (Eds), "Computer Aided Information Systems Analysis and Design", Studentlitteratur (1971).

BUBENKO J A, and others, "IAM : An Inferential Abstract Modelling

Approach to Design of Conceptual Schema", in Smith (op cit, 1977).

BUBENKO J A, "Information Modelling in the Context of System Development", in Lavington (op cit, 1980).

BUCHMANN A P and Dale, "Evaluation Criteria for Logical Database Design Methodologies", Computer aided Design Vol.11 No.3 (May 1979).

CAMPAGNE J P and others, "Introduction of Change in Data Systems within an Organisation : the pre-design stage", in Hawgood (op cit, 1982).

CARLSON W M, "BIAIT : Business Information Analysis and Integration Technique" IBM - San Jose, CA ACM-80, (October, 1980).

CHAMBERLIN D D, "Relational Data-Base Systems", Computer Surveys Vol.8, No.1, (March 1976).

CHAPIN N, "Flow Charting with ANSI Standards : a tutorial", Computing Surveys Vol.2, No.2 (1970).

CHAPIN N, "Graphic Tools in the Design of Information Systems", in Cotterman and others (op cit, 1981a).

CHAPIN N, "Structured Analysis and Design : an overview", in Cotterman and others (op cit, 1981b).

CHECKLAND P, "Systems Thinking, Systems Practice", Wiley (1981).

CHEN P, "The Entity Relationship Model- Toward a Unified View of Data", ACM Transactions on Database Systems, Vol.1, No.1, (March

1976).

CHEN P (editor), "Entity Relationship Approach to Systems Analysis and Design" (Proceedings Conference at Los Angeles; 1979), North-Holland (1980).

CODASYL, "An Information Algebra phase 1 report", Communications of the ACM Vol.5, No.4, (April 1962).

CODASYL, "DBTG Report" (1969).

CODD E F, "Relational Database : A Practical Foundation for Productivity" (The 1981 ACM Turing Award Lecture), Communications of ACM, Vol.25, No.2, (February 1982).

CORNER M, "Structured Analysis and Design Technique", in Cotterman and others (op cit, 1981).

Cotterman W W and others, "Systems Analysis and Design : a Foundation for the 1980s", North-Holland (1981).

COUGER J D, "Evolution of Business System Analysis Techniques", Computing Surveys Vol.5, No.3 (September 1973).

COUGER J D and Knapp, "System Analysis Techniques", Wiley (1974).

DARLINGTON J, Henderson and Turner (editors), "Functional Programming and its Applications", Cambridge University Press (1983).

DATE C J, "An Introduction to Database Systems" (3rd edition), Addison-Wesley (1981).

DATE C J, "Null Values in Database Management", in Deen and Hammersley (op cit, 1982).

DATE C J, "An Introduction to Database Systems - Volume 2", Addison-Wesley (1983).

DAVENPORT R A, "Data Analysis for Database Design", Australian Computer Journal, Vol.10 No.4 (November 1978).

DEARNLEY P A and others, "In Favour of System Prototypes and their Integration into the System Development Cycle", Computer Journal Vol.26, No.1 (Feburary 1983).

DEEN S M and Hammersley, "British National Conference on Databases", (Proceedings First British National Conference, July 1981), University of Cambridge (1981).

DEEN S M and others, "The Design of a Canonical Database System (PRECI)", Computer Journal Vol.24 No.3 (1981).

DEEN S M and Hammersley, "British National Conference on Databases", (Proceedings Second British National Conference, July 1982), Bristol University (1982).

DEMARCO T, "Structured Analysis and System Specification", Prentice-Hall (1979).

DOBOSZ J and Symanski, "An Implementation of Relational Interface to an Information Retrieval System", Information Systems Vol.6 No.3 (June 1981).

DoI (Department of Industry), "Report on the Study of an Ada-based Systems Development Methodology" (2 volumes) (1981).

DOLOTTA T A and others, "Data Processing in 1980-1985", Wiley (1976).

DOWNES V and Goldsack, "Programming Embedded Systems with Ada", Prentice-Hall (1982).

EHRENSBERGER M, "Data Dictionary - More on the impossible dream", in Gilchrist (op cit, 1977).

FALKENBERG E, "Concepts for Modelling Information", in Nijssen (op cit, 1976).

FALKENBERG E and Nijssen,"Feature Analysis of ACM/PCM, CIAM, ISAC, NIAM", in Olle (op cit, 1983).

FALLA M E, "An Introduction to Gamma System", Software Sciences Ltd.(1981).

FERGUS R, "Decision Tables, What, Why, How", in Couger and Knapp (op cit, 1974).

FLAVIN M, "Fundamental Concepts of Information Modelling", Yourdon Press (1981).

FRANK W A and others, "The Integrated Dictionary / Directory System", Computer Surveys, Vol.14, No.2 (June 1982).

FREEMAN M, "Software Design Representation : Analyses and Improvements", Software Practice and Experience, Vol.8, (1978).

FREEMAN P and Wasserman, "Report on Software Development Methodologies and Ada", University of California, San Francisco (1982).

FRY J P and Sibley, "Evolution of Database Management Systems", Computing Surveys Vol.8, No.1 (March 1976).

FURUTA R and others, "Document Formatting Systems : Survey, concepts and Issues", Computer Surveys, Vol.14, No.3, (September 1982).

GANE C and Sarson, "Structured Systems Analysis : Tools and Techniques", Prentice-Hall (1979).

GILBERT M H, "Functional Decomposition of Real Time Systems", Computer Science and Systems Division AERE Harwell (1982).

GILCHRIST B N (editor), "Information Processing 1977" (Proceedings of the IFIP Congress, August 1977), North-Holland (1977).

GILDERSLEEVE T R, "Successful Data Processing Systems Analysis" Prentice-Hall (1978).

GLAGOWSKI T G and White, "A Relational View of a Software Design Model"IEEE (1978).

GLASSON B C and Hodgson, "Information System Design Methodologies: an Analysis of Scope", unpublished paper (1983).

GRINDLEY C B B, "Systematics : a Non-programming Language for Designing and Specifying Commercial Systems for Computers", Computer Journal Vol.9, No.3 (1966).

GRINDLEY C B B, "A Language for Describing Formal Management Systems" (Ph.D Thesis), London School of Economics (1972).

GRINDLEY K, "Systematics : a New Approach to Systems Analysis", McGraw-Hill (1975).

GRINDLEY C B B, "The Role of the Trigger in Systematics", in Schneider (op cit, 1979).

GROCHLA E and Szyperski, "Information Systems and Organisational Structure", Walter de Gruyter (1975).

GROIENHUIS G and Brock, "A Conceptual Model for Information Processing", in Nijssen (op cit, 1976).

GUSTAFSSON M R and others, "A Declarative Approach to Conceptual Information Modelling : CIM", in Olle (op cit, 1982).

HALL P and Todd, "Relations and Entities", in Nijssen (op cit, 1976).

HALL J, "LBMS-SDM : Learmonth and Burchett Management Systems - System Methodology", LBMS London (1981).

HAMILTON M and Zeldin, "A Relationship between Design and Verification", The Journal of Systems and Software (1979).

HAMMER M and McLeod, "Database Description with SDM : A Semantic Database Model", ACM Transactions on Database Systems, Vol.6 No.3 (Sept 1981).

HANNAFORD D, "The BIS Data Analysis Methodology", Database Journal

Vol.10, NO.4 (1980).

HANSEN J V and Mekell, "A Computer Aid for Analysis of Complex Systems", Computer Journal, Vol.23 No.2 (May 1980).

HARTMAN W and others, "Grid-charting", in Couger and Knapp (op cit, 1974).

HAWGOOD J (editor), "Evolutionary Information Systems" (Proceedings of the IFIP TC-8 Conference September 1981), North-Holland (1982).

HERSHEY E A, "A Survey of Systems Design Aids", in Teichroew (op cit, 1977).

HICE G F and others, "System Development Methodology", revised edition, North-Holland (1978).

HIGHER ORDER SOFTWARE, "Conceptual Description ISDS/HOS" (TR - 3) (1977).

HIGHER ORDER SOFTWARE, "AXES SYNTAX DESCRIPTION" (TR - 4) (1976).

HIGHER ORDER SOFTWARE, "Verification of an Axiomatic Requirement Specification" (TR - 10) (1977).

HIGHER ORDER SOFTWARE, "Algebraic Specification of Data Types in HOS" (TR - 13) (1978 ).

HIGHER ORDER SOFTWARE, "Axiomatic Methodology Requirements" (TR - 14) (1978).

HIGHER ORDER SOFTWARE, "Properties of User Requirments" (TR - 20)

(1978).

HIGHER ORDER SOFTWARE, "The Relationship between Design and Verification" (TR - 21) (1978).

HONEYWELL, "Business System Analysis and Design : BISAD", in Couger and Knapp (op cit, 1974).

HUBBARD G U, "Computer Assisted Logical Database Design", Computer Aided Design, Vol.11 No.3 (May 1979).

HUTT A T F, "A Relational Data Base Management System", Wiley (1979).

IBM "SOP : Study Organisation Plan Documentation Techniques", in Couger and Knapp (op cit, 1974a).

IBM "TAG : The Time Automated Grid", in Couger and Knapp (op cit, 1974b).

INFOTECH, "Info Software : proceedings", (1975).

INFOTECH, "State of the Art Report: Data Base technology", Volume 2 (1978).

INFOTECH, "Data Design" (2 volumes) (1980).

ISHIKETA T and Yokoyama, "A Managerial Decision Making Tool - Computer Assisted Problem Solving System (CAPSS)", IFIP Congress Proceedings (1971).

ISO (International Standards Organisation), "Concepts and Terminology for the Conceptual Schema and the Information Base" ( Report of TC

97/SC5/WG3), edited Griethuysen (1982).

JACKSON M, "Principles of Program Design", Academic Press (1975).

JACKSON M A, "some Principles Underlying a Systems Development Method" in Cotterman and others (op cit, 1981).

JACKSON M, "System Development", Prentice-Hall (1983).

JOHN D, "Project Management of Systems Management", in Cotterman and others (op cit, 1981).

JOHN P and others, "Ethical and social Responsibilities of the Systems Analyst", in Cotterman and others (op cit, 1981).

JOHNSON R G and Prowse, "A Natural Language Database Interface to User" Computer Journal Vol.23 No.1 (Feburary 1980).

JONES M P, "The Practical Guide to Structured Systems Design", Yourdon Press (1980).

KAHN B K, "A Method for Describing Information Required by the Database Design Process", in Rothnie (op cit, 1976).

KANTER J, "Management-oriented Management Information Systems", Prentice-Hall (1979).

KEHA V, "GEIS : Gradual Evolution of Information Systems" in Freeman and Wasserman (op cit, 1982).

KENT W, "Entities and Relationships in Information", in Nijssen (op cit, 1977).

KENT W, "Data and Reality", North-Holland (1978).

KENT W, "Future Requirements on Data Modelling", in Shaw (op cit, 1979).

KENT W, "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM, Vol.26, No.2 (Feburary 1983).

KEROLA P and Taggart, "Human Information Processing Style in the Information Systems Development Process", in Hawgood (op cit, 1982).

KING P J H, "Conversion of Decision Tables to Computer Programs by Rule Mask Techniques" Communications of the ACM, Vol.9, p796-901 (1966).

KING P J H, "Some comment on Systematics "Computer Journal, Vol. 10, p.116 (1967a).

KING P J H, "Decision Tables", Computer Journal, Vol.10, p135-141, (1967b).

KNUTH and others, "SDLA : System Descripter and Logical Analyser", in Olle (op cit, 1982).

KUNG C H, "An Analysis of Three Conceptual Models", in Olle (op cit, 1983).

LAAGLAND P T J, "PRISMA : Planning and Requirement Analysis for Information Systems, Modelling Approach", unpublished paper (1982).

LAND F F, "Concepts and Perceptions - a review", in Hawgood (op cit,

1982).

LANGEFORS B, "Computation of Parts requirements for production Scheduling", BIT BIND 2, Hefte, No.2 (1962).

LANGEFORS B, "Theoretical Analysis of Information Systems" (4th edition), Auerbach (1973).

LANGEFORS B, "Theoretical Constructs of Information Processes", in Cotterman and others (op cit, 1981).

LANGEFORS B, "Models and Methodologies", in Hawgood (op cit, 1982).

LARCHER P, "Transaction Analysis Technique (User Guide)", The Plessey Company Ltd. (1980).

LAVINGTON S H, "Proceedings Information Processing 1980's", North-Holland (1980).

LEHMAN M M, "Program Evolution", Research Report, Imperial College London, (December 1982).

LEONG-HONG B W, "Data Dictionary/Directory System", Wiley (1982).

LOBELL R F, "Application Program Generators", NCC (1983).

LOCKEMANN P C and Neuhold (editors), "Systems for Large Data Bases" (IFIP Working Conference on Very Large Data Bases, September 1976), North-Holland (1977).

LOMAX J D, "Data Dictionary Systems", NCC (1977).

LUCAS and others (editors), "The Information System Environment" (Proceedings of the IFIP TC8 Conference, June 1979), North-Holland (1980).

LUDEWIG J and Streng, "Methods and Tools for Software Specification and Design - A Survey Report", European Purdue Workshop (1978).

LUNDEBERG M, "Utilization of New Information Systems Development Methods in Practice - Perspectives and Prospects", in Gilchrist (op cit, 1977).

LUNDEBERG M and others, "Information Systems Development", Prentice-Hall (1981).

LUNDEBERG M, "The ISAC approach to Specification of Information Systems and it's Applications", in Olle (op cit, 1982).

LYNCH H J, "ADS : A Technique in System Documentation", in Couger and Knapp (op cit, 1974).

MACDONALD I G and Palmer,"D2S2 : System Development in Shared Environment", in Olle (op cit, 1982).

MACDONALD I G, "What Must We Look for in a Systems Development Methodology", The DMW group, unpublished paper (1983).

MACLEOD I A, "The Relational Model as a Basis for Document Retrieval System Design" Computer Journal Vol. 24 No 4 (1981).

MADDISON R N and others, "Feature Analysis of Contemporary System Methodologies : a collective view", unpublished paper (1982).

MALMBORG E G, "An Analysis of Systems Design Methodology Using ISO Frame Work",in Olle (op cit, 1983).

MARTIN J, "Principles of Database Management", Prentice-Hall (1976).

MARTIN J, "An End User Guide to Data Base", Savant Research Institute (1980).

MARTIN J, "Information System Manifesto", Savant, Carnforth, Lancashire (1983).

McCRACKEN D M, "A Guide to NOMAD for Applications Development", Addison-Wesley (1980).

McCRACKEN D and others, "A Minority Dissenting Position", in Cotterman and others (op cit, 1981).

MICHAEL F, "An Integrated View of Computer Software Application Development Life Cycle", in Cotterman and others (op cit, 1981).

MILLINGTON D, "Systems Analysis and Design for Computer Applications", Wiley (1981).

MOULIN P and others, "Conceptual Model as a Data Base Design Tool", in Nijssen (op cit, 1976).

MOULIN B, "A Comparative Review of Information System Design Methodologies Using EPAS/IPSO Approach", in Olle (op cit, 1983).

MUMFORD E and Weir, "Computer Systems in Work Design - the ETHICS Method", Associated Business Press (1979).

MURDICK R, "MIS Development Procedures", Journal of Systems Management (December 1970).

MYERS G J, "Computer Structured Design", Van Nostrand Reinhold (1978).

NCC (National Computing Centre), "A system documented - in accordance with the standards in the systems documentation manual", (1971).

NCC (National Computing Centre), "Introducing System Analysis and Design" (2 volumes) (1982).

NIJSSEN G M (editor), "Architecture and Models in Data Base Management Systems" (IFIP Working Conference on Modelling in Data Base Management Systems), North-Holland (1976).

NIJSSEN G M (editor), "Modelling in Data Base Management Systems" (IFIP Working Conference on Modelling in Data Base Management Systems)", North-Holland (1977).

NIJSSEN G M, "Current Issues in Conceptual Schema", in Nijssen (op cit, 1977).

NISSEN H E, "When People Design Information System - Then Information System Design People", in Hawgood (op cit, 1982).

MULLERY G P, "CORE : A Method for Controlled Requirement Expressions", Systems Designers Ltd.(1979).

NORMAN L and others, "Classical and Structured Systems Life-cycle Phases", in Cotterman and others (op cit, 1981).

NUNAMAKER J F, "A Methodology for the Design and Optimisation of Information Processing Systems", in Couger and Knapp (op cit, 1974).

NUNAMAKER J P and others, "Formal and Automated Techniques of Systems Analysis and Design", in Cotterman and others (op cit, 1981).

OLIVE A, "DADES : A Methodology for Specification and Design of Information Systems", in Olle (op cit, 1982).

OLIVE, "Analysis of Conceptual and Logical Models in Information Systems Design Methodologies", in Olle (op cit, 1983).

OLIVE A, "Information Derivability Analysis in Logical Information Systems", Communications of the acm, Vol.26, Number 11, November (1983).

OLLE T W and others (editors), "Feature Analysis of Information System Design Methodologies" (Proceedings of the working Conference, June 1982), North-Holland (1982).

OLLE T W and others (editors), "Feature Analysis of Information System Design Methodologies" (Proceedings of the working Conference, July 1983), North-Holland (1983).

OSAMU S, and others, "A Software Design System Based on a Unified Design Methodology", Journal of Information Processing, Vol. 3, Number 3, (1980).

OSTERWEIL L J and others,"ASSET : A Life Cycle Verification and Visibility System", Journal of Systems and Software (1979).

PARNAS D L, "The Use of Precise Specifications in the Development of Software", in Gilcrest (op cit, 1977).

PARSLOW R D (editor), "Information Technology for the Eighties" (Proceedings of BCS '81, July 1981), Heyden (1981).

PAUL W, "Privacy, Security, and Auditing of Automated Systems" in Cotterman and others (op cit, 1981).

PAWLAK Z, "Information Systems Theoretical Foundations", Information Systems Vol.6, No.3 (1981).

PETERSON J L, "Petri Nets", Computer Surveys, Vol.9, No.3 (1977).

POLANYI M, "Knowledge and Being", Routledge and Kegan Paul (1969).

POLLACK S and others, "The Languages used in Decision Tables", in Couger and Knapp (op cit, 1974).

PROWSE P, "The Data Base Approach", Computer Journal, Vol.23 No.1 (Feburary 1980).

PRYWES N S and others, "Use of a Non-Procedural Specification Language and Associated Program", ACM Transactions on programming languages and systems, Vol.1 No.2 (October 1979) pp 196-217.

REINWALD L.T and others, "Conversion of Limited-entry Decision Tables to Optimal Computer Programs : Minimum Average Processing Time"Journal ACM No.3, Vol.13 pp 339-358 (July 1966).

REISNER P, "Human Factors Studies of Database Query Languages : A

Survey and Assessment", Computer Surveys Vol.13, No.1 (March 1981).

RHODES J, "Beyond Programming : Practical steps toward the Automation of D.P Systems Creation", in Couger and Knapp (op cit, 1974).

RICHTER G and Durchholz, "IML-inscribed nets", in Olle (op cit, 1982).

ROCK-EVANS, "Data Analysis", IPC Business Press (1981).

ROLLAND C, RICHARD C, "The REMORA Methodology for Information Systems Design and Management", in Olle (op cit, 1982).

ROMAN G C, "On Reducing Ambiguities in Methodology Definitions", IEEE (1982).

RONALD F, "A Simplified Universal Relation Assumption and its Properties", ACM Transactions Database Systems, Vol.7, No.3, (September 1982).

ROSENQUIST C J, "Entity Life Cycle Models and their Applicability to Information Systems Development Life Cycles", Computer Journal, Vol.25, No.3, (1982).

ROSS R G, "Data Base Systems", AMACOM (1978).

ROTHNIE J R (editor), "Data Modelling" (SIGMOD Proceedings of the Working Conference June 2-4, 1976), ACM (1976).

ROUSSOPOULOS N, "Tools for Designing Conceptual Schemata of Databases", Computer Aided Design, Vol.11 No.3 (May 1979).

RZEVSKI G and others, "The Evolutionary Design Methodology applied to

Information Systems", in Olle (op cit, 1982).

SASS C J, "COBOL Programming and Applications", Allyn and Bacon (1979).

RZEVSKI and others, "A Recursive Approach to the Comparison of Systems and Software Methodologies", Kingston Polytechnic Report (1983).

SCHMID H A, "An Analysis of some Constructs for Conceptual Models", in Nijssen (op cit, 1977).

SCHNEIDER H J (editor), "Formal Models and Practical Tools for Information Systems Design", North-Holland (1979).

SDL (Systems Designers Ltd), "Life Cycle Support in the Ada environment" (1982).

SENKO E, and Michael E, "Information Systems : Records, Relations, Sets, Entities, and Things", Information System Vol.1 (1975).

SERNADAS A, "Temporal Aspects of Logical Procedure Definition", Information Systems Vol.5 (1980).

SERNADAS "Systematics : its syntax and semantics as a query language", Computer Journal, Vol.24 No.1 and 2 (1981).

SHAW B (editor), "Proceedings of the Joint IBM/University of Newcastle Seminar", University of Newcastle (1979).

SHIGO OSAMU and others, "A Software Design System Based on a Unified Design Methodology", Journal of Information Processing, Vol.3, No.3

(1980).

SMITH D C P (editor), "ACM SIGMOD : Management of Data", (Proceedings International Conference, (August 1977).

SMITH J and Smith , "Data base Abstractions, Aggregation", Communicatin ACM (June 1977a).

SMITH J and Smith, "Database Abstractions: aggregation, and Generalisation", TODS (June 1977b).

SMITH D C P and Smith, "Computerised Database Design", Infotech State of the Art, series 8, No.4, (1980).

SOLVBERG A, "A Draft Proposal for Integrating Systems Models", in Olle (op cit, 1982a).

SOLVBERG A, "Introduction to Workshop Session on Methods and Models for Evolutionary Information Systems", in Hawgood (op cit, 1982b).

STAMPER R, "Physical Objects, Human Discourse, and Formal Systems", in Nijssen (op cit, 1977).

STEYER F, "A Uniform Formal Description of Data Base Management Systems"Information Systems Vol.5 (1980).

STL (Standard Telecommunication Laboratories), "Formal Design Methodologies" (Proceedings of the Symposium, April 1979) (1979)..

STRUNZ M, "A Review of the Current State of the Art", Boot (op cit, 1973).

SWANSON E B, "A View of Information System Evolution", in Hawgood (op cit, 1982).

SWIGCHEM C VAN and Essink, "Systems Design and Evaluation Criteria", in Olle (op cit, 1983).

TAGGART W M (Jr) and Tharp M O, "A Survey of Information Requirements Analysis Techniques", Computing Surveys, Vol.9 No.4 (Dec 1977).

TEICHROEW D and Hasan, "Automation of System Building", Datamation (August 1971).

TEICHROEW D, "A Survey of Languages for Stating Requirements for Computer Based Information Systems", Fall Joint Computer Conference (1972).

TEICHROEW D, "Problem Statement Analysis : requirements for the problem statement analyser", in Couger and Knapp (op cit, 1974a).

TEICHROEW D, "Problem Statement Languages in MIS", in Couger and Knapp (op cit, 1974b).

TEICHROEW D and others, "An Introduction to Computer Aided Documentation of User Requirements for Computer Based Processing Systems", in Grochla and Szyperski (op cit, 1975).

TEICHROEW D, "Computer-aided Software Development", Infotech (1977).

TEICHROEW D, "Improvements in the Systems Life Cycle", in Teichroew (op cit, 1977a).

TEICHROEW D, and Hershy, "Computer Aided Software", in Teichroew (op cit, 1977b).

TEICHROEW and Hershy, "Computer Aided Structered Documentation, and Analysis of Information Processing Requirements-ISDOS working paper", in Teichroew (op cit, 1977c).

THOMAS R J and Kirkham, "MICRO-PSL Project Report, University of Bradford (1983).

THURNER R, "Systems Development Technology, A Management Concern", in Proceedings of Butler cox Foundation Conference (October 1978).

TSE T H and Pong, "A review of Systems Development Systems", University of Hong kong (1982).

TULLY C J, "Definition of a Language for System Specification and Design and a comparison with other approaches", paper presented in IKD Berlin (1982).

TURN R, "Computers in the 1980s", Columbia University Press (1974).

ULLMAN J D, "Principles of Database Systems", Computer Science Press (1980).

VERHEIJEN G M A and others " NIAM Methodology", in Olle (op cit, 1982).

WALTER M and Carlson, "Making your Work Effective", in Cotterman and others (op cit, 1981).

WARNIER J D, "Logical Construction of Systems", Van Nostrand Reinhold (1981).

WASSERMAN A I, "A Methodology for the Design and Development of Interactive Information Systems", in Olle (op cit, 1982).

WASSERMAN A I, Freeman and others, "Characteristics of Software Development Methodologies", in Olle (op cit, 1983).

WELDON J L, "Using Data Base Abstractions for Logical Design", The Computer Journal, Vol.23 No. 1 (Feburary 1980).

WHITTINGTON R P and Tully, "A Seven-Subschema Model for Evolutionary Database Development", in Deen and Hammersley (op cit, 1982).

WHITTINGTON R P, "A Comprehensive and Flexible Methodology for the Development of Effective Database Systems", D.Phil thesis, Department of Computer Science University of York (1983).

WILLIAMS M H, "A Flexible Notation for Syntactic Definitions", ACM Transactions Vol.4 No.1 (January 1982).

WINDSOR A, (editor), "Using the ICL Data Dictionary", Shiva (1980).

WIRTH N, "Algorithms + Data structures = Programs", Prentice-Hall (1976).

WOOD-HARPER A T, and Flynn, "Action Learning for Teaching Information Systems BCS Computer Journal Vol.26, No.1, (Feburary 1983).

YEH R T and others, "Database Design - an approach and some issues" in

INFOTECH (op cit, 1978).

YOUNG J W and Kent, "Abstract Formulation of Data Processing problems", Journal of Industrial Engineering (November 1958).

YOURDON E and Constantine, "Structured Design : Fundamentals of a Discipline of Computer Program and System Design", Prentice-Hall (1979).

# APPENDIX-A

## THE FEATURE LIST ADOPTED FOR THE PRESENT COMPARATIVE SURVEY

The starting point was the union of features from the comparative surveys in section 3.2 of chapter 3. This was both unnecessarily large and insufficiently embracing. The following set of features aims to be necessary and sufficient to present an essential description of methodologies for the purpose of evaluation and comparison.

### PART 1: METHODOLOGY SUMMARY

| | |
|---|---|
| Short name: | acronym; if developer does not provide one then create one. |
| Full name: | full name as given by the developer. |
| Author(s) and institution(s): | developer/organisation identification. |
| Date of first reference: | date of first reference on the basis of available literature. |
| Application field(s): | the main types of application to which the methodology is relevant, from the following set: data processing, defence, embedded, systems software, telecommunications. |

Life cycle stages: the main life cycle stages in which the methodology may be applicable, from the following set: requirements analysis, outline specification, functional specification, structural design, detailed design, programming.

(The above six stages are a fairly arbitrary generalisation from the many variants of the life-cycle model. They are intended to be readily understandable to the reader. They do not match the life-cycle model (if any) of any individual methodology; nor do they match the model of the system development process proposed later in this thesis.)

Requirements analysis This involves the answers to three questions, which are: what is the new system required to do? within what constraints must the new system operate? how is the new system's performance evaluated? The typical output is a requirements definition for the new system agreed by all parties.

Boundary specification This is a precise

set of all anticipated outputs and inputs of the proposed system followed by comments, if any.

Functional specification This is the process of defining the functions that are necessary in order to derive the required outputs from the available inputs, together with the relationships between them.

Structural design This is an iterative process of: decomposition of a system into subsystems, boundary specification of subsystems, functional specification of subsystems, decomposition into components, analysis into elements.

Detailed design This is a description of how the system is to achieve its specifications i.e the selection of algorithms, data structures and equipment that will fulfill the system functions.

Programming

Software support: provision of computer aids, from the following set: data dictionary, analysis

and checking, detailed design aids, prototyping, code generation.

(The above six types of software support are not intended to be an exhaustive set, but rather to cover the main observed area in which software tools are currently offered. Explanatory comments are only needed for two of them.)

Analysis and checking This is a mechanism for checking automatically the completeness and consistency of a specification or design at semantic and syntactic levels.

Detailed design aids These are tools and techniques which automatically generate supporting documents and messages to aid the developer in developing his target system.

Development status: one of the following set (with variants and comments): published but not used, used but obsolete, in use, in use and still under development, under development.

Comment: brief notes on all the important aspects of part 1, and any other relevant remarks.

PART 2: LIFE CYCLE MODEL — A brief description of the major stages prescribed for target system development.

PART 3: ENVIRONMENT MODEL

Concepts used to describe the system environment: description of conceptual entities and constraints used to describe the Universe of Discourse including entities, objects, events, triggers, functions, relationships etc.

Notation used: types of notation used to describe the environment model (e.g. textual, graphical, mathematical, etc.)

PART 4: SYSTEM MODEL

Concepts used in specification and design: description of conceptual entities and constraints used to describe target systems.

Notation used: The types of notation used to describe

the system model (eg. textual, graphical, mathematical, etc.)

PART 5: COMMENT

Completeness:

Economy:

Ease of use:

Additional comments:

PART 6: REFERENCES

# APPENDIX B

# SURVEY OF METHODOLOGIES

## INTRODUCTORY NOTE

This appendix presents a survey of fortythree methodologies. The survey is based on the list of the features described in appendix A. The main headings for each methodology are numbered (after the decimal point) to correspond with the part numbers identified in appendix A.

## 1.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | ACM/PCM |
| Full name | Active and Passive Component Modelling |
| Author(s) and institution(s) | Brodie M L and Silva; University of Maryland |
| Date of first reference | 1982 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>structural design<br>detailed design |
| Software support | |
| Development status | under development as a research project, being applied in criminal scheduling, university registration and hotel reservations. |

Comment

It supports the functional decomposition, data decomposition, interface definitions, data flow, sequence control flow, concurrency and formal program verification. The approach is

claimed to have three consequences, which are; (a) equal emphasis on integrity of structured and behavioural properties of an application, (b) complete life cycle coverage, (c) modelling through the levels of abstraction. The principle of abstraction is a powerful tool which allows development to be carried out systematically by suppression of some details in order to place more emphasis on others.

## 1.2 LIFE-CYCLE MODEL

- Requirement formulation: an informal description of the real world knowledge of application,

- Logical design/specification: specification of an abstract semantic data model of the application ie global conceptual data and process models,

- Implementation design: definitions of schemas and programs in terms that fits the data model of the target system,

- Implementation: encoding and testing the implementation model,

- Operation, maintenance and monitoring: installation of the system,

- Evolution, adaptation and modification: meet the changing requirements.

## 1.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

The conceptual schema captures the following.

- Basic objects of the problem,
- Classification of each object as temporary or permanent, and either dependent or independent,
- Construction of individual object schemas by considering various relationship forms,
- Construction of object schema, and an identification of constraints.

Extensive use of the abstraction approach has been followed for both structural (data and static) and behavioural (process and dynamic) properties. The behavioural property refers to state transitions and dynamics (ie operations and their relationships). The structural property refers to both static and dynamic properties.

Notation used mostly graphical, some textual, and mathematical.

## 1.4 SYSTEM MODEL

Concepts used in system specification and design

predicate logic, BNF and transform techniques.

Notation used BETA language (sometimes graphical assistance is taken) contains difficult axiomatic and predicate transform techniques.

## 1.5 COMMENT

| | |
|---|---|
| Completeness | above average |
| Economy | below average |
| Ease of use | low |

Additional comments

Methodology covers mostly the logical design and specification stages of the system life cycle. The purpose of the methodology is to build an abstract model (requirements of the information system) as such it is more closed towards the activity of system description. Specificaion and logical design phases are procedural while other phases are fairly ambiguous. Logical design and specification phases describe "abstraction specification" of a system, whereas the analysis phase describes the real world informally and may be regarded as a fact-finding activity. Facilities for schema generation and program generation from the schema actions and transactions are not provided. It does not deal with boundary specification of the system, automation aspects, or management aspects. There is a lack of guidence for selecting object classes for integrating various object schema, selecting relationship abstractions for redundancy checking, and overall completeness and consistency

checking. It is not clear how using Pascal-R could be mapped to the implementation model. The developer requires skill to synthesize each object class and cope with the various relationships and schemas, constraints and assertions. The language (BETA) requires mathematical skills. The methodology is quite difficult from a user's point of view, the abstract model is not clearly attained, there is no clear distinction between things and their names.

## 1.6 REFERENCES


1. BRODIE M L and Silva (1982)

2. FREEMAN P and Wasserman (1982)

## 2.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | ASSET |
| Full name | Automated Systems and Software Engineering Technique |
| Author(s) and institution(s) | Osterweil L J and others; University of Colorado and Boeing Computer Company USA. |
| Date of first reference | 1979 |
| Application field(s) | embedded |
| Life cycle stages | requirement analysis<br>functional specification<br>detailed design |
| Software support | analysis and checking<br>prototyping<br>detailed design aids<br>code generation |
| Development status | under development |
| Comments | |

Early efforts related to methodology are focussing on an implementation of the key analytic capabilities (and front-end) to process requirements, design, and specific coding languages.

It recognises a need for, and incorporates the use of, iteration in systematic definition, refinements, and verification of requirements and design. It uses four implementation tools: syntax and standards checkers, DAVE (for static analysis), PET and prototype (to monitor executing Fortran and PL/1 programs), and symbolic execution technique (for source code, design and requirement specification).

## 2.2 LIFE CYCLE MODEL

- phase 1: requirement analysis
- phase 2: preliminary design
- phase 3: detailed design
- phase 4: coding

Testing and verification are included throughout the phases of software development.

## 2.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | above average |
| Ease of use | average |

Additional comments

The heart of the ASSET is a database containing all the information needed for making and implementing management decisions about a given program. The database contains source code, object code, documentation, support libraries, and project utilities. Requirement and design specification for the program also resides in the database. The important principle in ASSET is verification and testing during each phase of development and maintenance cycle, which provides the assurance that the software product is developing correctly. It may be regarded more as a verification methodology than an information system development methodology. It does not provide any assistance to capture and describe inputs, and does not describe different phases of development. BCS is also actively engaged in developing IDAP (system improving visibility and providing design verification), and to create and analyse SAMM diagrams (SAMM is a technique for hierarchically decomposing by the use of graphic tools). From the available literature notation used is a mix of graphic and texts, and no specific notation is prescribed.

## 2.6 REFERENCES

OSTERWEIL L J and others (1979)

## 3.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | ADS |
| Full name | Accurately Defined System |
| Author(s) and institution(s) | Lynch H J; NCR |
| Date of first reference | 1966 |
| Application field(s) | data processing |
| Life cycle stages | requirement analysis<br>functional specification<br>detailed design<br>programming |
| Software support | analysis and checking |
| Development status | used but obsolete |

Comments

ADS is a general purpose tool and functions with an equal effectiveness for any type of computer system.

## 3.2 LIFE CYCLE MODEL

- Study the feasibility of the application
- Survey of application
- Specification of time and cost factors,
- Development of computer programs

- Implementation and installation of the system,

All the above mentioned activities are performed in a circular sequence resembling the face of the clock.

Notation used forms and tables

## 3.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | average |
| Ease of use | high |

Additional comments

ADS facilitates the definition and communication of the objectives criteria and specification of an EDP system. It approaches the system definition by starting with specification of a report-form which is to be output. From this point, separate-integrated-forms are completed to specify input records, history records, computation and logic operations. All the system elements are tied together by a cross reference table, and the result is a precise set of system definition

## 3.6 REFERENCES

LYNCH H J (1974)

## 4.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | BISAD |
| Full name | Business System Analysis and Design |
| Author(s) and institution(s) | Honeywell |
| Date of reference | 1968 |
| Application field(s) | data processing |
| Life cycle stages | requirement analysis<br>functional specification<br>detailed design |
| Software support | prototyping<br>detailed design aids |
| Development status | used but obsolete |

Comment

The developer performs definite tasks in his efforts to analyse a business and to design an information system that responds to the needs of the management. An information matrix is used to represent the activities of data processing, which has five connections between inputs, outputs and files of the functional model.

Background analysis is the foundation upon which the future

system work is build up. Functional analysis implies the breakdown the total operation into logical groups of tasks to be carried out. A logical group of tasks is called a function, and each task therein is an activity. Once the model is approved and priorities areas are selected then prototypes are converted to a working model. Much of the problem associated with the systems work is implementation for which a plan should be established with detailed implementation criteria. All the documents resulting from the previous steps are collected and together are known as a "system specification". The last step is then to make the system operational.

## 4.2 LIFE CYCLE MODEL

- background analysis,
- functional analysis,
- designing the prototype
- designing the working system,
- operational planning,
- system specification,
- implementation and control.

## 4.4 SYSTEM MODEL

Concepts used in system specification and design

document, files, processing steps (activities), flow and their directions, generator functions, user functions, inputs and ouputs.

Notation used graphical and matrix form

## 4.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | average |

Additional comments

The central point of BISAD is system prototype design which is performed either by a decision table or flow chart or by an information matrix. This matrix shows five connections between input, the files and the output of the functional model. This matrix is specified completely, it is used for tracing the flow of information through the activities.

In order to show to the management that the required system is complete and meets their objectives, the system prototype is presented for their approval. This system prototype describes the functional model, business logic, the total system and the

general equipment requirements. BISAD does not have a specific notation of its own, the use of the information matrix may become complicated in large systems.

## 4.6 REFERENCES


1. HONEYWELL (1974)

2. COUGER and Knapp (1974)

## 5.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | CASCADE |
| Full name | Computer Aided Systems Construction and design Evaluation |
| Author(s) and institution(s) | Solvberg A; University of Trondheim, Norway. |
| Date of first reference | 1969 |
| Application field(s) | data processing, Science/Engineering |
| Life cycle stages | requirement analysis<br>boundary specification<br>detailed design |
| Software support | analysis and checking<br>detailed design aids<br>code generation |
| Development status | used but obsolete |

Comments

A software tool CASCADE/2, has been developed containing the modules for system specifications and presentation, specification analysis and program system production. A computer aided design module is also under development. Syntactic checks, consistency checks, and checking for the compatability of levels are

performed. The outcome of system analysis can partly be used for the definition of the new system. The designed system can automatically be documented in different ways eg. flow charts, lists and matrices.

## 5.2 LIFE CYCLE MODEL

- system specification,

- system description, and

- analysis by mathematical methods.

## 5.4 SYSTEM MODEL

Concepts used in system specification and design

P= process; A1, A2= information objects; IPS= information processing system; INF= information; I= input; O= output.

Notation used graphical and textual and symbols of its own.

## 5.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | high |
| Ease of use | low |

Additional comments

The statements of the system description are written in a formal language, which describes the information obtained through interviewing. This set of DATAWRITE is automatically controlled and combined to form a model file. All the work is limited by the features and characteristics of DATAWRITE language, which was designed to allow a static, formulised description of data systems. DATAWRITE has seven operations +, -, /, *, copying, accepting, despatching.

The most important problem in system design is that system documentation is not used for the direct benefit of designer and therefore may be felt as burden to him. Moreover subsystems are described and analysed by mathematical methods. A traditional life cycle model has been adopted for system development, no environment modelling is considered, and the notation used is difficult.

## 5.6 REFERENCES


1. INFOTECH (1975)

2. STRUNZ H (1973)

## 6.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | CORE |
| Full name | COntrolled REquirement Specification |
| author(s) and insttution(s) | Mullery G P and others; System Designers Ltd. |
| Date of first reference | 1979 |
| Application field(s) | defence and data processing |
| Life cycle stages | requirement analysis<br>boundary specification<br>functional specification<br>detailed design |
| Software support | analysis and checking<br>prototyping |
| Development status | in use |

Comments

The methodology leads to an early identification of subsystems which are of assistance in team structuring and control, but does not assist in areas of planning or budgeting. It was initially developed for avionics project, but can be used in other large and complex systems.

CORE allows the designer some degree of choice in the way in

which the proposed concepts are applied to a problem. The notation of CORE has been used to draw the view point diagrams for the hospital system as seen in DOWNES (1982).

## 6.2 LIFE CYCLE MODEL

- Examine the view points from which the requirements may be considered. These are the requirements as seen by various parties who interact to form the system.

- Specification of the requirements required by each view point from the proposed system.

- Drawing a table showing operations and the flow of data necessary to achieve the desired outputs. The table has five columns; sources, inputs, actions, outputs and destination. Each action has atleast one input, and each input has a source. Similarly each action must generate atleast one output and each output must have a destination. The data flows are shown by arrows.

- Using CORE diagrams to describe the implied action sequencing for each view point. These diagrams are known as data and action diagrams, and provide simplicity, quality control, and an assistance in providing description.

- Checking of completenes and consistency of different view points is performed against: inputs, outputs, actions, sources, and destinations.

## 6.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

events, actions

Notation used graphical

## 6.4 SYSTEM MODEL

Concepts used in system specification and design

data, action, store, item, composite item, activate, validation, data pool, request data.

Notation used graphical and textual

## 6.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | low |
| Ease of use | low |

Additional comments

CORE is based on answers to three questions, which are; (a) What are we trying to achieve? (b) Why we do fail to achieve that often? (c) What should we do to improve ?

CORE is more concerned with file design system rather than a

database system, and does not mention the technique of data integration. CORE diagrams may be regarded as a combination of SADT and SREM diagrams. . CORE accomodates different points of view, but the mechanism to aviod redundancies due to different points of view of the same data is not mentioned.

## 6.6 REFERENCES


1. MULLERY G P (1979)

2. DOWNES V (1982)

## 7.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | CSE-DBD |
| Full name | Constraint Specification in Evolutionary Database Design |
| Author(s) and institution(s) | Bracchi G and others; Instituto di Electtrotecnica, Milans, Italy. |
| Date of first reference | 1979 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>detailed design |
| Software support | |
| Development status | under development as a research project |
| Comments | |

Methodology concentrates mainly on requirement analysis and, to some extent, deals with the boundary and functional specifications. It also provides an assistance in specifying conceptual and quantitative requirements for database design. Static and dynamic requirements are defined.

## 7.2 LIFE CYCLE MODEL

The developer subdivides the elements into two classes, (a) conceptual requirements, and (b) quantitative requirements. These are summarised as follows.

- Specification of conceptual requirements: data schema (entities, relationships, attributes, static constraints; functional schema (operations, transactions, parameters, dynamic constraints); evolution schema (events, rules).

- Specification of quantitative requirements: for each entity type (number of instances of entity type, number of instances of entity type associated via a relationship); for each relationship type (number of tuples associated via a relationship type); for each attribute (size of possible values, number of different values), for each operation (frequency of execution, frequency of execution inside each transaction); for each transaction (frequency of execution); for each parameter (specification of elements to be used for process and access).

## 7.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entities, attributes, static and dynamic constraints, and parameters.

Notation used mainly graphical and some textual

## 7.4 SYSTEM MODEL

Concepts used in specification and design

data schema, functional schema, transactions.

Notation used textual and graphical

## 7.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | average |
| Ease of use | above average |

Additional comments

Methodology represents an integrated approach to requirements analysis, and shows that the conceptual requirements may be collected in three schemas: data schema, functional schema, and evolution schema.

Methodology is independent of any data model, design method, and database techniques. It can be used as conceptual foundation of an integrated methodology for collecting and expressing requirements needed to take specific design decisions.

## 7.6 REFERENCES

BRACCHI and others (1979)

## 8.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | CIM |
| Full name | Conceptual Information Modelling |
| Author(s) and institutions(s) | Gustafsson M R and others; University of Goteborg, Sweden. |
| Date of first reference | 1982 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design<br>programming |
| Software support | detailed design aids |
| Development status | under development as a research project |

Comments

CIM is mainly concerned with the conceptual modelling phase of information system development. It is a set of definitions of assertion types, rules and constraints which govern the relationship between assertions. CIM supports an incremental development during the initial phases of system life cycle. Two software tools are mentioned: CIPS (conceptual information processing system), and DBMS adaptation but not explicitly

defined. CIM supports the initial activities, and presents equations of the universe of discourse. The second role of CIM together with requirements (including layouts, response time, and timeliness requirements and interactions patterns) is to act as a formal base from which an information system model can be defined. This design step is analoguous to devising a set of numerical solution procedures for a set of mathematical equations, and it also involves storage and effeciency decisions.

## 8.2 LIFE CYCLE MODEL

- Development of an initial conceptual model
- Function and activity analysis
- Inference analysis
- Global constraints specification
- Consistency, completeness and satisfiability tests

## 8.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

entity type, attribute, function, events (external and internal), constrints, time, relationship type, data type.

### Notation used

textual (use of mathematical terms) and graphical

## 8.4 SYSTEM MODEL

Concepts used in specification and design

All those which are used in environment modelling.

Notation used textual and mathematical

## 8.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | low |
| Ease of use | low |

Additional comments

CIM contains useful theoretical concepts which are generally expressed in mathematical concepts and notation, as such may not be very suitable for data processing organisations. CIM is similar to NIAM except that NIAM is more user oriented, and CIM lacks a graphical representation, but both are data oriented. CIM describes a conceptual model, and a conceptual information processing system, but lacks in establishing the goals of the information system being developed, and the technical design consideration (estimates of machine load). Concentrates on a top-down approach, use of predicate calculus, clear conceptual model with time over which associations and attributes hold, declarative model between organisational and procedural models

and inference analysis. It also supports man-machine interface, data analysis, functions, files and database, programs and modules, and data set specifications. Mathematical view adapted provides the equations of the system which forms the basis for various processing solutions, and gives a real sense to the temporal dimension and insights into data and process behaviour.

Processes are given secondary importance; there are no algorithms for validation, completeness/consistency checking, there is no provision of a particular graphical representation, the design phases are ambiguous, there are too many artificial entities and events produced due to inference analysis, and, as such, CIM becomes very large and unmanageable. Specificaion of global constraints is separated from events, relationships, and entity types which may cause inconsistencies or incompleteness in specifications, as the rule is not associated with its constituent parts. The identifier of an entity type consists of attribute functions, which is an unnecessary restriction on naming convention, the ability to use more complete means of reference are desirable. Data model is described by first order predicate calculus, events by separate system object, constraints by identifier, value set, generalisation, and derivation rules are described by formulae of predicate calculus.

## 8.6 REFERENCES

GUSTAFSSON M R and others, (1982).

## 9.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | CADES |
| Full name | Computer Aided Design and Evaluation System |
| Author(s) and institution(s) | Warboys B; ICL |
| Date of first reference | 1970 |
| Application field(s) | data processing and operating system |
| Life cycle stages | functional specification<br>detailed design |
| Software support | analysis and checking<br>detailed design aids<br>code generation |
| Development status | used but obsolete |

Comments

The methodology and its associated high level languages obviate the need for using any other methods in parallel. Its database holds all information relevant to the project throughout the development process.

## 9.2 LIFE CYCLE MODEL

### Tasks to be performed by the developer

- High level design: providing an initial abstract analysis of holons which form the operating system, data entities, external and internal interfaces. At this level the holons are humans, devices and jobs etc; the data entities are messages, sets, events and job control programs.

- Low level design: providing a high level language representation of the operating system. The holons are high level language procedures and macros.

- High level design implementation: providing a loadable binary representation of the operating system. At this level holons are regarded as loadable binary modules, and data entities are loadable binary areas. The information in a CADES database describing this level determines how the loadable binary objects (areas and modules) are collected together to form various loadable binary versions of the system.

- Loading: provides the loaded version of operating system. At this level of abstraction the holons and data entities are hardware oriented entities. The information at this level in the CADES database determines how the operating system is mapped to the hardware entities.

## 9.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

data, entity and relationship

Notation used graphical and some textual

## 9.4 SYSTEM MODEL

Cconcepts used in system specification and design

holons, mapping, responsibility, data used, function.

Notation used textual

## 9.5 COMMENT

| | |
|---|---|
| Completeness | above average |
| Economy | average |
| Ease of use | low |

Additional comments

CADES is based on top-down hierarchical decomposition of data handled by the system, and a tree structure of data decomposition. Holons (functions applicable to data) are decomposed producing a functional design tree. Any level in the data tree, and the corresponding functions in the holon tree constitute an abstract machine. Holon descriptions are entered

into the CADES database using a SDL and allow a formal syntax but it does not provide a formal semantic description. Method refines the data and holon trees from the highest requirement to the lowest implementation level. Development of this methodology took 750 man years, spread over five years. CADES is useful for large projects, being used for operating system development. The methodology and computer aided system would have to facilitate all stages of operating system development ie. high level design, low level design, implementation and maintenance. They would have to encourage the codes of good practices which prevailed within the computer industry ie. structured programming, data entity driven design, delays fixing and binding, design of resilience etc. Structural modelling supports certain characteristics such as modularity, top-down abstraction, top-down detail, database view and management control.

## 9.6 REFERENCES


1. DoI (1981)

2. INFOTECH (1975)

## 10.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | D2S2 |
| Full name | Development of Data-sharing Systems (System development in shared data environment) |
| Author(s) and institution(s) | Palmer I R and others; DMW group, London |
| Date of first reference | 1982 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design<br>programming |
| Software support | data dictionary<br>detailed design aids |
| Development status | in use and under development |

Comments

D2S2 is still being improved under the guidence of I R Palmer. The original work was undertaken by Tozer E (Scicon) in 1973. It was first used by CACI on consultancy projects in 1975, and being inhanced continuously by the above mentioned team. The extent of its use is not known. Up to 1978 it was purely data analysis

oriented, during 1979 it was extended to include certain process analysis techniques eg. DFD, functional decomposition etc. After this, D2S2 being revamped to integrate fully its data and process analysis aspects. In its current form it contains six phases of development life cycle of which analysis and design are defined in detail.

## 10.2 LIFE CYCLE MODEL

- Strategy stage in which the organisation is documented in terms of entities and functions,

- Analysis stage, consisting of the analysis of; decompositions, interactions, decisions, application, transition), and test for completion.

- Design stage, (consolidation, global design, data design, application design, operational design, program design and transition design).

- Construction stage, (new equipment, database construction, program construction and system construction).

- Transition stage, (user preparation, data conversion, parallel operation, user acceptance, operational documents).

- production stage, (evaluation, documentation adequacy, running system cost, user reactions).

## 10.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entity, attribute, relationship, and event.

Notation used graphical and matrix

## 10.4 SYSTEM MODEL

Concepts used in system specification and design

entity type, relationship type, optional relationship types, contingent relationship type, mandatory relationships, degree of the relationship, function type, identifier, entity function matrix, functional dependency diagram, data dictionary, decomposition of functions and entities, logical and physical data models, input forms, report layout, screen layout. (Methodology uses most features of entity-attribute-relationship approach with some extension).

Notation used graphical, tabular and textual

## 10.5 COMMENT

| | |
|---|---|
| Completeness | above average |
| Economy | low |
| Ease of use | average |

Additional comments

D2S2 is most suitable for shared environment of data for its development. Six fundamental principles constitute D2S2, which are:

- a clear distinction between analysis and design phases,

- a complete separation between analysis and design tasks,

- an orientiation towards producing a strategy for system development,

- decomposition into well defined tasks,

- emphasis on simple diagrams with structured specifications,

- interactive use of data dictionary system, and

- Production of business specification, system design and program specification.

About forty tasks are defined respectively for design and analysis. The data analysis is based on a conceptual data model in terms of entities, attributes and relationships. The process analysis is similar to "YOURDON" structured analysis technique. No allowance appears to be made for iterative work. Each selected area is analysed in detail until its complexities are understood, through functional decomposition, detailed entity model diagram, functional logic model.

Design objectives are to produce the outputs, entity usage analysis, entity usage matrix, database schema, entity usage cluster analysis, transaction control matrix, program flow diagram for data, and test plan. D2S2 is defined at three level of decomposition, external, logical and physical.

## 10.6 REFERENCES


1. OLLE T W, (1982)

2. FREEMAN P and Wasserman, (1982).

## 11.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | DADES |
| Full name | A method for specification and design of information systems |
| Author(s) and institution(s) | Olive A; Universitat Politecnica de Barcelona. |
| Date of first reference | 1982 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>detailed design |
| Software support | analysis and checking |
| Development status | under development as a research project |
| Comments | |

Requirement analysis covered broadly as in ISAC and is data oriented. Methodology is based on the concepts of "precedence between sets" (Langefors, 1973), Young and Kent Algebra (1958). DADES supports data decomposition, interface definition, and formal program verification. Specific tool support is TBD which

is being developed. It does not support management aspects. Completed system is validated against original requirements by a consistence/derivability analysis.

## 11.2 LIFE CYCLE MODEL

- list input requirements,
- develop an abstract conceptual schema,
- decide naming conventions,
- develop the conceptual schema,
- define final input/output requirements,
- define derivation rules,
- validate specifications,
- architectural design.

## 11.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

conceptual schema, universe of discourse, assertion time, extrinsic and intrinsic time.

Notation used graphical

## 11.4 SYSTEM MODEL

Concepts used in system specification and design

time functions, derivation rules, activities, output requirements, domains of schema, points of interval of life cycle, relation schemes, time ordering.

Notation used diagrams and tables

## 11.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | average |
| Ease of use | low |

Additional comments

The current version is still in reseach stage, and the useful features are: specification of information system without making assumptions about the system structure or database; validation of the logical consistency by using precedence analysis (Langefors) at static and dynamic levels; verification of decisions before making further decisions; focuses mainly on data and little on processes; consists of a formal language. DADES notation for specification is ambiguous, being a combination of some existing notations.

Its precedence analysis method is similar to Langefors (1973);

derivation analysis and consistency checking is similar to Systematics-Grindley (1975) and treatment of time and predicate expressions are similar to Young and Kent (1958). Prescribed workproducts are formal specifications and architectural design.

## 11.6 REFERENCES

OLIVE A, (1982).

## 12.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | EDM |
| Full name | Evolutionary Design Methodology |
| Author(s) and institution(s) | Rzevski G and others; Kingston Polytechnic U.K. |
| Date of first reference | 1982 |
| Application field(s) | data processing, embedded, science/engg. |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>prototyping |
| Development status | under development as a research project |

Comments

EDM is strongly based on functional decomposition. The research study has been empirical. Hypotheses are made on the importance of various factors, dealt one at a time, and then these hypotheses are claimed as tested during information development project. The aim of the project has been to improve the quality

of information systems and the productivity of engineering personnel. EDM prescribes that, before any agreement on user requirement is finished, the developer should develop, with full participation of users, a model of the total information system of which the target system is a subset.

## 12.2 LIFE CYCLE MODEL

- Formulate the functional specification of the total information system.

- Formulate the data structures of the total information system.

- Apply the above activity to each newly created function, in turn, until the functions are designated to be either manual or interactive (ie until there is no function left which needs to be performed by a combination of these two methods).

- Summarise the model of the total system by a diagram depicting its hierarchical structures.

- Formulate man-machine system design (9 tasks are mentioed); form data flow structures of man-machine system, and design control structures of man-machine system.

- For each man-machine system, formulate the functional specification as described in stepl, above.

- Decide which system is to be designed first.

- Design data flow structures of the selected man-machine

subsystem.

- Design external data structures for the selected man-machine subsystem (ie. for each set of data entities which is transmitted between the user and the machine in one transaction).

- Design control structures for selected man-machine subsystem (ie. for each function to be performed by machine).

- summarise the design of man-machine subsystem by means of diagrams depicting its hierarchical structure.

- Software design (ten steps are summarised), which are: formulate functional specification; define input/output data sets; design the conceptual data structures; design data flow structures; design external data structures; design control structures; define functional specification for each subsystem; design the software subsystem which supports the first man-machine subsystem; define constraint module of the subsystem; carry out implementation design.

## 12.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

entity, attribute, relationship, domain.

### Notation used textual (not defined explicitly)

## 12.4 SYSTEM MODEL

Concepts used in system specification and design

domain description, data type, range, functional decomposition, data flows, conceptual data structures, hierarchical structures and man-machine system, entity sets, relation sets, group.

Notation used graphical, tabular, relational.

## 12.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | above average |
| Ease of use | average |

Additional comments

EDM concentrates more on processes and less on environment modelling. Specification of quality parameters is comprehensive to determine the quality of information system, traditional life cycle phases are followed, and contains a set of constituent activities of information engineering. Participation of users and designers guarantees the success of the target system. The importance of requirements specification in the system life cycle is widely recognised. It may be difficult to users to visualise by just agreeing on a textual document how the system will actually work in their environment. Moreover the correctness of

the requirement analysis is not guaranteed.

EDM lacks in formality and gives several solutions of a problem at a time, and sometimes system design stages may become clumsy to follow. The factors during that affect the completeness of the requirements specifications and changes in user requirements, which are in partial control of the developer, are: inability of users to anticipate their needs; inability of users to anticipate the direction of their future requirements; and the lack of designers appreciation of users needs. EDM is similar to SADT, CIM in the early part of modelling of the life cycle. Automated support is provided for design document preparation and some for testing the design solution.

## 12.6 REFERENCES


1. RZEVSKI G and others, (1982)

2. FREEMAN and Wasserman (1982).

## 13.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | FAGIN |
| Full name | FAGIN design and code inspection |
| Author(s) and institution(s) | Fagin M E; IBM |
| Date of first reference | 1975 |
| Application field(s) | data processing (only for determining the check points for inspection) |
| Life cycle stages | detailed design |
| Software support | analysis and checking |
| Development status | developed in IBM |

Comment

This is not a design methodology, but a set of methods for finding errors in designs, code and test plans. These test plans are called inspection plans, and are applicable to design and implementation and test planning stage of system development process.

## 13.2 LIFE CYCLE MODEL

- Overview: where the developer describes the product to the remainder of the inspection team,

- Preparation: study of individual products by the inspection team members,

- Inspections: user describes his understanding of the product, and the moderator writes the inspection report within one day,

- Rework: all the errors noted by inspection teams are resolved by the developer,

- Followup: moderator verifies the quality of the rework, if the rework is > 5%, then a complete reinspection is carried out.

## 13.4 SYSTEM MODEL

Concepts used in specification and design

Since it in not a design methodology, therefore only check points which require inspection are mentioned.

Notation used textual

## 13.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | average |
| Ease of use | low |

## Additional comments

FAGIN is a technique only for checking and can be applied to any methodology. Designs are checked for compliance with requirements, code is checked for the compliance with design and test plans. Procedures are checked against requirement and designs. All the checks are performed for internal consistency. Once a product has passed its inspection then it is bonded (frozen).

These inspections can constitute to the technical control aspect of software project management. A status reports can be produced from inspections, enabling project management to monitor the state of each product. Since this inspection continues throughout the design and implementation phases, progress can be monitored continually in the early phases of the project development.

## 13.6 REFERENCES

DoI, (1981).

## 14.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | GEIS |
| Full name | Gradual Evolution of Information System |
| Author(s) and institution(s) | Keha V; Finland |
| Date of first reference | 1981 |
| Application field(s) | d/p, o/s, and tools |
| Life cycle stages | requirements analysis<br>functional specification<br>programming |
| Software support | analysis and checking<br>prototyping |
| Development status | under development |

Comments

GEIS does not provide very clear definitions of disjoint stages of development process, and supports functional hierarchy, data hierarchy and interface definitions, seems to be weak in boundary specification and detailed design. The work product is a specification library.

## 14.2 LIFE CYCLE MODEL

- Initial phase to determine general schema,

- Limitation of the system: specification of limitation schema,

- Description of objects: specification of object schema,

- Descriptions of transactions and associated functions: specification of transaction schema,

- Specification of fields,

- Specification of programs,

- Testing and interacting.

## 14.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

instance, object and relationships.

Notation used graphical

## 14.4 SYSTEM MODEL

Concepts used in system specification and design

general schema, limitation schema, object schema, transaction schema.

Notation used graphical with some textual

## 14.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | low |
| Ease of use | average |

Additional comments

Methodology seems to have ideas from JSP and relational database model. The main objectives of the methodology are to provide the information system acceptable to the users, and also that the users understand the working of the information system.

It lacks in some technical concepts e.g.response time, security, integrity etc; there does not exist specific constructs to build the information system on evolutionary basis; it lacks in theoretical foundations and may be suitable for simple data intensive applications. The purpose of schema tools is not clearly defined, no clear definition of automated generating functions.

The strength appear to be its emphasis on an incremental design, and its accessability to both user and developer, the provision of software tools for interpretative execution and program generation. It has same type of tool kit as the structural design school, but is very much vaguer. It also resembles to the Cobol program generator school, specially in its identification of functions (ie. selection, projection, sort, match); while

inputs, outputs, Cobol generator tools are not well defined. Author has incorporated JSP in his standard practices for creating Cobol programs which seem to be similar to WARNIER (1981). GEIS do not support management aspects, quality assurance methods applied to work product is "author reader cycle" and the completed system is validated against original requirements by end user feedback.

## 14.6 REFERENCES


1. KEHA V, (1982),

2. FREEMAN and Wasserman (1982)

## 15.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | GAMMA |
| Full name | GAMMA |
| Author(s) and institution(s) | Falla M E; Software Sciences Ltd |
| Date of reference | 1980 |
| Application field(s) | defence and data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>prototyping<br>detailed design aids<br>code generation |
| Development status | under development |

Comments

There is no distinction between the design and coding stages of implementation. It does not contain an effective set of tools and techniques to cover all stages of system development life cycle. GAMMA philosophy is evolutionary is based on empirical development, and is more a documentation technique. The tools

are: language to state design, design documentation system, a set of extra tools to the developer, and computer based tools for correctness and performance.

## 15.2 LIFE CYCLE MODEL

- Determine resources,

- Determine the system model,

- top-down design,

- Project work bench.

## 15.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

class, data class, modules.

Notation used graphical

## 15.4 SYSTEM MODEL

Concepts used in specification and design

procedures, data structures, implementation techniques, system structures, processes, decision tables.

Notation used textual (mainly)

## 15.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | average |
| Ease of use | low |

Additional comments

Gamma is more suitable for medium to large size projects. The number of code generators available limits the number of suitable applications. Its database contains tools for creating, amending and inspecting abstract modules together with some checking tools. It is available on IBM 360/370. It uses a planning for modification of sequential upgrades of a product supplied to a single user. The design of a software product with variants produced parallel for several users has not been dealt, and do not provide facilities to define global constants. It allows each user to evolve a language closely adapted to each application area.

## 15.6 REFERENCES


1. FALLA M E (1981)

2. DoI, (1981)

## 16.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | HIPO |
| Full name | Hierarchy plus Input, Process, Output |
| Author(s) and institution(s) | Welf W; IBM |
| Date of first reference | 1972 |
| Application field(s) | data processing |
| Life cycle stages | functional specification<br>structural design<br>detailed design<br>programming |
| Software support | analysis and checking<br>prototyping |
| Development status | used but obsolete |

Comments

HIPO was developed as a documentation package consisting of four parts: a visual table of contents (VTOC); an overview diagram; a detailed diagram and an extended description. It is used to support the use of structured programming as a design approach.

## 16.2 LIFE CYCLE MODEL

First the developer identifies a function and enters it as a new box in the VOTC; prepares an overview diagram (to do this first he lists all the outputs on the RHS of a sheet of paper) and then specifies the inputs needed to produce these outputs, which are noted on the LHS of the same sheet; rearranges the processes in a logical order; summarises the columns in a format of an overview diagram. After this the developer prepares detailed diagrams, and an extended description of each box is entered in reviewed VTOC. This process is like the preparation of overview diagrams but with a rearrangement of data within the input and output columns, and linking with connecting arrows, and data items are similarly linked with processes. The preparation of detailed diagrams has two added middle stages: (a) course tuning, and (b) fine tuning; to simplify the appearance of detailed diagrams.

After another review the detailed diagrams and VTOC can be further amplified and the extended descriptions enable the programmers the implementation by using HIPO charts, as a basis for programming and testing.

## 16.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

input, process, output, data, control.

Notation used graphical

## 16.4 SYSTEM MODEL

### Concepts used in specification and design

identification of data and processes, overview and detailed diagrams, visual table of contents.

Notation used mainly graphical

## 16.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | average |
| Ease of use | average |

Additional comments

HIPO concentrates on processes and their hierarchy. A process is completely described by graphical notation. It is mainly limited how to use forms and templates and a narrative means to represent design. HIPO charts are useful for defining major program functions, but they provide a disjointed view what a program is doing as a whole. It ignores the sequential nature of programming. It is difficult to estimate the degree of complexity and the amount of coding required.

It is claimed that HIPO can be used as a design tool to improve communication with users; as a means to provide documentation; and as an aid for maintenance.

## 16.6 REFERENCES


1. TEICHROEW D, (1977)

2. BREWER T, (1979)

3. COTTERMAN and others, (1981)

4. LUDEWIG and others, (1978).

## 17.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | HOS |
| Full name | Higher Order Software |
| Author(s) and institution(s) | Hamilton M and Zeldin S; Higher Order Software Inc |
| Date of first reference | 1976 |
| Application field(s) | defence, science/engg, o/s, tools, experts systems. |
| Life cycle stages | requirement analysis<br>functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>detailed design aids<br>code generation |
| Development status | in use |
| Comments | USA and Isreal defence departments have recently funded the project for further development to suit their demands. HOS requirements in terms of design are stated as in SADT and ISDOS. |

There is no specific aspect of methodology for modification, evolution and control. It can be used with other methodology which addresses configuration control or top level requirements analysis. HOS specification ultimately becomes of the form which can be used by Ada, Simula and ALGOL-68.

HOS developed after APOLLO-11 with the aim to develop techniques to apply to some of sky-lab software, which was a kind of maintenance mode to APOLLO, and later on the shuttle flight software. Some anomalies were noted during configuration control: (a) 70% problems occurred due to interface and timing; (b) conflicts between software and hardware and between man and machine. Tools to support the development process are: RAT (resource allocation), Analyser (to make sure the rules are followed), Collector (to collect hardware to execute system on the higher order machine).

## 17.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

empirical data, control axioms, control map.

Notation used graphical

## 17.4 SYSTEM MODEL

Concepts used in system specification and design

data types, primitive operations, control structures, graphical control map, functional hierarchies, set partition, union and intersection and set category theory.

Notation used textual, graphical and mathematical.

## 17.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | low |

Additional comments

It supports: top-down design strategy; correctness of design at successive levels of decomposition; information of users through control map; encourage a dialogue for requirement formation. HOS may be used as a meta-methodology in the sense that one can define the syntax of SREM, MASCOT graphics in terms of the HOS notation (ie. AXES).

HOS mechanism is to read library that currently exists for building systems and to evolve new mechanisms to obtain more abstract control structures, abstract data types, and operations. This process continues recursively until the library is complete.

HOS supports the technical concepts such as function and data decomposition, interface definitions, data flow, sequence control flow, concurrency, and formal program verification. Its products are: formal specification in a library, graphic control map, and program code. It also supports the management aspects: project, technical and validate work products and system evolution. HOS determines inputs/outputs keeping in view that one does not know before hand and an interactive procedure is adopted to achieve this, HOS has two aspects one as a realtime, and the other is the system development itself in order to make the deliverables. The language "AXES" used is quite difficult and unsuitable for data processing community, control map and axioms are ambiguous and also difficult for an average developer.

## 17.6 REFERENCES


1. TEICHROEW D, (1977)

2. DoI, (1981)

3. LUDEWIG J, (1978)

4. FREEMAN and Wasserman, (1982)

## 18.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | Information Algebra |
| Full name | Information Algebra |
| Author(s) and institution(s) | Bosak R and others; CODASYL. |
| Date of first reference | 1962 |
| Application field(s) | dp, sc./engg, tools, expert systems. |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design |
| Software support | |
| Development status | published but probably never used |
| Comments | |

The language structure group (LSG) of CODASYL formed in 1959 to provide a formal theoretical base to programming languages and theory of data processing. LSG has not produced a comprehensive theory, but many concepts in the report could contribute to such a development and further research in this area. The algebra cannot be applied to data processing as a methodology, but certainly it provided valuable concepts which are reflected in most of modern methodologies.

## 18.2 LIFE CYCLE MODEL

- Specify the property space (entities, relationships),

- Determine areas (files to be used),

- Determine value set of properties,

- Define function of glumps,

- Describe the glumping function of the system,

- Describe areas glumped,

- Take union of or cartesian product or join (as appropriate) of the areas which determine the outputs.

## 18.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entity, property, event, instance.

Notation used mathematical

## 18.4 SYSTEM MODEL

Concepts used in specification and design

property space, line, bundle, glump, bundling function, function of glumps, datum point, and area.

Notation used mathematical and tabular, textual.

## 18.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | high |
| Ease of use | low |

Additional comments

It provides manual design method, but might be developed to permit automatic generation of programs, and provides a good formal theoretical analysis of problem domain. It is an important initiative and requires further research to make it in a more usable form for the data processing community to describe the models of information systems. LSG has not been able to produce a user oriented easy language for defining problems, neither it provides algorithms to translate I.A statements into machine level programs, but these efforts do contribute to further refinement and extension of the I.A, by incorporating essential functions and operators.

## 18.6 REFERENCES


1. CODASYL, (1962)

2. TEICHROEW D, (1972)

## 19.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | ISAC |
| Full name | Information Systems work and Analysis of Changes |
| Author(s) and institution(s) | Lundeberg M; The Institute of Development of Activities in Organisations, Sweden. |
| Date of first reference | 1879 |
| Application field(s) | dp, tools, science/engg. |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>structural design<br>detailed design |
| Software support | prototyping |
| Development status | in use |

Comments

It is process oriented and covers all stages of system development process, except operation and maintenance, and deals in detail the early part of system development process. Problem oriented work is concerned with requirements analysis to analyse

the problems of organisation and to determine changes which are needed, such as the development of a new version of information system. Data oriented work is concerned with implementation aspects of the system. The outputs of ISAC are: A-graph, text pages, property tables, and other tables. ISAC is not based on a particular data model.

## 19.2 LIFE CYCLE MODEL

- Change analysis,(description of changes in the problems of the enterprise)

- Activity analysis, (different ambition levels are specified),

- information analysis, (specification of the aims of the target system).

Each of the above mentioned levels may further subdivided into a number of steps.

## 19.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

A-graph, activity, set, flows.

### Notation used mainly graphical

## 19.4 SYSTEM MODEL

Concepts used in specification and design

A-graphs, I-graphs, C-graphs, process tables, D-graphs.

Notation used graphical

## 19.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | low |

Additional comments

It provides: an understanding of the application; identification of socio-technical and economic problems; can solve complex problems by decomposing it into subproblems; concepts taken from Langefors (1973) of separating infological and data logical problems; triggering mechanism is similar to Grindley (1972). It stresses the learning of information system design; investigative and diagnostic aspects; user participation; environment modelling (through change analysis and activity study); a sequence of systematic activities of specification and design.

ISAC does not provide any commercially available hardware or software support, and the details of developing software are not specified; computer aspects are not considered as a part of

documentation; graphical notation is sometime clumsy. The rules for consistency checking and decomposition of activities; algorithms to specify different type of graphs; rules for data modelling and user interface are not clearly described. Complexity of ISAC can be in the order: NIAM, EDM, ISAC. ISAC is based on the importance of people in the organisation and provides structured walkthoughts inspections and prototyping for checking against original specifications.

## 19.6 REFERENCES


1. LUNDEBERG M, (1982)

2. FREEMAN P and Wasserman, (1982).

## 20.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | IML |
| Full name | IML-Inscribed High Level Petri nets |
| Author(s) and institution(s) | Richter G and others; West Germany. |
| Date of first reference | 1882 |
| Application field(s) | dp and real time |
| Life cycle stages | functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>detailed design aids |
| Development status | under development as a research project |
| Comments | |

IML emphasises the distribution and concurrency aspects in the design of information systems. Although hardware development made the distribution commercially attractive there remains still some engineering problems intrinsicly tied to the idea of distributing processing autonomy. Such an approach must first establish the casual structure of the problem and only then to proceed to the problem of designing a system which is complete

with the casual structure.

In a Petri net, a channel represents a predicate, the set of things for which the predicate holds (interpreted as the representative of the contents of the channel). Agencies represents all possibilities of coincidence change of the predicate extensions, and are known as transactions. The methodology provides insights into distribution and concurrency aspects in the design, used to obtain a distributed solution (if required). The casual system structure is elaborated with minimal data structure. Very few details are given about a method which enables the analyst to desgn Petri nets, and also do not give the information regarding conceptual modelling.

## 20.2 LIFE CYCLE MODEL

- Specify first overview nets (channel agency nets) to provide a general overview of the information flows in the proposed information system problem.

- Specify high concurrency net (to arrive at a more detailed description of the information flow and introduce a stricter interpretation of the nets to arrive at a predicate/transition net namely Petri net).

- Specify second overview net (to understand and survey the entire organisation).

- Specify low concurrency nets (for implementation, making decision

as to which functions and data are to be grouped into functional units).

## 20.4 SYSTEM MODEL

Concepts used in specification and design

first overview channel agency net, double arrow convention for Petri nets, convertion for a non-destructive read operation, inscription macros, IML box representation, second overview nets, low concurrency net.

Notation used mainly graphical some textual

## 20.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | average |
| Ease of use | low |

Additional comments

User particaipation in the system development phases entails new requirements for system design methods and tools, the most conspicuous requirement being a high level emphasis on modifiability of design due to interface adaptation and evolution. Thus at each step in the development process, the design and implemented product must be alterable and hence mentally manageable on any level of detail. This approach first

identifies the conditions of application (to be created during any distribution), and then identifies the casual structure and proceed to a design compatible with casual structure. The description tool is a cross between two independent conceptual systems: predicate/transition nets and information management concepts, for which a suitable language IML has been specified. In channel agency nets nothing is mentioned about the packaging of information into messages, sequencing, information transformation; and the nets are also unspecific with regard to the disposal of used information (ie. whether it is retained or eliminated).

Notation is difficult and insufficient to describe the entire development process.

## 20.6 REFERENCES

RICHTER G and Durchholz, (1982)

## 21.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | JSD |
| Full name | Jackson System Development |
| Author(s) and institution(s) | Jackson M; M. Jackson Ltd. |
| Date of first reference | 1980 |
| Application field(s) | dp, o/s, tools, embedded. |
| Life cycle stages | requirements analysis<br>functional specification<br>structural design<br>detailed design<br>programming |
| Software support | |
| Development status | in use, though still under development |

Comments

JSD being relatively new, there is little experience upon which to base the judgement. It is based on simulation modelling. It may be regarded as an extension of JSP, into the areas of systems analysis, specification, design and implementation. It is used as a basis for program design, and system design by providing a representation of the structure of the data handled by the system.

Technical concepts supported by JSP are: data hierarchy, interface definitions, data flow, sequence control flow and concurrency, but do not support formal program verification, and functional hierarchy.

Work products are: entity and action list, entity structures (trees), system specification structure texts, system implementation diagrams, executable texts and database design.

Quality assurance methods are: author/reader cycle, structured walkthoughts and inspections. Completed system is validated against original requirements by manual checking, specific tool support is under development. JSD system building of the information system comes before describing any function, because it assumed that model itself implies functions, change can be easily incorporated in the model, and the model is more stable than functional description and improves developer/user communication.

## 21.2 LIFE CYCLE MODEL

- Entity action step: specification of real world entities and actions.

- Entity structure step: actions suffered or performed by each entity are arranged in their ordering of time.

- Initial model step: description of reality in terms of entities and actions and the connections between the model and real world.

- Function step: functions are specified to produce the required outputs of the system.

- System timing step: considerations of the process scheduling which might affect the correctness or the timeliness of the system's functional outputs.

- Implementation step: specification of hardware and software (transformation, scheduling, database definition techniques are applied to run the system efficiently).

## 21.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entities, action, process.

Notation used graphical (strucured diagrams)

## 21.4 SYSTEM MODEL

Concepts used in specification and design

reality, functional specification, time dimension, static and dynamic models, process connections, channel, data stream, inversion, levels 0,1 and 2, state vectors, sequential data streams and dismembering.

Notation used mostly graphical and some textual

## 21.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | low (for large systems specially) |
| Ease of use | average |

Additional comments

Model in term of processes is expressed as: (a) specification of the processes to be contained in the model, (b) how the processes to be connected with data streams, and after this the developer consider the functions. The model is chosen with some idea of functions, but this idea is articulated in terms of model itself, and as such some functions become possible while others impossible. The impossible functions are those referring to entities and actions which are omitted from the model.

JSD recognises that the specification lies at the process level; a sequential process is regarded as an entity; the process scheduling is determined at specification rather than when the system is implemented.

It supports that the complete reality should be mirrored by the model, but the parallel and intermediate ways are not clearly defined. Three types of functions are specified which are: embedded, imposed and interactive. JSD embodies the JSP implementation technique of process scheduling by program inversion; and it shares underlying principles and concepts which

in JSP are not so clear but in JSD they are more explicit.

JSD in the initial step of his procedure enables the designer to have the specification of functions in his mind, which is an ambiguous state in the methodology. JSD allows sequential processes while in the real world there are processes which are not sequential. Real world fighter plane fire and fly at the same time, while JSD models the reality in sequential way because programs are sequntial. Outputs are of primary importance in the design, while JSD considers them in fifth step of his method.

## 21.6 REFERENCES


1. COTTERMAN and others, (1981)

2. FREEMAN and Wasserman, (1982)

3. JACKSON M, (1983).

## 22.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | LBMS |
| Full name | Learmonth Burchett Management Systems Development Methodology |
| Author(s) and institution(s) | Hall J; LBMS London. |
| Date of first reference | 1981 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>structural design<br>detailed design<br>programming |
| Software support | data dictionary |
| Development status | in use |

Comments

Methodology is structured on the lines of Gane and Sarson, and includes some concepts from Codd, Martin and Bachman, and covers most of the stages of traditional development life cycle. The input to the methodology is initial study, and the outputs are program specification, user procedures, operating instructions and database design. It introduces a set of rules (first cut)

which automatically converts the logical design to a physical organisation for DBMS; uses DFD, logical data structuring techniques and 3NF synthesis.

## 22.2 LIFE CYCLE MODEL

- Analysis of current automated or manual system (initial report),
- Outline the design of the proposed system (both processing and data),
- User management selection of the service required, based on the cost, time, and available resources,
- Detailed data design
- Detailed process design,
- physical design

A detailed design set of steps is also provided.

## 22.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

events, state, change.

Notation used graphical some textual

## 22.4 SYSTEM MODEL

Concepts used in system specification and design

audit control, logical data structures, process, user options, 3NF relations, composite logical structures, function catalog, first cut program outline, program specification, operating procedures and DBMS or file definitions.

Notation used graphical, textual

## 22.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | low |
| Ease of use | average |

Additional comments

Methodology may be regarded as a mixture of both data and process analysis, and represents data in three forms: DFD, entity model, and transaction histories. The logical data structuring technique overview runs in parallel with creation of DFD's. The idea is that the detailed investigation should be complete before other subphases are started. There is no specific mention of boundary specification. The developer outlines the design of the target system followed by data design, procedure design and physical design. One tool is specifically mentioned ie. data

dictionary. Checking is performed in four phases: program testing, system testing, acceptance testing and volume testing.

All documentation is prepared manually and creates a test strategy. Full system documentation is built up as analysis and design process. Approach is independent of any hardware. No major software aids seem to required except for standard utilities for testing/checking. No specific SSDL is mentioned.

## 22.6 REFERENCES

HALL J, (1981).

## 23.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | Langefors Algebra |
| Full name | Langefors Algebra |
| Author(s) and institution(s) | Langefors B; Stockholm University |
| Date of first reference | 1964 |
| Application field(s) | dp, o/s, AI/Exp., embedded and science/engg. |
| Life cycle stages | requirements analysis<br>boumdary specification<br>functional specification<br>detailed design |
| Software support | |
| Development status | published but probably not directly used |

Comments

It gives over emphasis on data transport and less on design, no unified notation for system specification and design, does not provide a complete system model. Precedence analysis and other theoretical concepts provide a very strong theoretical base to the developer for his system development work. The fundamental principle of systems work is the key point suggested for the design considerations.

## 23.2 LIFE CYCLE MODEL

1. Fundamental principle of system work, is defined as follows:

- Definition of the total system as a set of parts,

- Definition of system structures (ie. interconnections between the parts),

- Specification of the system parts and properties of each part,

- Specification of the properties of the total system, and repeat the above mentioned procedure until system specification is satisfied.

2. Divide the system work among several people who have experties in the particulr task.

3. Formulise the tasks mathematically: this is known as most efficient way in which the "fundamental principle" can be applied, and successfully experimented in electrical networks and elastic structures, and is performed by applying matrix algebra and algebraic topology.

4. If a mathematical model of working principle is not possible, a strict adherence to the principle led to successful systems work, making extensive distribution of labour possible and yet leading to no incompatibality problems when connecting design parts, a problem which is otherwise common.

## 23.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

pre-knowledge, e-message, e-facts, event.

Notation used graphical mathematical and textual

## 23.4 SYSTEM MODEL

Concepts used in system specification and design

precedence, succedence, process, crude analysis, eighth theorem. It deals mainly with processes and data to be handled in these processes and there is no a specific technique for the design.

Notation used textual, mathematical, matrix.

## 23.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | high |
| Ease of use | low |

Additional comments

Langefors Algebra assumes that an information system is designed to handle such functions as collecting, storing, processing and displaying of data, which implies that the information system grows in a way dependent on the development of data processing

machinery. It provides low level design considerations, difficult mathematical notation. The importance of methodological concepts lies mainly in its influence on other successful work such as: ISDOS and real time systems. The e-record represents e-messages of e-concepts which will have a type design based on e-message type, but requires still further decisions about representation, and further schema design; ie. choice of value domains, their representations say size, picture, to use Cobol terminology. It is not clear whether e-records will be stored directly or separately or will be embedded in large records or structures.

The precedence analysis concept is a most practical tool being adapted in some form by most of the system development methodologies. Petri-net which is the most active current area of research may be regarded as a dynamic model of precedence analysis.

Langefors stresses the early stages of system development and establishing algorithms for solutions to specific problems rather than a formal description of a system development methodology. The relativity principle mentioned is also appealing for the data processing community which is "every system which is subject to influence from its environment is a subsystem of some large system and every system part is potentially a system".

## 23.6 REFERENCES


1. LANGEFORS B, (1973).

2. LANGEFORS B, (1981).

3. LANGEFORS B, (1982).

4. TEICHROEW D, (1971).

5. GRINDLEY C B B, (1972).

6. COUGER and Knapp, (1974).

## 24.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | MASCOT |
| Full name | Modular Approach to Software Construction, Operation and Test. |
| Author(s) and institution(s) | Jackson K; TRE Malvern. |
| Date of first reference | 1976 |
| Application field(s) | embedded, real time. |
| Life cycle stages | requirements analysis<br>functional specification<br>structural design<br>detailed design |
| Software support | data dictionary<br>analysis and checking<br>detailed design aids |
| Development status | in use |

Comments

Software comprises a "Kernel" which contains a schedular, intrupt handler, system clock, error handler, process synchronisation, monitor, organiser and a driver for input/output devices and a link driver. The functions specified are represented by circles which enables the developer to postulate data flow network from

input through to output. It is then processed by a set of processors on the way, and in that the processing may need to remember internal information as the journey proceeds. Software development has three phases: overall software design, detailed software design, implementation and test.

## 24.2 LIFE CYCLE MODEL

- Divide the system into subsystems,
- Specify activities within each subsystem, and then draw an ACP diagram,
- Produce specifications for each activity,
- Specify a detailed design structure (of access, initialisation and point procedures).

## 24.4 SYSTEM MODEL

### Concepts used in system specification and design

ACG diagram (activity channel pool diagram), detailed diagrams of structures, connection diagram, source, sinks, data paths.

### Notation used graphical

## 24.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | average |
| Ease of use | low |

Additional comments

Formulation of software structure and methods for designing, implementing and testing using the formulism of the network diagram is similar to SREM. MASCOT has the same goals as HOS, WALMADE and SREM, but MASCOT and WALMADE may be regarded as complementary to eachother. GAMMA misses out, in common with a lot of others, is that there is no top, and no way of getting an overall picture of the scheme, and if anything, that the network diagram really does. There are run time facilities provided within the Kernel, specially for scheduling and synchronisation aspects, and monitoring facilities which permit a detailed snapshot of realtime events. MASCOT is concerned with those real time systems where whole data is in main memory. One can map a channel or a pool into the backing memory, but the problem of memory swapping of activities is not mentioned.

MASCOT has been used on whole range of machines from INTEL 8080 microcomputer with 700 bytes, to IBM 370. The majority of implementations for defence have been in mini-computers.

MASCOT use Pascal and other languages and recently a Pascal

implementation is reported at Imperial college. There is a strong 1-1 relationship between the user requirements and software modules, in that it is comparable with SADT. Petri-net work also seems to contributed to MASCOT development, because Petri-nets are concerned with the flow of tokens and controls, whereas MASCOT is trying to de-couple entry from that, and is based on data.

MASCOT supports languages which recognises activities, channels, pools and messages.

## 24.6 REFERENCES


1. ASWE, (1979).

2. DoI, (1981).

## 25.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | NIAM |
| Full name | Nijssen Information Analysis Method. |
| Author(s) and institution(s) | Nijssen G M and others; Control Data B.V, Netherlands |
| Date of first reference | 1982 |
| Application field(s) | dp, tools, A.I, |
| Life cycle stages | requirements analysis<br>functional analysis<br>structural design<br>detailed design<br>programming |
| Software support | data dictionary<br>analysis and checking<br>detailed design aids |
| Development status | in use |
| Comments | |

NIAM supports the concepts of: functional decomposition, data decomposition, interface definitions, sequence control flow, formal program verification. Its main emphasis is on information analysis, makes an inventory of all functions of the target

system, decomposition of these functions to a level where information flows and transformations become clear, every level of decomposition is specified in terms of a hierarchy of IFD's, functions and constraints are described formally. All the results are expressed in a formal language.

NIAM does not support system development in a strict system development life cycle, but adopts the idea of a framework in which three main components are distuinguished; the object system, information system, and the environment. There is no feasibility study phase, and the methodology is based on a binary relationship approach, it does not consider concurrency problems.

## 25.2 LIFE CYCLE MODEL

- Specification of a conceptual model,
- Function decomposition,
- Specification of sentence model,
- Decomposition of sentence type,
- Specification of constraints,
- Specification of subtypes,
- Express the above in a graphical notation (IFO's),
- Specification of population and set oriented diagrams,

- Express IFD's in a conceptual grammar,

- Formulise the constraints.

## 25.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

abstraction system, object system, conceptual grammer, information base, and meta-conceptual grammer.

Notation used graphical

## 25.4 SYSTEM MODEL

Concepts used in specification and design

lexical objects (string for representing real objects), non-lexical objects (object in the real world), idea, subtype (objects sharing a property), constraints, identifier, set, total role, information flow, bridge type, lot, non-lot.

Notation used textual (RIDL) and graphical.

## 25.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | average |
| Ease of use | above average |

## Additional comments

The following tools are supported: enforcer to enforce the rules of grammer; interpreter between environment and enforcer; collector of request from the environment; presenter of the requirements from the environment. There are some similarities with ACM/PCM; both gives equal emphasis on the analysis of structural and functional similarities; both employ the principle of abstraction. The abstraction used in ACM/PCM helps in managing the complexity of the overall system which is also true in NIAM.

NIAM is remarkbly in its resemblence to structurel analysis, and its philosophy is based on a perception of the real world in terms of: object system, abstraction system, conceptual grammer, information base and environment.

The authors claimed that they have met the objectives of the designed system by defining its concepts on the principle that all functions performed by an information system can be completely described by a conceptual grammer which is the only communication between the user and information system. NIAM is a process oriented and is concerned with the information flows between user and information system and between the processes within these systems, but it makes a little mention of the organisation structure. NIAM maintenance of analysis and design is done in a specific language RIDLE, definitions and changes are stored in ISDIS which acts as a data dictionaty which stores and

updates the conceptual statements of NIAM, shows the implications of the specified conceptual grammer, and compile the conceptual grammer to make it suitable for enforcer.

NIAM work products are: knowledge based integrated software information; generating system documentation; cross references, reports and process description. Representation schemes used are: IFD, ISD, dictionary, and formal specification language.

Completed system is validated against original requirements by the acceptance test ie verification of: IFD's and constraint definitions; walkthoughts; impact of change in specification.

Investigative, creativity and feasibility aspects are not dealt. It also do not deal the technical design of data systems and is more mechanistic due to emphasis on reality modelling. Its theoretical base is mainly from computer science and linguistics and contains a high data complexity. Data structures represents a conceptual view of the database and may be implementated by using ISDIL, but little information is given how these diagrams to be drawn, and what happens when the developer faces naming problems. NIAM excludes the decision for computerisation strategy, feasibility, implementation.

## 25.6 REFERENCES


1. VERHEIJEN G M, (1982).

2. FREEMAN and Wasserman, (1982).

## 26.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | PRISMA |
| Full name | Planning and Requirements Analysis for Information Systems (a modelling approach) |
| Author(s) and institution(s) | Laagland P T J and others; Klynveld, Kraayenhof and Co. Netherlands. |
| Date of first reference | 1981 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design |
| Software support | |
| Development status | under development |
| Comments | |

It seems to be stronger on organisational and requirements analysis aspects and weaker in system specification, design and implementation.

## 26.2 LIFE CYCLE MODEL

- System planning phase (information structure plan, information system plan, feasibility study),

- Development of business model (identify scope of: business system and subsystems, information needs, data stores and analyse the data flows),

- Development of information model (identify the scope of information system, definition of information functions, usage of data stores),

- System development (system specification, design and implementation).

## 26.3 ENVIRONMENT MODEL

Concepts used to describe system environment

environmental units, generates, generated by, receives, received by, GSD flow etc.

Notation used graplical

## 26.4 SYSTEM MODEL

Concepts used in system specification and design

N-square chart, business function matrix, organisation units.

Notation used matrix, textual

26.5 COMMENT

Completeness low

Economy average

Ease of use average

Additional comments

PRISMA may be regarded as a set of two methods: a business system and a model of the information system. A structured model description language together with modelling concepts is described. To describe business model, business functions are identified and linked by data flows of goods and services with or without data stores, and the descriptions of data stores and flows are ambiguous. To describe the information model, information functions of data generation, enquiry update and recording are defined together with their sources and destinations. Information system modelling is very briefly defined and gives static and mechanistic view of the business reality. Methodology is a set of N-square charts, a binary relationship model. The transition from business model to information model is not demonatrated. The binary relationships to describe data stores becomes lenghty, and this can be solved if n-ary relationships are also used. It provides no rule as how to get output from given inputs. In some respects it may

regarded similar to NIAM, BISAD, and ISAC. Its environment modelling and physical levels are not clear. N-square mtrix may become unmanageable for large systems. PRISMA has not yet reached to a state where it should merit as a complete methodology.

## 26.6 REFERENCES

LAAGLAND T M, (1982).

## 27.1 METHODOLOGY SUMMARYe

| | |
|---|---|
| Short name | PSL/PSA |
| Full name | Problem Statement Language and Problem Ststement Analyser |
| Author(s) and institution(s) | Teichroew D; University of Michigan |
| Date of first reference | 1969 |
| Application field(s) | dp and embedded |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design |
| Software support | analysis and checking<br>prototyping<br>detailed design aids |
| Development status | in use, though under continual development |

Comments

It provides a model of information systems, present and future needs of the organisation, requirements specification and analysis, a basis for making decisions in the current and subsequent stages of the system development process. It provides a basis for integrating and extending automated design

methodologies.

27.2 LIFE CYCLE MODEL

- Study and describe the current system,

- Improve the current system,

- Propose an improved system,

- Divide the proposed system into subsystems,

- Identify required information,

- Express the requirements in PSL,

- Specify a computerised database,

- Specify capability to display data to users,

- Specify the capability to check consistency and completeness,

- Specify the capability for analysis and evolution,

- Specify decision making aids.

27.4 SYSTEM MODEL

Concepts used in system specification and design

entity name, attribute name, attribute value, cardinality, identified by, consists of, security, relation, synonyms.

Notation used formal textual notation (PSL)

## 27.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Ease of use | average |
| Economy | high |

Additional comments

It contains a number of tools which fall into three categories, namely: report generator, database enquiries, and completeness checkers. The report generators are the largest group and consist of a collection of programs that traverse the database and prints out various parts of it in a variety of ways.

There is not a fixed way of using PSL/PSA as demonstrated by Teichroew and others by using the tool kit with various procedural methodologies, and also they claimed that it is possible to use the tool kit to support any methodology, and aids in the organisation of the large project teams. It is an evolving system, do not have the capability for the modelling of conceptual schema. It incorporates three important concepts: all information of the target system is to be kept in a computerised development information system database; processing of this information is done by the computer to a maximum extent; and specifications are to be given in "what" and not in "how" terms.

The automated analyser, PSA, operates on the database of development information that has been built up out of PSL inputs.

It provides reports indicating changes to the development database. It also performs some analyses of information in the database to indicate such things as gaps in the specified information flow, unused data objects and the dynamic behaviour of the target system.

## 27.6 REFERENCES


1. DoI, (1981)

2. TEICHROEW D, (1976)

3. KAHAN B K, (1976)

4. LARCHER, (1980).

## 28.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | REMORA |
| Full name | The REMORA methodology for information systems design and management |
| Author(s) and institution(s) | Rolland C and others; University of Paris |
| Date of first reference | 1982 |
| Application field(s) | dp, tools, embedded |
| Life cycle stages | functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>detailed design aids |
| Development status | under development as a research project |

Comments

Information system development process is completely assisted by an automation system organised around the pair (man, automation). A set of models have been proposed, describing the system from the conceptual description through implementation. The approach may be regarded as structuralist type and not functional type, corresponds to the development of databases, DBMS and build the

system through the definition of its structure, and is claimed to be more complete than most of existing approaches. Information system development process is split up into two steps: (a) conceptual step, and (b) internal step (includes the technical aspects of the solution ignored in the first step and takes into account the participation of users). In both of these steps a solution is obtained through modelling using theoretical concepts, methods, logical rules and then describing the model by using a formal language. There are eight phases of the modelling: static, dynamic, using ISDEL language, module and trigger concepts, control-integrate document system functions to add the application specifications from the process synchronisation subschema to the previous ISDEL statements; derivation of a logical data schema from static data subschema; and specific logical data schema.

## 28.2 LIFE CYCLE MODEL

- Choose a typical relational model,

- Represent each class of phenomena and each class of associations by relations,

- Specify each property of a class of phenomena by an attribute of relation,

- Specify each category of the class of phenomena by its relation type ( (c-object, c-operation, c-events),

- Describe the above relation type in temporal normal form.

## 28.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

object, event, trigger, operation, modify, ascertain. A state change expresses the passage from one state to another; event is a state change and triggers determined operations; assertion is an assertion between event and one or more objects; and a trigger is a an association between event and operations.

Notation used graphical

## 28.4 SYSTEM MODEL

Concepts used in system specification and design

relation type and description, relation specification, list of all attributes, integrity constraints, domains.

Notation used formal textual (ISDEL) and graphical

## 28.5 COMMENT

| | |
|---|---|
| Completeness | high |
| Economy | high |
| Ease of use | above average |

Additional comments

A number of notations are used, each to describe a specific level. Schema Programming and schema like languages describe the static and dynamic levels of conceptual schema and also a specific graphical notation may be used for dynamic subschema. The relationship between ISMS, computer aided system and the pilot is obscure. The DBMS and SOCRATE and syntax should have been defined. The project management is based on a formal model derived from conceptual schema which is widely recognised for the design of large and complex information systems employing automatic data processing, network communication and real time responses. Automated tools are not defined, however conceptual design specification for these tools is comprehensive. REMORA deals logical design, storage structure, access, program decisions, but their description is not provided in relation to development phases rather described in a mixed up format, and some time becomes ambiguous. REMORA contains ideas similar to DADES, ISDOS, NIAM and Systematics. Events and triggering condition specifications are difficult to apply. It contains a static schema and a dynamic schema of ACM/PCM, and identifies two levels of abstractions ie. conceptual and logical, which respectively deal with semantic representation of the real world and a definition of the technical solution. Validation and simulation tools include CAD with original architecture, the design process is controlled by an automation PILOT which coordinates man-tools interventions. Tools perform: control,

integration, simulation and documentation. In the dynamic schema the border of the universe of discourse may be difficult to recognise because of schema structuring style. REMORA seems to be practical, well justified, reasonably complete and computer aided, and may be considered as a promising candidate for future development. Investigative/diagnostic and creativity aspects are not dealt, and use ANSI sparc to specify levels of abstraction. There is no enough information about: empirical experimentation, the specification of the project report, roles of users and developers, the methods to obtain static and dynamic subschema. The project covers both the aspects, academic and industrial while some definitions are not very clear such as event = state change of an object. It does not support management aspects, and supports: consistency checks, implementaion, function/data decomposition, interface definition, data flow, sequence control flow, currency and formal program verification.

## 28.6 REFERENCES


1. ROLLAND C, (1982)

2. FREEMAN P and Wasserman, (1982).

## 29.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | Sysdoc/Systemator |
| Author(s) and institution(s) | Aschim F and others; Central Institute of Industrial Research, Norway |
| Date of first reference | 1982 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>prototyping<br>detailed design aids<br>code generation |
| Development status | in use |

Comments

Sysdoc is an information design methodology which contains a high level language SYSDUL, and Systemator is a software tool which provides computer aids to all phases of information system development. It also contains modules: to provide designs from requirements; storing; modifying; documenting; and analysing system requirements. It has an analysis capabilty of PSL/PSA and

SREM. Methodology may be regarded as data oriented, the results of analysis are stored in a data dictionary supported by SYSTEMATOR, which translates the design results in a prototype implementation, generating a schema and programs to manipulate the database. Sysdoc concentrates on analysis, user interface specification and design, and technical design. It does not support strategic planning and feasibility study.

## 29.2 LIFE CYCLE MODEL

- Identify the users of the proposed system,
- Prepare a list of the entities involved,
- Specify relationship types and construct a data model,
- Compile a list of inputs and outputs,
- Define the entity types and associated data elements.

Sysdoc covers the design and implementation phases as follows:

- Form the definition of the problem and list the classes of end-users,
- Interview users and management to find: the list primitive (preliminary) entity types; relationship types.
- For each transaction type specify input and report lists,
- Specify the occurrences of representations of each entity type

identified by end-users and computer system,

- Describe each transaction type giving screen data content, screen data layout, and processing rules as abstract programs in SYSDUL,

- Generate a prototype runable system by generating a primitive database design, a primitive physical design, appropriate job control language (JCL), generating application programs in Fortran or Cobol from the abstract programs, evaluate and modify the primitive system, and improve the physical design.

## 29.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entity type, relationship type, data element type (attribute)

Notation used graphical

## 29.4 SYSTEM MODEL

Concepts used in system specification and design

conceptual data modelling technique is used for requirement specification, elementary entity tyes, data element types, relationship type, mixture type, description of: processing rules, end users, data contents, data structures, screen layouts, time and volume, transactions.

Notation used tabular (forms) and textual (SYSDUL language)

## 29.5 COMMENT

| | |
|---|---|
| Completeness | above average |
| Economy | average |
| Ease of use | low |

Additional comments

Conceptual data modelling is the main technique used for requirement specification in consultation with users, and the corner stone of Sysdoc is a specification of requirements, a set of tools. No formal language has been made in Sysdoc to provide a functionally oriented description (other than abstract programs). It generates a correct database design but no emphasis is given to optimal database design. End-user particaipation is not comprehensive except in the discussion of requirements and the data model. The main result of Sysdoc analysis is a conceptual model which can be expressed as submodels, the models and submodels are represented graphically and input directly to the dictionary. The transactions are expressed in a VHL SYSDUL with entity names used in the data model, and the user interface is defined by describing the screens and user dialogues. Sysdoc makes no reference to the design of any network of terminals and processors, privacy, security, integrity, recovery, conventional file design and distribution of processing and storage. It does not provide a detailed design (which depends on prototype), nor any manual

techniques to tranform the conceptual data model into a logical database design.

## 29.6 REFERENCES

ASCHIM F, (1982)

## 30.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SREM |
| Full name | System Requirement Engineering Methodology |
| Author(s) and institution(s) | Alford M; TRW Defence and Space Systems Group |
| Date of first reference | 1975 |
| Application field(s) | embedded |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design |
| Software support | data dictionary<br>analysis and checking<br>prototyping<br>detailed design aids |
| Development status | in use |
| Comments | |

Requirement phase is under-development. SREM may be labelled as a methodology, because it includes necessary tools, techniques and procedures. It is applicable throughout the system development process. SREM has been used for specifications; and

no software has been developed from these specifications, the reason may be long lead time associated with projects. In the case of tools REVS has approached sufficient maturity to be used in real projects in terms of things like run time, memory size and availability on some computers.

## 30.2 LIFE CYCLE MODEL

- Translate and interpret system specifications to produce flows and data messages,

- Complete functional requirement details (inputs, outputs, processes),

- Develop functional models and note model inconsistencies,

- Allocate performance requirements in relation to paths and timing and test,

- Develop candidate algorithms.

## 30.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

entity, event, relationship, attribute, stimulus.

### Notation used graphical and tabular

## 30.4 SYSTEM MODEL

Concepts used in system specification and design

stimulus, response, processing path, class, optional word, structure and structure sequence, file, source, synonym, version, associate, connect, compose, destroy, create, sub-net, validation path.

Notation used graphical and textual

## 30.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | low |

Additional comments

SREM supports step by step approach, and top-down development, while considerable design freedom is provided. SREM emphasises a separation of concerns between static consistency checks, functional simulation and performance prediction. Central to the system are support software tools, project database. Support tools include: RSL analysers, static completeness and consistency checkers, simulation generation aids, performance predictors, and interpretative graphics for manipulating R-nets. The support system is 45K Pascal statements, and 10K Fortran statements for

the graphic support. Four basic aspects are claimed to be achieved which are: requirement statement language (RSL); requirement engineering and validation system (REVS); formulism by defining terms based on extension of group model; and the procedures and milestones. It uses ISDOS data management system. SREM with SREP tools is very powerful to handle Ballistic missile problems but has no ability to handle data processing/file processing and man-machine interaction. It deals first identifying the interfaces, the things one deals with and the messages that cross the interface both in and out, which corresponds to the Jackson approach of identifying inputs and outputs. Its weakness lies in its abandonment of traditional functional decomposition, its inability to cater for systems with human operators in the process, loop and its heavy reliance on sophisticated computer facilities. Prescribed workproducts are; requirements definition, software requirements specification, documentation from queries to requirement database. Quality assurance methods supported are: design reviews, automatic data flow analysis of R-nets, static consistency, completion checks on requirement database. Completed system is validated against original requirements by dynamic validation of performance requirements using simulation and post processing. SREM also addresses management aspects, such as: project, technical, team and system evolution.

## 30.6 REFERENCES


1. DoI, (1981)

2. LARCHER, (1980)

3. TEICHROEW D, (1972)

4. FREEMAN P and Wasserman, (1982)

## 31.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | Solvberg |
| Full name | A draft proposal for integrating systems specification model |
| Author(s) and institution(s) | Solvberg A; University of Trondheim |
| Date of first reference | 1982 |
| Application field(s) | dp, science/engg, Expert, and general |
| Life cycle stages | functional specification<br>structural design<br>programming |
| Software support | |
| Development status | under development as a research project |
| Comments | |

Like EDM, Solvberg does not deal with problem study phase. The levels of abstraction dealt are: external, conceptual, logical and physical. User interface design is treated explicitly. No concrete tools are specified, and neither deals the problem study phase. There is no practical experience of its use and employs several existing techniques. Model has elements from both data and process, contains entity-relationship structures.

## 31.2 LIFE CYCLE MODEL

- Terminology development phase: concepts and environment are classified,

- Processing specification phase,

- Information resource definition: necessary contents of operations on information system are specified,

- Error analysis,

- Responsibility analysis,

- Process resource allocation,

- Man-machine interface design,

- Resource management system design,

- Operational design to determine the procedures for restart and initialisation,

- Database design,

- Program structure design.

## 31.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

entity-relationship model: entity type, connection type, events, change (active and passive), tasks.

Notation used graphical and mathematical

## 31.4 SYSTEM MODEL

Concepts used in system specification and design

task, message, interface, data store, task responsbility, message allocation to performance resources.

Notation used textual, graphical pseudo-code

## 31.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | low |
| Ease of use | average |

Additional comments

Special object types are introduced for functional specifications. Data types, stores, messages are the means of specifying the contents and structures of data transmitted and stored in the system. Data type objects are defined independent of time, while messages consist of data having time-limited -existence.

The methodology does not support boundary specification, derivation rules, and no roles of users are defined and any computer aid. It mentions the activity of identification of

system objectives and constraints including conflicts of interest among user groups; makes reference to decision analysis, activities studies, user requirement definition, but do not include concepts or models and tools.

This can be efficiently implemented or used in an organisation if these aspects integrated with automated design tools. The presentation provides a broad idea of the most of concepts and techniques used in modern information system design (data flow diagrams, control flow diagrams, forms specifications and application control diagrams).

## 31.6 REFERENCES

SOLVBERG A, (1982)

## 32.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | Systematics |
| Full name | Systematics: a new approach to systems analysis |
| Author(s) and institution(s) | Grindley C B B; Urwick Diebold Ltd. |
| Date of first reference | 1966 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>detailed design |
| Software support | data dictionary (manual) |
| Development status | published, probably still under development and in limited use |

Comments

An output is defined in a systematics sentence consisting of a trigger, an output item and an identifier. A set of dictionaries is specified, a derivation dictionary which describes the formula for each derived output item, an input dictionary providing the entries of all input items involved, and an identification dictionary for the specification of all primary identifiers.

Methodology is mostly suitable for control systems.

## 32.2 LIFE CYCLE MODEL

- Specify outputs,

- Specify main trigger conditions,

- Specify subsidiary trigger conditions,

- List all the contents of the output set,

- List the data sets,

- Specify derivation formula,

- Design the contents of the input set,

- Specify and construct identification and derivation chains.

## 32.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

given set, derived set, trigger, relationship.

Notation used graphical and set theoretic

## 32.4 SYSTEM MODEL

Concepts used in specification and design

item, state, identifiers, derived items and given items, information set, effective time, trigger, descrete and continuous identification and time substitute.

Notation used tabular, textual, graphical

## 32.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | average (some time clumsy) |

Additional comments

It does not support database/files in the model, identifier concept is not clear, statistical information is ignored, notation is clumsy and insufficient, error and consistency mechanisms are not clear, absent items are ignored ie. whether a set contains all items of the set or only those given to the set, do derived items exist other than as outputs. The concept of sequence or ordering is missing which some time leads to illogical identification (derivation chain ordring is absent). If a series is to be outut it is not clear in which order it should be produced, similarly if a series of derivations is to

performed then its order should have been mentioned, how to identify items within a series eg. maximum, minimum, mode etc., and how to count the population of a series.

Dictionaries are limited in scope and there is no provision of type, domain, frequency, volume etc; the decision tables are also inadequate as described in KING P J H (1967), the treatment of time is unnecessarily clumsy, identification chains are ambiguous and difficult to construct, determination of given and derived items is manual and may not be suitable for systems with extensive data. The language is based on three propositions: (a) certain items are given to the system as relatives being input together; (b) all outputs are triggered by an input; (c) all items have an effective time for establishing relationships. This shows that the Systematics describes: how system components are described and identified, what relationships explored and how they are derived. It may regarded as a major step for providing formal concepts for the development of modern information system methodologies.

## 32.6 REFERENCES


1. GRINDLEY C B B,(1975)

2. KING P J H (1967)

3. GRINDLEY C B B, (1973)

4. GRINDLEY C B B, (1966)

5. TEICHROEW D, (1971)

## 33.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SDM |
| Full name | A Semantic Database Model |
| Author(s) and institution(s) | Hammer M and others; Massachusetts Institute of Technology |
| Date of first reference | 1981 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design |
| Software support | |
| Development status | under development |

Comments

It serves as a formal specification mechanism for describing the meaning of database, provides a precise means of documentation and communication medium for database users, provides a basis for a high level semantic based user interfaces to a database to non-programmers. It also provide a foundation for supporting the effective and structured design of database-intensive application systems. It does not mention any facility for boundary specifications.

## 33.2 LIFE CYCLE MODEL

1 - Write all items of the system.

2 - Determine class name. and names of the members of the class.

3 - Describe whether the class is a base class or nonbase.

4 - Take each item of the class and specify the items mentioned as follows: the value class, may not be null, not changeable, member attribute name, class attribute name, multiple or single valued, exhaust the value class, type, member attribute interclass relationship (inverse or match), mapping, derivation (ordering, boolean, recursive combinations, collections of members, sub-value, and set operators, exponential, max, min, average, sum).

## 33.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entity, relationship, attribute.

Notation used textual

## 33.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | low |
| Ease of use | low |

Additional comments

SDM is a database description model and describes the database in terms of the kind of entities that exist in the application environment, the classifications and groupings of these entities and structural interconnections among them. It does not deal with the mechanism of defining inputs and outputs and neither the notation is simple and usable by an average developer. Moreover the notation becomes clumsy and tedious due to several subdivisions into classes, subclasses.

## 33.6 REFERENCES


1. DoI, (1981)

2. HAMMER M and Mcleod, (1981)

## 34.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SDLA |
| Full name | System Descripter and Logical Analyser |
| Author(s) and institution(s) | Knuth E and others; Academy of Sciences Budapest. |
| Date of first reference | 1982 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>detailed design |
| Software support | analysis and checking<br>prototyping<br>detailed design aids |
| Development status | under development |

Comments

It is similar to PSL/PSA, and Chen entity-relationship model. It has been developed in a cooperation with ISDOS project and is claimed that it is being used throughout the development process. SDLA may be regarded as a set of fundamental tools for information system design, but do not constitute a methodology in

itself.

## 34.2 LIFE CYCLE MODEL

- Environment specification,

- Formation of top-down hierarchic structures from data descriptions of above,

- Input/output specifications ie. the data structures produced or consumed by the software components,

- Specify system functions (processes, procedures, routines, functional modules; and function input data, output data, and other data utilised by the function),

- Implementation design (this phase is not covered because it is assumed that the logical design should be fine enough to tell what is fundamental and functional.

## 34.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

entity, relationship, attribute.

Notation used textual and some graphical

## 34.4 SYSTEM MODEL

Concepts used in specification and design

information structures to be stored (set, list, elements, type, inclusion and relationship with other data, derivation relationships, initialisation, update and usage, activisation and time conditions, restoribility constraints, integrity properties, security limitations)

- Input/output specifications,

- System functions (function input/output data, other data to be utilised, and the description of function).

Notation used graphical and textual

## 34.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | average |
| Ease of use | high |

Additional comments

The objects to be stored in the database is an instant of the abstract concept; objects are described by attributes; an abstract concept is described by its associated set of attributes. The actual set of objects as instances to a given

concept can always be considered as relative to subset of the cartesian product of the attribute value ranges. This approach gives original idea about a versatile tool, which would enable the developer to specify concepts they plan to use during the design process. The notation used is difficult and not formal. It is more process oriented, establishes required controls, revise logical inputs and outputs, define logical operations, supports man-machine interface, define functions, file/database, programs and modules.

## 34.6 REFERENCES

KNUTH E and others, (1982)

## 35.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SADT |
| Full name | Structured Analysis and Design Technique |
| Author(s) and institution(s) | Douglas T Ross; Softech |
| Date of first reference | 1974 |
| Application field(s) | embedded, dp, science and engg., o/p, tools and expert systems. |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design |
| Software support | analysis and checking |
| Development status | in use |

Comments

SADT is more concerned with functional decomposition of business activities emphasising on the presentation of data processing aspects in order to facilitate the thinking process of the developer, and a communication of the results to users. It mainly concentrates on study requirements and constraints, analysis of system functions, and representing them by models based on SADT diagrams. For double checking purposes the dual representation of the system is elaborated independent of

activity diagrams.

It is not used for software module design, because the constructs such as sequence, selection and iteration are missing in SADT. It is used in the planning analysis and general design phases, and use the techniques of Jackson, Warnier, and Constantine for detail design activities. It is a general purpose technique applicable to a wide range of problems and not only to computer applications. It was developed to provide a disciplined approach to achieve users understanding of his needs prior to a design solution. It did not evolve from design technique but by examining the problems associated with defining systems requirements.

## 35.2 LIFE CYCLE MODEL

- Specify the model or models of the required system showing the justification, activities and data that make up the system,

- Determine the needs for the new system,

- Functional description: what should be done to resolve the existing issues, needs and influences; identify all activities and data which is to be used,

- Realisation of the system: model showing the software architecture is used to present a structure to be used to identify software functions (activities and data), and also an organisation of software systems to satisfy the requirements,

- Requirements and design traceability: determine redundancy and overlap by cross referencing the models,

- View points: a separate SADT can be beveloped for each view point of current operations as future operations,

- Logical and physical models: specify the purpose of the model, and then decide which of the two models communicate most clearly with the intended audiance and provide best answers to questions.

## 35.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

subject, model (assumption is that a model 'M' is the model of a subject if 'M' can be used to answer the questions about the subject).

| | |
|---|---|
| Notation used | graphical |

## 35.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | average |
| Ease of use | average |

Additional comments

The process of design in SADT is the process of stating off with some current, but abstractly expressed, notation and refining it into greater and greater level of detail. What one produces is called a model in terms of structured analysis "a pyramid of diagrams" with boxes which themselves be broken down into further diagrams. The advantage of the technique is that, if a man-machine interaction aspect of a system is required then, give a complete model, start at the top and proceed down the levels until the first mention man-machine part is seen, and then follow it through (because the right section of the model is achieved). The establishing of a system development framework (a standard system life cycle) provides the users, developers, programmers a basis upon which a variety of software development tools, techniques and methods can realise their full potential. The principles of structuring are a combination of common sense and proven concepts, and each principle stresses a different aspect of organising and presenting information about a given system. SADT is an analysis and design methodology and focuses on how the analysis and design can be performed. The development of software systems is a necessary pre-requisite for the effective utilisation of SADT. Complementary analysis approaches are used to build on the activity/object duality of most situations. It does not provide a clear definition of system boundary, and emphasises on conceptual definition of users requirements. It then concentrates on functional analysis and the result of

requirements analysis ends at "what" and particularly "how it should be done", instead of what should be achieved in the environment. Activity diagram is similar to HIPO. SADT diagrams are not a concise form of expression and, especially in large systems, they may spread over hundreds of pages. Other structured approaches such as Gane and Sarson, Yourdon, Constantine and Demarco have more descriptive capabilities having diagrams, structured English, data dictionaries and decision tables etc.

SADT seems to quite rich in technical concepts and supports: function and data hierarchy decomposition, interface definitions, data flow, sequence control flow, concurrency and formal program verification. Work products of SADT are: model kit, node indexes, large schematic. Quality assurance methods are: author/reader cycle, structured walkthrough, automatic consistency checks, and graphical notation. The completed system is validated against original requirements by cross referencing: from notation or diagrams, walkthrough sessins with users. SADT also supports management issues; project, technical team, validate work products.

## 35.6 REFERENCES


1. DoI, (1981)

2. BREWER T, Report series number 11, (1979)

3. LARCHER, (1980)

4. INFOTECH, (March 1977)

## 36.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SD |
| Full name | Structured Analysis and Design |
| Author(s) and institution(s) | De-Marco T, Yourdon E, Constantine L, Myers, Gane and Sarson. |
| Date of first reference | 1974 |
| Application field(s) | dp, embedded, science/engg, o/p, tools |
| Life cycle stages | requirements analysis<br>functional specification<br>structural design<br>detailed design<br>programming |
| Software support | |
| Development status | in use and under continuing development |
| Comments | |

Method was introduced by Constantine (structure charts), then by Myers (composite design), and in an elaborated form by Yourdon and Constantine (1975) and Gane and sarson (1979).

SD covers mainly analysis and design but not strategy and feasibility, though some implementation techniques are suggested. As described in Yourdon (1979), that SD is not a methodology but

a set of process oriented techniques ie. data flow diagrams, structure diagrams, structured English, data dictionary and decision tables/trees etc. The analysis and design phases are broken down into well defined subphases. The completeness is checked by referring back to users. The basic aim of SD is to: produce maintainable documentation, reduce the size of the problem by a suitable partitioning, increase understandability by using graphic, and distinguish logical and physical design phases. DFDs play a central role in allowing the developers to demonstrate the model to users and the use of decision tables/trees.

## 36.2 LIFE CYCLE MODEL

- Initial study of the organisation, a detailed study of the organisation,

- Build a logical model,

- Define a minu of alternatives,

- Refine physical design.

## 36.3 ENVIRONMENT MODEL

### Concepts used to describe the system environment

DFD, data (its a general purpose methodology and the environment can be extracted from the above concepts). A technique

reflecting activities and data also reflects the reality.

Notation used graphical

## 36.4 SYSTEM MODEL

Concepts used in system specification and design

data flow diagrams, data store, data dictionary, logic of process, decision table/trees, structured English, tuples, relation, normal form, security.

Notation used textual and graphical

## 36.5 COMMENT

Completeness above average

Economy low

Ease of use above average

Additional comments

The data flow diagrams of SD are similar to SADT diagrams, except the decomposition is used loosely and the graphical representation includes data stores directly in the DFD's. The development of computer-based systems is a creative process, and there does not exist a unique soluton to all problems, and therefore, no structured method can be a substitute of intelligence, thinking and experience. However, a structured

method may make their absence embarrassingly apparent. SD design terminology is: apparent modules ie. handling input data; main data transform process; strength of processing activities within a module and coupling. The DFD's depicts the system into network of activities, inferences, origins, destinations, and data stores; fan-in and fan-out ie. number of superiors and subordinates; activity of DFD transforming input data flows to output; structuring of charts for the decomposition system into modules and a communication between them. Methodology does not provide any computer aid but the design procedure can be supported by PSL/PSA. The top-down development is attractive, and the criterian for terminating decomposition provides a management flavour and the concept of abstraction introduced by Dijkestra ie. the description of design concepts independent of hardware realities is also appealing. In the "Walston and Felix" study of the productivity of an IBM project which did not use a top-down approach had 196 lines per-man month; projects which did use it averaged 321 lines per-man month ie. an improvement of about 60%.

The looser syntax of SD data flow diagrams makes for easier visibility and understandability than SADT, but SD still suffers from its business data process origins, in that it does not provide full solution to the problems inherent in the real time systems. The work products of SD are for analysis ( structured specifications, data dictionary, mini-specification, state transition model): for design (specification for design, database

design, operational constraints, physical constrints); and for implementation (structured code). Representation schema used are: data flow diagrams, structure charts, data structure diagrams, finite state diagrams, decision tables/trees, and a program design language PDL. The technical concepts supported are: function and data decomposition, interface definitions, data flow, sequence control flow and concurrency.

## 36.6 REFERENCES


1. GANE C and Sarson, (1979)

2. DEMARCO T, (1979)

3. MYERS G J, (1978)

4. YOURDON E and Constantine, (1979)

5. FREEMAN and Wasserman, (1982)

6. LARCHER, (1980)

7. TEICHROEW D, (1977)

## 37.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SDS |
| Full name | Software Development Systems |
| Author(s) and institution(s) | Royal Signal and Radar Establishment<br>Malvern |
| Date of first reference | 1978 |
| Application field(s) | embedded |
| Life cycle stages | requirement analysis<br>functional specification<br>detailed design |
| Software support | analysis and checking<br>prototyping<br>detailed design aids |
| Development status | in use |

Comments

It can be used at all stages of a project development but is mostly used for specification and design. It may be regarded as a flexible tool which imposes very few restrictions and could be used to support a wide range of methodologies. It is currently available on ICL 1900 computers, and is currently being implemented on a wider range of hardware.

## 37.2 LIFE CYCLE MODEL

General guidelines, based on a top-down decomposition approach, using SDS are available but these are not given in the literature. Since it is intended to assist with a wide range of problems and approaches there is not a standard set of procedures for its use.

## 37.4 SYSTEM MODEL

Concepts used in system specification and design

components, categories, requirements, key words, standard field.

Notation used graphical

## 37.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | low |

Additional comments

SDS is of value to large projects, and the lack of explicit configuration control mechanisms may limit the usefulness of SDS where a large number of variants are to be produced. There is a considerable amount of work required by any user organisation

before any useful results can be obtained. In addition to the facilities for updating the database model, a number of checking tools are available including a query language. A number of completeness and consistency checks are provided from simple name checks to checks on hierarchic consistency, and also further checks may be performed by using query language facilities. Project planning and control facilities are also provided.

## 37.6 REFERENCES

DoI, (1981).

## 38.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SARA |
| Full name | System Architect Apprentice |
| Author(s) and institution(s) | University of California |
| Date of first reference | 1978 |
| Application field(s) | embedded, science/engg., dp, o/s, tools |
| Life cycle stages | requirements analysis<br>functional specification<br>detailed design<br>programming |
| Software support | analysis and checking<br>prototyping<br>detailed design aids |
| Development status | in use |

Comments

It gives more emphasis on implementation and less on analysis and design. However, System design stage supports functional decomposition, data decomposition, interface definition, data flows, sequence control flow, concurrency and program verification, and consists of a package for semantic, syntax and consistency checking. It has been implemented on Vax-Berkley,

Unix.

## 38.2 LIFE CYCLE MODEL

- Preparation of requirement document,

- Preparation of functional analysis specification or description,

- Preparation of design models: structural data description, behaviour (graph model of behaviour), module interface,

- Preparation of implementation document: concurrency and parallelism, sequence control flow, formal program verification,

Initially a control flow model of the system is constructed and investigated by analysis (searching for potential deadlocks, liveness problem etc) or a simulation (ie token movement around the graph). Then a data flow model is constructed by specifying processes and data structures, which can in turn be simulated by linking control model to processes. High level system models may be decomposed into lower level subsystem models for further investigations or constructed from the predefined models resident in a library. When the complete set of models is build and validated, these forms appropriate specifications for implementation activities and provide benchmarks for unit and integration test.

## 38.4 SYSTEM MODEL

Concepts used in system specification and design

module, sockets, interconnection, control flow, data flow, interpretation dynamics.

Notation used textual and graphical

## 38.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | low |
| Ease of use | average |

Additional comments

SARA does not describe management aspects, boundary specifications, system evolution or version control. The basic objective is to build various models of the proposed system and its components along with the models of the corresponding environment. Staticaly-checkable attributes and constraints can be specified as the communication parts between various building blocks. The main automated tools are: mark graph analyser, simulator generator, compiler and consistency checker. The use of the tools is limited in real world enviroment because the conceptual schema is completely ignored by SARA. It supports semantic and syntactic consistency checks of the requirement

documentation, module interface definition, behaviour models, interactive simulation, evaluation using the test environment, control flow analysis of the test environment, automated tools. It provides automated support requirement and design specification documents, testing and checking and also an optional tool for simulation/prototyping.

## 38.6 REFERENCES


1. DoI, (1981)

2. FREEMAN P and Wasserman, (1982).

## 39.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | SDL |
| Full name | Specification and Description of Logic-process |
| Author(s) and institution(s) | CCITT |
| Date of first reference | 1976 |
| Application field(s) | embedded, science/engg., telecommunication |
| Life cycle stages | functional specification<br>detailed design |
| Software support | analysis and checking |
| Development status | in use |

Comments

It is used in telecommunication and switching system. Despite some claims that it is used as a specification tool, there is a strong indication in the form of solutions that it is, in reality, applicable to bottom level design only. Data abstractions and interface representation are not handled.

## 39.2 LIFE CYCLE MODEL

No methodical procedure for using the methodology is laid down in the CCIT guidelines, but the developer has to specify his own procedure depending on the nature of the problem. The inherent nature of the notation leads to the solutions expressed in terms of state machines. Specification and design are not separated, because the separation of development concern is not emphasised by SDL insofar as it expresses the behaviour of a system by means of an operational (state machine) model.

## 39.4 SYSTEM MODEL

Concepts used in system specification and design

functions of system, functional block, process, signal, communication path, level of functional block, state machine, inputs/output, actions and decision.

Notation used graphical and textual

## 39.5 COMMENT

| | |
|---|---|
| Completeness | low |
| Economy | average |
| Ease of use | not known (being used only in telecomm.) |

Additional comments

SDL is applicable to real time concurrent processing, and could be used for process control and military applications in addition to its market in telecommunications. It can describe the software carried in a multi-processor or distributed processor environment. SDL is not suitable for database systems or complex sequential numeric algorithms (ie. not applicable out of those concurrent processes applications in which component processes are simple in function). It does not provide any specific support for management functions, and is not suitable for database systems or complex numerical algorithms.

## 39.6 REFERENCES

DoI, (1981)

## 40.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | TAT |
| Full name | Transaction Analysis Technique |
| Author(s) and institution(s) | Larcher J; The Plessey Company Ltd. |
| Date of first reference | 1980 |
| Application field(s) | embedded and data processing |
| Life cycle stages | requirements analysis<br>boundary specification<br>functional specification<br>detailed design |
| Development status | published but probably little used |

Comments

TAT provides maximum attention on requirements analysis. The main function of TAT is to define the problem for the buyer, user or developer in a complete and precise way.

## 40.2 LIFE CYCLE MODEL

- Establish the requirement base line,
- Construction of a logical model,
- Validation of the logical model,

- Analysis of the external interfaces.

## 40.3 ENVIRONMENT MODEL

Concepts used to describe the system environment

trigger, event, object, transaction.

Notation used textual and graphical

## 40.4 SYSTEM MODEL

Concepts used in specification and design

responses, events, transactions, stimulus/trigger, system condition or time trigger, constrints, DFD, thread diagrams, existence, property, independence, modifier, subsetting, operation.

Notation used graphical and textual

## 40.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | low |
| Ease of use | average |

Additional comments

It takes into consideration the customer satisfaction but does not provide any method how the user will be involved throughout the system development. It supports a cooperation between requirement analyst and customer in analysing the operational requirements, specifies each required system action, so that these actions may be used to map the system requirements on to the system design. Grouping of documents is a precise and manageable way of describing different components of the system e.g group A: project context; group B: system facilities; group C: constraints; group C: functional operations; group E: man-machine interface. Techniques used are mainly DFD's, complex and inconvenient thread diagrams, which is the key concept and shows, how the individual transactions are liked together in time, to achieve the overall system objective. It is a graphical equivalent of a verbal walkthrough of data flow diagrams, and fills the place taken at the detailed programming level by the flow chart. A thread corresponds to an event based coordinated sequence which may be executed partially in parallel. These activities are performed by the people within the development, in order to progress a particular piece of work through the organisation and thus assisting the overall objective of the organisation.

## 40.6 REFERENCES

LARCHER, (1980).

## 41.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | TAG |
| Full name | Time Automated Grid |
| Author(s) and institution(s) | Myers D H; IBM |
| Date of first reference | 1966 |
| Application field(s) | data processing |
| Life cycle stages | requirements analysis |
| | functional specification |
| | detailed design |
| Software support | data dictionary |
| | analysis and checking |
| | prototyping |
| Development status | used but obsolete |

Comments

TAG is an interactive tool, its function is to develop an integrated system flow and to maintain that integration, no matter how many changes or how much additional data the user introduces. This is a third generation technique developed as a manual system and later automated by IBM in 1966.

## 41.2 LIFE CYCLE MODEL

- Specification of the activities around which the development of the new system begins, these details are then coded in an input/output analysis form and then reviewed by the developer,

- The automated analysis will produce the errors in the data which must be corrected by the developer,

- Assigning the priorities of the system outputs, where all outputs for a given time period are received together. With reports created by the TAG, the developer redefines the time intervals at which the output must be produced,

- Using the reports of the unsolved conditions as a checklist, the developer now considers the question of the availability of input data, and then analyse the nature of each input by using his own technique,

- By examining the TAG generated glossary, specify, when in time to introduce the document or file and the problem of what other elements key-fields and additional data fields are to brought in with the required input items,

- Specify the world picture of the system: after TAG has processed the required information on user inputs/outputs and files, data and job description reports are created that help analyst in providing a world picture of the users system,

- Working from the format definition supplied by TAG, the analyst must develop a database compatible with these figures, hardware

and configuration of the proposed system.

## 41.4 SYSTEM MODEL

Concepts used in specification and design

data type, frequency, period, priority, volume, survey period, data name, data size, input/output analysis form, chart for coding of class use, input/output results form.

Notation used textual and graphical

## 41.5 COMMENT

Completeness average

Economy average

Ease of use low

Additional comments

User is assured of defining only pertinent input elements and bringing them into the system at their proper place, all with a minimum efforts on his part. Superfluous and repetitious data can be identified and eliminated from the system, and descriptions are corrected. After all the inputs and outputs are defined to TAG, the next iteration of the program provides file formats and system flow descriptions, based on time (the time data enters the system and produced by it). The user gets an

overview of the system including all relationships of the system. The creative ability of the developer is also enhanced because of knowing all relevant data and relationships.

TAG is a general purpose technique applicable to the design of any data processing system in the commercial environment, and in the development of management information system, particularly where diversified activities, requiring several outputs are to be brought together and supported by an integrated database. The outputs of TAG are a set of ten reports.

## 41.6 REFERENCES

IBM: TAG, (1974b).

## 42.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | USE |
| Full name | User Software Engineering |
| Author(s) and institution(s) | Wasserman A I; University of California. |
| Date of first reference | 1979 |
| Application field(s) | data processing, science/engg., tools, AI. |
| Life cycle stages | requirement analysis<br>functional specification<br>structural design<br>detailed design<br>programming |
| Software support | data dictionary<br>analysis and checking<br>prototyping<br>detailed design aids |
| Development status | under development as a research project |

Comments

Supports entire system development process, except conceptual modelling phase. The aim is to provide tools and techniques that can lead to systematisation of an interactive information system

development process. It describes a software subsystem through a programming language called namely "PLAIN". 'User centredness' during the system development process is the central idea. It seeks to form an integrated system development environment addressed to specification and implementation of interactive information system development.

## 42.2 LIFE CYCLE MODEL

- Identify system objectives and constraints, including the conflicts of interest among user groups, based on the problem statement,

- Model the existing system using the requirement analysis method,

- Construct the conceptual model of the database, using semantic hierarchy model of Smith and Smith,

- Produce a system dictionary with all operation names, data items and data flows,

- Review the above steps, the analysis results within the development group, and insofor as possible, with users/customers,

- Build prototype of user dialogue tool through transaction diagram,

- Complete the architectural design,

- Complete the detailed design.

Notation used textual (Plain language)

## 42.4 SYSTEM MODEL

Concepts used in system specification and design

relational database, management facilities, lexical ordering function, strings, exception handling, input/output features, TDI, troll etc.

Notation used textual and graphical

## 42.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | average |
| Ease of use | above average |

Additional comments

USE provides to the developer with a method and tools that improve the quality of the system, and the processes used in system development. The tools specified are: transition-Diagram Interpreter for dialogue design, a programming language, a DBMS and an editor. It is highly pragmatic depending on traditional life cycle and software tools and can be defined as a mechanism combining all notions of software engineering and user involvement. The central focus is on the development of

interactive information system and specification of user dialogue for the development of such systems through prototyping. Formal specification techniques and normalised relations, as seems increasingly mandatory, which play important roles: so do the transition diagram, for user interface definition. Though one would like to need more discussion of their idea of extended use "inside" the system and how they relate to data flow diagrams.

It provides an integrated approach using data flow diagrams, and in studying man-machine interfaces. The transition diagram describes the interfaces, and may be encoded for the simulation needs, but little information is provided about the methods provided to the developer for obtaining data structures.

USE does not describe information analysis but discusses how a set of programs may be developed to support a particular case study. There is a plea for iterative approach like SADT, and ISAC. conceptual modelling has not been dealt explicitly. USE is low in data structuring aspects and high in technical aspects; system development phases are not clearly defined; cross referencing is difficult and lacks when dealing with parallelism.

The technical concepts supported by USE are: function and data decomposition, interface definitions, data flow, sequence control flow, concurrency and formal program verification. The work products are: specification, architectural design, detailed design, source code; and the quality assurance methods are: structured walkthroughs, design, transition diagrams and

consistency checks. It does not provide a technique for the validation of the finally developed system against the original requirements. It supports tools; and the equipment required is: unix V7 of 4.1 BSD and specialised automated support provided is consistency checking. The management aspects dealt are: version control, coding management, and system evolution.

## 42.6 REFERENCES


1. WASSERMAN A I, (1982)

2. FREEMAN P and Wasserman (1982)

3. WASSERMAN A I and others, (1983).

## 43.1 METHODOLOGY SUMMARY

| | |
|---|---|
| Short name | Young and Kent Algebra |
| Full name | Abstract formulation of data processing problems |
| author(s) and institution(s) | Young J W and Kent |
| Date of first reference | 1958 |
| Application field(s) | data processing |
| Life cycle stages | boundary specification<br>functional specification<br>detailed design |
| Software support | |
| Development status | published but probably never used |
| Comments | |

Provides little assistance in system development process, derivation relationships are established using algebraic notation, and these derivations are similar to Systematics (GRINDLEY, 1975) derivation chains and LANGEFORS (1973) precedence analysis, but Young and Kent method is difficult. Model is based on simple relationships and networks but notation is difficult. Ideas presented are useful, which are used in PSL/PSA and DADES methodology (OLIVE, 1982).

## 43.2 LIFE CYCLE MODEL

- Specification of information sets

- specification of abstract statement of the problem which consists (a) information sets, and (b) a list of documents (ie. input document, output document),

Graphical notation is also used to specify these documents.

## 43.4 SYSTEM MODEL

Concepts used in system specification and design

Information sets, documents, relationships (morphisms, defining relationships, producing relationships), items, conditions, operational requirements (volume, time).

Notation used graphic, tabular and textual

## 43.5 COMMENT

| | |
|---|---|
| Completeness | average |
| Economy | high |
| Ease of use | low |

Additional comments

Notation is mathematical and graphic, and provides sufficient accuracy but is difficult. The designer receives assistance in

the determination of the organisation of files, subroutines and redundancy checks. Since the "morphisms" are transitive, other relationships can also be derived. Though the notation is difficult, a tool is available to the designer for the description of his problem in pseudo-mathematical form, which also assists in ensuring that all inputs are utilised to produce outputs, and that a programmer gets a precise document. The number of files, record lengths, file densities, volumes and type of computations can be determined. This may be regarded as a good early effort for problem specification, and the concepts to be used in the development of a modern methodology.

## 43.6 REFERENCES


1. YOUNG J W and Kent (1958)

2. TEICHROEW D (1972, 1974b,)