Bo Sundgren

An Infological Approach to Data Bases
Bo Sundgren

An Infological Approach to Data Bases
Bo Sundgren

An Infological Approach to Data Bases (Urvår nr 7)

National Central Bureau of Statistics, Sweden, and
University of Stockholm,
Department of Administrative Information Processing
ISBN 91-38-01750-4

Printed in Sweden
K L Beckmans Tryckerier AB
Stockholm 1973
Table of contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>1.1</td>
<td>12</td>
</tr>
<tr>
<td>1.2</td>
<td>14</td>
</tr>
<tr>
<td>1.3</td>
<td>17</td>
</tr>
<tr>
<td>1.4</td>
<td>20</td>
</tr>
<tr>
<td>1.5</td>
<td>22</td>
</tr>
<tr>
<td>1.6</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>2.1</td>
<td>39</td>
</tr>
<tr>
<td>2.2</td>
<td>43</td>
</tr>
<tr>
<td>2.2.1</td>
<td>43</td>
</tr>
<tr>
<td>2.2.2</td>
<td>48</td>
</tr>
<tr>
<td>2.2.3</td>
<td>53</td>
</tr>
<tr>
<td>2.2.4</td>
<td>58</td>
</tr>
<tr>
<td>2.3</td>
<td>61</td>
</tr>
<tr>
<td>2.3.1</td>
<td>61</td>
</tr>
<tr>
<td>2.3.2</td>
<td>63</td>
</tr>
<tr>
<td>2.3.3</td>
<td>66</td>
</tr>
<tr>
<td>2.3.4</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>74</td>
</tr>
<tr>
<td>3.1</td>
<td>74</td>
</tr>
<tr>
<td>3.1.1</td>
<td>77</td>
</tr>
<tr>
<td>3.1.2</td>
<td>79</td>
</tr>
<tr>
<td>3.1.3</td>
<td>82</td>
</tr>
<tr>
<td>3.1.4</td>
<td>85</td>
</tr>
<tr>
<td>3.2</td>
<td>92</td>
</tr>
<tr>
<td>3.2.1</td>
<td>92</td>
</tr>
<tr>
<td>3.2.2</td>
<td>94</td>
</tr>
<tr>
<td>3.2.3</td>
<td>99</td>
</tr>
<tr>
<td>3.2.4</td>
<td>101</td>
</tr>
<tr>
<td>3.2.5</td>
<td>104</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>The infological data base</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2</td>
<td>The black box reservoir of information</td>
</tr>
<tr>
<td>4.3</td>
<td>Environment subsystems</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Observed and controlled object system</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Information supply and consumption</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Data base goal-stating, decision-making, and administration</td>
</tr>
<tr>
<td>4.3.4</td>
<td>The computer</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Other data bases</td>
</tr>
<tr>
<td>4.4</td>
<td>Basic subsystems of the infological data base</td>
</tr>
<tr>
<td>4.4.1</td>
<td>The schema</td>
</tr>
<tr>
<td>4.4.2</td>
<td>The nucleus</td>
</tr>
<tr>
<td>4.4.3</td>
<td>The filter</td>
</tr>
<tr>
<td>4.5</td>
<td>The dynamic data base</td>
</tr>
<tr>
<td>5</td>
<td>Data base interaction</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>The interaction process</td>
</tr>
<tr>
<td>5.2.1</td>
<td>A classification of interactions based upon the subsystem structure of the data base environment</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Open and closed interactions</td>
</tr>
<tr>
<td>5.2.3</td>
<td>A classification of interactions based upon the S-O-R model</td>
</tr>
<tr>
<td>5.3</td>
<td>Data base transactions</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Modification transactions</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Query transactions</td>
</tr>
<tr>
<td>5.4</td>
<td>Communication and illustration techniques</td>
</tr>
<tr>
<td>5.4.1</td>
<td>An example</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Data base interaction languages</td>
</tr>
<tr>
<td>6</td>
<td>Box structures of aggregate messages</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>6.2</td>
<td>A general theory of boxes</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Basic concepts</td>
</tr>
<tr>
<td>6.2.2</td>
<td>A convenient notation</td>
</tr>
</tbody>
</table>
6.3 Box theory and the infological framework
6.3.1 αβ-queries and αβγ-queries
6.3.2 Object boxes
6.3.3 Fact and message boxes
6.4 Applications of infological box theory
6.4.1 Infological descriptions of data base processes
6.4.2 Quality and protection problems
6.4.3 Descriptive data base interaction languages

7 Data base systemeering
7.1 Systems and systemeering
7.1.1 Introduction
7.1.2 Different approaches to systems
7.1.3 A fundamental principle of systems work
7.1.4 The system and its environment
7.1.5 Systemeering techniques
7.2 Systemeering and decision-making
7.3 How to systemeer a data base
7.3.1 The purpose of a data base
7.3.2 A high-level goal structure
7.3.2.1 Usefulness
7.3.2.2 Non-harmfulness
7.3.2.3 Economy
7.3.2.4 Viability
7.3.3 Reconciliation of goals
7.3.4 Data base subsystems
7.3.5 Matching the goals and the functions

8 File design
8.1 Introduction
8.2 Basic concepts
8.2.1 Accessible memories
8.2.2 Files, entries, and terms
8.2.3 Different kinds of files
8.2.4 Clusters and file allocation
8.3 File structuring operators
  8.3.1 File establishment
  8.3.2 Intra-entry structuring
    8.3.2.1 Entry compression
    8.3.2.2 Reordering of terms
    8.3.2.3 Entry chopping
    8.3.2.4 Ordering of segments
  8.3.3 Inter-entry structuring
    8.3.3.1 Ordering of entries
    8.3.3.2 Intra-file consolidation
  8.3.4 Inter-file structuring
    8.3.4.1 Consolidation
    8.3.4.2 File overlapping
  8.3.5 File allocation
  8.4 File structuring techniques
    8.4.1 Demarcation of αβ-complexes
    8.4.2 An exemplifying analysis of a directory/file complex

9 Process design
  9.1 Basic concepts and philosophy
    9.1.1 Data base processes and resources
    9.1.2 Design of programs, processes, and routines in "conventional systems" and in data base systems
    9.1.3 Resource-oriented and transaction-oriented process design
    9.1.4 Transaction-oriented process design

9.2 The processing of an αβ-query
  9.2.1 A basic retrieval situation
  9.2.2 Other than α-attributes in the compound attribute key
  9.2.3 β-attributes in the compound attribute key
  9.2.4 Scattered α-attributes
  9.2.5 Overlapping relevant files
<table>
<thead>
<tr>
<th></th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Summary and bibliography</td>
<td>442</td>
</tr>
<tr>
<td>10.1</td>
<td>The infological approach in retrospect</td>
<td>442</td>
</tr>
<tr>
<td>10.2</td>
<td>Bibliographical remarks</td>
<td>452</td>
</tr>
<tr>
<td>10.3</td>
<td>List of references</td>
<td>457</td>
</tr>
</tbody>
</table>
"An infological approach to data bases" reports part of the data base research and development work which has been carried out over a number of years at the National Central Bureau of Statistics, Sweden. Professor Börje Langefors, University of Stockholm, Department of Administrative Information Processing, has been the scientific supervisor of the reported project. Very briefly the objective of the project has been to develop an integrated theoretical framework for design of large-scale data bases. The framework should

(a) enable people who are not data processing professionals to co-operate actively and constructively in data base design projects

(b) make it possible to transform systematically the problems, desires, and requirements of those who are affected by a projected data base into problems which can be tackled by data processing specialists

(c) enable data processing specialists to analyze the computer-oriented data base problems systematically and with sufficient precision

(d) make it possible to design data bases with which decision-makers, planners, and researches within different specialized fields could interact constructively, even if the information needs of the interactors are complex, and even if they lack knowledge about computers and computing

There are definitely different opinions among authorities in the computing world as to whether it is feasible to cover all the aspects (a)–(d) within one and the same framework. This report supplies evidence in support of the hypothesis that an integrated approach is both feasible
and necessary for the success of large-scale data base undertakings.

"An infological approach to data bases" is certainly not the product of one single hero's ivory tower thinking. Instead the number of people who have in some way or another contributed to the reported results is so great that it is possible to mention only a few of them here.

First of all I want to thank Professor Börje Langelors who has played a double rôle in the research project. Many of the basic ideas in the report can be traced back to his "Theoretical Analysis of Information Systems", to numerous working papers from his hand, and to seminars and discussions, which he has led. Moreover, his recurrent and enthusiastic encouragement while supervising the research project has been of paramount importance to the progress of the work.

The National Central Bureau of Statistics, Sweden, has turned out to be an excellent environment for advanced data base research and experimentation. To my knowledge, the Central Bureau of Statistics, headed by Dr. Ingvar Ohlsson, was one of the first organizations in the world to give the data base concept an explicit and central rôle in its goals and policies. This in turn is very much due to the influence of Professor Svein Nordbotten, who in fact anticipated the data base concept in the early sixties, several years before the very term "data base" became popular. Nordbotten, who was then at the Norwegian Central Bureau of Statistics, launched the idea that the returns obtainable from the huge data capital investments of a central bureau of statistics could be substantially increased by organizing the data logically as one integrated statistical file system instead of running different branches of statistics production more or less independently of each other.
Among the leading persons within the Central Bureau of Statistics I want to direct special thanks to Dr. Lennart Fastbom, Head of the Planning Department, and to Dr. Christer Arvas, Head of the Data Processing Methods Unit. They have taken a very active interest in the project and contributed with a lot of valuable ideas and constructive criticism.

Many stimulating discussions concerning the basic conceptual framework presented in chapters 2 and 3 of this report have taken place within the project group for development of a system for socio-demographic statistics (SSDS). The SSDS project, which is part of a joint international effort, has offered and still offers great opportunities to concretize and test the conceptual foundation of infological database theory. In this connection I want to emphasize the contributions of Fil. kand. Svante Öberg who has devoted a lot of genius, time, and energy to chiseling out and testing some of the basic infological concepts.

Other colleagues who have significantly contributed to the research results are Björn Nilsson, Lars Olsson, Michael Bucknell, and Harald W Thorburn. A special acknowledgment goes to Senior Lecturer Jeremy W Firth, Canberra College of Advanced Education, who spent a sabbatical year at the Central Bureau of Statistics as a consultant to the database research project. My own confidence in the infological approach to data bases has been significantly strengthened by Jerry Firth's combined enthusiasm and frank, constructive criticism.

Finally I want to thank Miss Birgitta Tillquist and Miss Marja Talonpoika who have patiently and efficiently typed the manuscript and drawn the figures.

Stockholm in July 1973

Bo Sundgren
1. Introduction

1.1 The infological approach

Over the decades to come, data processing systems are going to be more and more integrated with the planning and decision-making processes of human beings, organizations, and societies. If we are to successfully accomplish the task of designing such systems we have to consider explicitly how planning and decision-making people form, save, and process knowledge about the reality with which they interact. We should design systems and system design techniques to facilitate communication. This does not mean, however, that we should persuade people to use vague natural language, when a stricter mode of expression would be more appropriate for systematic thinking.

The material presented in this report is meant as an embryo of an infological theory of data bases matching the above-mentioned requirements. It is a theory which basically treats the data base as an information storing and processing black box extension to the human mind. A number of basic concepts are introduced in order to describe the inherent structure of information used for instance in planning and decision-making processes.

The infological approach to data bases puts more emphasis upon the conceptual and semantic aspects of data structures and data processing than is common practice or even common theory today. It could be suspected that system design based upon an infologically oriented theory would be less efficient in terms of bytes and micro-seconds than system design based upon a strictly datalogical theory, i.e., a theory which starts from a set of storage and file structures. I have come to the conclusion that this may not be true. In order to supply some evidence in support of this hypothesis I have devoted two chapters towards the end of the report to the discussion of two typical datalogical problem areas: the design of files and processes. Among other things I try to show there that it should be possible to integrate existing theory on file organization and process design with a broader infological theory of data bases. I therefore propose definitions which
explicitly relate the basic datalogical concepts to the non-representational infological frame-work. The analysis seems to indicate that even those who are not particularly interested in making things easier for the ultimate users of data processing systems, could very well adopt the concepts introduced in this paper without giving up anything but possibly the air of mystery which has too long isolated us, the computer professionals, from the rest of the world.
1.2 Infological aspects of the human mind

Figure 1 shows an infological model of the human mind. It summarizes a number of assumptions which are basic to the following analysis and concept formation. We are going to discuss these assumptions in this section.¹

First, as can be seen from figure 1, we postulate the existence of an objective reality outside the mind of every beholder. As we have thereby taken a particular position vis-à-vis one of the eternal philosophical issues, some argument on the subject should be provided. The assumption of an objective reality is equivalent to the assumption that it is always possible to determine objectively, at least in principle, whether a particular statement about reality is true or false. It seems reasonable to maintain that statements, which users cannot agree upon as being verifiable, have no place in data bases of the kind to be discussed in this paper. These are data bases which are to be shared by groups of planners and other decision-influencers.

Next we turn to the mental processes, represented in the figure by boxes marked ①, ②, ③, and ④.

① We postulate that before a human being can gain any knowledge at all about the reality, he must form certain basic concepts. We also assume that this concept formation is achieved by repeated perception². This process is continually going on during a person's life time. New concepts are formed all the time, and old ones may be modified, expelled, or forgotten.

② Equipped with a set of concepts, the human being is capable of transforming its sense-organ perceptions into specific knowledge about the reality, i.e., knowledge about particular situations and events.

¹ There are several psychological models of the intellectual processes. The "infological view" accounted for here seems to be at least compatible with the view taken by Madsen in [6], chapter 9. Compare, for example, figure 1 below with Madsen's figure 9.4.

² The assumption that some basic concepts are inherited is also compatible with the rest of the model, however.
Figure 1  An infological model of the human mind
From its store of specific knowledge the human being may be able to induce the empirical law type of general knowledge. The body of specific knowledge may also suggest the formation of new concepts, possibly defined in terms of existing ones. Note the feed-back effect of the induced concepts upon the perception process.

In the deduction process the human being uses among other things the laws of logic, which are supposed to be inherited and identical for all human beings at every point of time. By combining the laws of logic with empirical laws and specific knowledge, a human being may deduce new specific knowledge and new empirical laws.
1.3 The frame of reference

Suppose P is a human being, a person. Then by P's frame of reference $R_P(t)$ at time $t$ we mean the collection of concepts, definitions, laws of logic, empirical laws and perceived, deduced, or deducible knowledge belonging to the mind of P at $t$.

A person is more or less conscious of different parts of his frame of reference. We postulate the existence of a consciousness function $c(k, P, t)$, which may take values between 0 and 1, and which tells the degree of consciousness of different pieces of knowledge $k$ for different persons $P$ at different points of time $t$. We define that

$$c(k, P, t) = 0 \text{ iff } k \notin R_P(t)$$

whereas as $c(k, P, t) = 1$ represents a situation where $P$ is immediately aware of $k$ at $t$.

The on-going perception process of a person (cf figure 1) does not only affect his awareness of new knowledge, information, but may also increase or decrease his degree of consciousness with respect to knowledge which is already part of his frame of reference. New perceptions may even change or destroy old knowledge.

We may think of a person's frame of reference as a gravity field around the person's centre of consciousness, $C_P$. The shorter distance is between a piece of knowledge, $k$, and $C_P$, the higher the degree of consciousness $c(k, P, t)$ will be. Figure 2 visualizes the effects of a perception process. New knowledge, information, is added to the frame of reference (the continuous arrow), and old pieces of knowledge are moved towards or away from the centre of consciousness (dotted arrows).

The contents of a person's frame of reference may also be classified as explicit or implicit knowledge. A deducible

$^1$ if and only if
Figure 2  The frame of reference as a field of gravity
piece of knowledge is implicit until it has actually been deduced in a deduction process (figure 1) or directly perceived, at which time it becomes explicit. Note that "explicit-implicit" is another dimension than "more or less conscious". Explicit knowledge may be far from the centre of consciousness, and implicit knowledge may be close to it. For instance the total price for 31,397 articles à $9.97 may be implicit and yet very close to $p$ for a person who knows his arithmetic. Specific knowledge deducible from other specific knowledge in combination with an advanced theorem of a complicated theory will certainly be implicit knowledge of extremely low degree of consciousness to a person who has just incorporated the axioms of the theory with his frame of reference. Memory-pictures from the childhood seem to be good examples of explicit knowledge which may be very far from a person's $p$ during his active life, but which may become more conscious in the old age.
1.4 Data

In the previous section we stated that the perception process has side-effects like

(a) moving knowledge from the centre of consciousness to more peripheral parts of the frame of reference

(b) distorting parts of the frame of reference

As is well-known, these effects may be detrimental to a person's decision-making ability. It will cost time and energy to make conscious, or access, distant parts of the frame of reference, and distorted knowledge may lead to false deductions and ultimately to bad decisions. It is probably, among other things, negative experience of this kind that has led man to create artificial extensions to his own mind. With these extensions human beings have been enabled to store knowledge outside their mental frames of references by the use of data, and to replace slow, error-prone mental accesses and deduction processes with data retrieval and processing operations performed by people or machines.

Human beings have also learnt to enrich their mental frames of reference by man-to-man communication. This, too, was enabled by the use of data.

What then is "data"? Let us try the following definition.

Definition. If a person intentionally arranges one piece of reality to represent another, we shall call the former arrangement data, and we shall say that the arranged piece of reality is a medium, which is used for storing the data.

This definition covers all kinds of data: digital data, analogous representations, spoken and written language, etc. In a particular context it may be convenient to limit the scope of analysis to, say, computer-adapted digital data. We shall do so in the latter part of this paper.
Note the wording "... intentionally arranges ... to represent". It is not sufficient that two phenomena are related to each other, incidentally or by design. There should have been a human intention of representing, and not only correlating, one thing with another.

In most cases data represent primarily a person's knowledge about reality and only secondarily the piece of reality itself. The represented piece of reality has then been observed by a human being, and the data registration has been made on the basis of the observer's perceived knowledge. Direct registration is not quite unusual, however. Consider for instance automatic cameras, chart-writing hygrometers, max/min-thermometers, etc. These examples show that the property of being data need not be dependent upon the co-operation of a human mind in the very registration process. The important thing is that somebody has intentionally arranged for the registration.
1.5 Information

We have already used the term "information" a couple of times as synonymous with "new knowledge", and as a matter of fact it is not easy to find a better definition. With the conceptual frame-work introduced in the preceding section "new knowledge" implies many important properties of information. For example:

- Being knowledge, information exists only in the mind of a human being as a part of a mental frame of reference.

- Information is always related to one particular existing or imagined person, the reference person; i.e., information is always somebody's information. When we talk of storing information in a data base, we often implicitly assume all the users of a data base to be identical with some "average" reference person. The fact that this assumption is seldom even approximately true has consequences which may easily be overlooked at data base design time.

- Information is always related to a set of old knowledge, the reference knowledge. More precisely the reference knowledge is the frame of reference of the reference person at a particular point of time. For instance, when we talk of the information of a message received by a person we usually tacitly understand the reference person to be the (typical) receiver of the message and the reference knowledge to be the frame of reference of the same person at the point of time when he receives the message.

- Assuming that new knowledge cannot arise spontaneously within the human mind, the definition suggests the existence of an external source for every piece of information.

- If i is a piece of information, and k is the reference knowledge of i, then i is not a logical consequence of k, because if it were, then i would be at least implicit knowledge of the reference person, i.e., i would not be new knowledge.
In the previous sections we have already identified two major sources of information about a piece of reality p, viz.

(a) p itself
(b) data representing p

In the latter case we may talk of the representing data as a data message, or a record. Data messages are transformed into information by a mental process, which may be a fairly complicated combination of perception, deduction, and inference processes. The transformation process is schematically lined out in figure 3, which is basically identical with figure 1.

It is postulated in figure 3 that we may consider any data-information transformation process to be executed in two steps, the first of which is completed, before the second is initiated.

The two steps are

(a) interpretation, and
(b) derivation

The interpretation process yields as output a conceptual message. The structure of conceptual messages, or just messages as we shall call them, will be studied in chapter 3. The conceptual message is the meaning or semantic contents of the data message as interpreted by the receiver (perceiver) of the data message.

In order to be able to interpret a data message, a person must possess a compatible frame of reference $R_p(t)$ at the time when he (finally¹) interprets the data. $R_p(t)$ must contain interpretation rules, which may be thought of as a special

¹ Usually interpretation time = perception time, but we may imagine situations, where a person first perceives data without knowing how to interpret them, or even without knowing that they are data, and where he only later gets a compatible frame of reference. Example: a person first learning mechanically a latin poem and then learning latin.
Figure 3  The transformation of data into information
kind of empirical law type of general knowledge\(^1\) (cf figure 1), and which make the person associate the data with a piece of reality, supposed by \(P\) to be represented by the data.

If a person possesses a frame of reference which is compatible with a data message, we shall say that the data are meaningful to the person. Data which are not meaningful do not convey information. On the other hand meaningful data may but need not convey information. A data message telling a person what he already knows is an example of meaningful, informationless data. However, note that the fact that a certain person tells another person something may in itself convey information to the latter. But then it is not the data message which conveys information; the new knowledge is based upon direct perception of reality and is not obtained indirectly via data (cf the definition of "data").

**Example.** Suppose that \(B\) knows that \(C\) is dead, and that \(A\) then tells him so. The spoken data message "\(C\) is dead" is then informationless to \(B\), but \(B\)'s perception of \(A\) telling him that \(C\) is dead, could give him some pieces of new knowledge depending upon the contents of his frame of reference. For instance, the perception of the event could inform \(B\) that

- \(A\) knows that \(C\) is dead
- \(A\) does not know that \(B\) knows that \(C\) is dead
- \(A\) can speak

The last conceptual message would not be information under normal circumstances but could be so, if \(A\) had earlier been dumb.

End of example.

\(^1\) Usually interpretation rules are not inferred but rather themselves interpreted from data messages, as may other empirical laws be perceived, too, by the way, or just "established" (decided upon) by \(P\) himself, if he was also the original sender of the data message, i.e. the person who arranged for the data representation (cf the definition of "data" in section 1.4). However, for instance when ciphered data messages are unduly deciphered, the interpretation rules are often inferred, e.g. by means of statistical techniques.
As was said above, a compatible frame of reference will make data meaningful to the receiver. Note that this does not at all imply that the receiver will necessarily interpret the data as intended by the sender when he arranged for a piece of reality to be represented by the data, or that two receivers of the data message, each with a compatible frame of reference, will interpret the data into identical conceptual messages. Still less does it imply that different receivers with different, though compatible, frames of reference will get the same information.

The same conceptual message may yield different information to different people due to differences in the second step of the data-information transformation, the derivation process (cf figure 3). These differences in turn are due to differences in the frames of reference. Naturally the derivations, i.e. the deductions and the inferences, may yield different results if they are based upon different specific and general knowledge. Even if the derivation results are identical as such, they may have different information implications as we have defined information as the difference set

\[ \text{(updated knowledge)\setminus\text{(old knowledge)}} \]

(cf figure 3).

Thus the data-information transformation process is dependent upon the frame of reference in two ways. Firstly, the concepts and interpretation rules determine the meaning of the data, i.e. the conceptual message corresponding to the data message. Secondly, the existing body of specific and general knowledge determines what new knowledge, information, the absorption of the conceptual message implies.

We have mentioned two sources of (new) knowledge, viz.

(a) direct observation of a piece of reality, and
(b) interpretation of data representing a piece of reality

1 The interpretation process is assumed to include the direct observation of the data.
If we assume that these are the only sources of knowledge, and that the mental process associated with (a) like that of (b) proceeds in two steps, where the output of the first one is a conceptual message, we may summarize all knowledge formation processes by figure 4. We postulate there that all specific and general knowledge derivable from the (primary) conceptual message can be regarded as a set of (secondary) conceptual messages. As we have also assumed that all knowledge must emanate from one of the two sources indicated in figure 4, it follows that the whole body of knowledge contained in any frame of reference \( R_p(t) \) can be regarded as a set of conceptual messages.

Thus the **semantic contents** (meaning) of a data message, \( dm \), is a primary conceptual message, \( pm \), whereas the **information contents** of \( dm \) is a set of secondary conceptual messages, \( \{ sm \} \), where every \( sm \) is derivable from \( dm \) and \( R_p(t) \) and where no \( sm \) is contained in \( R_p(t) \). We could also express \( \{ sm \} \) as the difference set

\[
R_p(t + \Delta t) \setminus R_p(t)
\]

assuming that \( dm \) and only \( dm \) is perceived by \( P \) during the time interval \( \langle t, t + \Delta t \rangle \).

If we relate the discussion of information in this section to the discussion of **consciousness** and **explicitness** in section 1.3, we may draw several important conclusions. For example:

- Being knowledge, information may be more or less conscious. Every secondary conceptual message in the information contents of a data message need not necessarily be placed close to the centre of consciousness of the perceiving person.

- Being knowledge, information may be explicit or implicit, depending upon whether potential deductions are actually performed in connection with the perception or not. Implicit information may become explicit knowledge at a later point of time.
Figure 4: Knowledge formation in two steps
If the perception of a data message (or another piece of reality) has the side-effect of making an implicit message explicit or moving an old message closer to the centre of consciousness, the message concerned is not contained in the information contents of the data message. From this follows that a data message may be of value as a "reminder", even if it should not convey any information to a particular person with a particular frame of reference.
1.6 The need for infological models

A model is defined here as an abstraction, a simplifying representation, of a slice of reality. A model is often created in order to facilitate analysis, planning, and decision-making within a particular subject matter area. Formally a model ought to be the result of a homomorphic mapping of one subset of reality upon another.

According to the definition given in section 1.4, a set of data is a representation of a piece of reality. This implies that sets of data are models. As a matter of fact, human beings have used data models as extensions to their minds for thousands of years.

The use of models has several advantages. For example, it is usually much easier to manipulate the entities of a model than to carry out the corresponding actions upon the represented reality. Moreover, model manipulations need not affect the represented reality. This is important in connection with planning and decision-making, where one often wants to compare the effects of different actions before they are actually carried out. Not even the model itself need be destroyed by the use of it. The data of a data model may be retrieved and processed several times, and each time it may be considerably faster to retrieve the data than to make the corresponding reality observation.

The advance of data processing machines has significantly increased the relative advantage of using digital data models in planning and decision-making. In the early days of computers the interest was focused upon the possibilities of transforming data according to complicated algorithms millions of times faster than before. However, the structure and scope of the models used were by and large inherited from earlier periods.

With the development of new input/output and storage equipment it became feasible to use computers for clerical routines, where bulks of data are processed according to fairly simple
algorithms. Still, however, it was very much the question of translating models, where the manipulations were made by human beings or by mechanical equipment, into models where the same manipulations were made by a computer. Model structure or model scope was seldom very much altered.

Thus a computer application, which may today be regarded as conventional, typically concerns a fairly small slice of reality. The data representations are often straight-forward translations of form or card equivalents, and so are the structurings of data into files.

During the last decade there have been vivid discussions about the integration possibilities created by the technological progress. It seems, however, that the successful implementations of "integrated file systems", "management information systems", "data base systems", and similar concepts have been embarrassingly few. Why have we not been able to fully utilize the potential inherent in modern computer technology? Let us use figure 5 in an attempt to analyze this problem.

Figure 5 shows among other things how we may sub-divide a reality → data base 1 mapping into four sub-mappings:

1. the reality + subject matter model sub-mapping, established by the abstraction process

2. the subject matter model + infological model sub-mapping, established by the specification process

3. the infological model + datalogical model sub-mapping, established by the design process

4. the datalogical model + data base sub-mapping, established by the implementation process and maintained by the operation process.

1 At this stage we may define a data base as a permanently maintained digital data model of a slice of reality.
Figure 5  The homomorphic mapping of a slice of reality into a data base model
As long as the data models covered very small slices of reality and involved only small volumes of data it was not necessary to sub-divide the "reality + data" mapping. Many a time it was even possible for one and the same person to formulate a subject matter problem, write a computer program, run the computer, and re-interpret the result into "real" terms.

With growing volumes of data it became necessary to introduce intermediary models reflecting at the same time the structure of the slice of reality, as conceived by the subject matter specialist, and the storage structure, as conceived by the programmer. These models, which may be called file models or datalogical models, focus interest upon classes or types of entities; the storage mapping and management of individual entities is taken care of by the computer's operating system.

There have been important efforts during the last couple of years to develop general languages, in which very complex datalogical models may be defined, and to develop software, which makes the datalogical models largely invariant to physical changes. Such software makes it possible to change the "datalogical model + data base" mapping without changing the datalogical model. This is a desirable property if the contents of the data base are changing, and storage, retrieval, and maintenance economy is important. ¹

There is an intrinsic source of contradiction and conflict in all "traditional" datalogical models. As was stated above, they are supposed to reflect at the same time the structure of a slice of reality, as seen by a subject matter specialist, and the storage structure, as seen by the programmer. Experience shows that it is possible, but not always easy, to reconcile without formal aids the different views as long as

(a) the subject matter models are of limited scope and not too complex, and

¹ The invariance of datalogical models and the consequent invariance of application programs may also be important if the data base has to be reorganized due to other reasons than dynamic contents, e.g. changing user preferences or shifts in cost relationships.
(b) file organization problems are not critical, i.e.
there are straight-forward solutions to them, and
imperfections in file consolidations, file + storage
mappings, etc, are not disastrous.

These conditions have been fulfilled in most conventional
computer applications, which, as stated earlier, are often
conversions of old manual systems. Because of the technological
break-throughs and the shifted cost relationships, it has not
been too difficult to make the new system do what the old
system did - and a little more - at the same or lower costs.

When we attack the task of designing a data base giving more
complete information support to planning and decision-making
activities in broader and more complex subject matter areas,
it is not so easy to overcome communication problems and to
avoid "design disasters".

One reason for this is that more ambitious applications will
use up the extra resources created by the technological break-
throughs. It becomes more forcing than with conventional
applications not to waste resources, and the theoretical
limitations become practical limitations as well. This in
turn necessitates more complex datalogical models, file
structures, and more complex and dynamic file + storage
mappings.

It is unrealistic to believe that subject matter oriented
people would be capable and willing to formulate system
specifications in terms of the more complex datalogical models.
Even if they were, this approach would force them to solve
at the same time, explicitly or implicitly, the major design
problems, such as what information to group together into one
file, what information to store redundantly in several files,
what general organization and access schemes to choose for
different files, and so on.

On the other hand, if we make a computer-oriented systemeer
responsible for the design of the datalogical model, we force
him
a) to learn all subject matter models used in the different decision areas to be supported by the database

b) to communicate with the future users of the database in terms of their respective subject matter models in order to set the goals for the database

c) to use this goal structure formulated in the terms of overlapping, incompatible subject matter models and vague natural language, in his more or less intuitive search for an optimal file structure

d) to design the database system to communicate with each user in that user's subject matter model.

A reasonable compromise between the extreme approaches seems to be to follow figure 5 and sub-divide the "subject matter model ↔ datalogical model" mapping by the introduction of an infological model.

An infological model for a particular application should be sufficiently detailed and formal as to give a computer oriented systemeer precise guide-lines for his design work - without anticipating any design decision.

On the other hand it should ideally be as simple for a database user to state design goals, formulate information requests, and interpret answers to the requests in terms of the infological model, as it would be for him to make any precise statements about and within "his own" subject matter oriented model.

This report proposes a general conceptual framework, or the embryo of a theory, for infological models.

Note the distinction between the general infological framework and infological theory on the one hand and special infological models on the other. For each database to be built according
to the approach proposed in this report, one particular infological model has to be specified (cf figure 5). Whenever we make such a specification we use concepts and terms, which are part of the general infological framework, and we accept certain rules, the axioms and theorems of the infological theory. Thus there is in a sense a one-to-many relationship between the general infological framework and the special infological models.

The relationship between subject matter models and infological models is more complicated; it could be described as "many-to-many". On the one hand, as has been stated, users from different subject matter areas could very well want to share the same data base. Then ideally the interested parties should agree upon an infological model such that there is a bi-unique mapping between each of the subject matter models involved and the common infological model. On the other hand, there is no unique way to specify an infological model, given a particular subject matter model. The restrictions imposed by infological theory supplemented with practical recommendations cannot eliminate all "degrees of freedom" from the specification process, which like all modeling work, is basically intuitive and subject to judgment.

If we go on comparing the different kinds of models indicated in figure 5, we easily realize that a particular infological model does not uniquely point out one particular datalogical model. In fact, if it did, the concept of infological models would be superfluous. It is the task of the design process to select one datalogical model for implementation, and this task is certainly complicated enough. However, the design process can be much more systematically carried out, if it starts from a specified infological model, than if this intermediary problem formulation is omitted.

The former part of this report presents a basic set of infological concepts and the general idea of infological models. The latter part of the report suggests how, in a natural way, the general infological framework and theory
may be extended into a general datalogical framework and theory. The hypothesis that this approach is suitable for theoretical and practical purposes is supported by a general analysis of the data base design process and sample analyses of a few major design issues.
2.1 Introduction

All systematic description, explanation, prediction, design, construction, implementation, and control of real-world phenomena seems to imply the use of models of the slice of reality which is described, explained, etc. Typically different kinds of specialists, and even different individuals working within the same speciality, use different models even for the same slice of reality. When we are about to design a data base which is supposed to be able to serve the information needs of different kinds of specialists and different interest groups, we should like to have a model which could be understood and accepted, at least as a communications interface, by all parties who are interested in the data base, be they information consumers, information suppliers, data base designers, programmers, or something else. The infological framework, to be presented in this and the following chapter, offers a set of general concepts, in terms of which common models of the kind mentioned may be specified. One unique model, a so-called particular infological model may have to be specified for each particular data base to be designed, but it should be possible to use the same basic infological concepts for different models and different data bases.

Data base unique infological models which are understood and accepted by all interested parties seem to be necessary for appropriate design and operation of data bases to be possible. For example, it is difficult to see, how it would otherwise be possible to give different categories of people equal chances to influence upon data base design decisions, or how it would be possible to prevent flagrant misinterpretations of the information contents of a data base.

If it turns out to be possible to use the same conceptual framework in the formulation of different infological models for different data bases, additional advantages will be gained. Firstly, the designers of each particular data base will be saved the trouble of having to invent and define their own consistent set of basic concepts. Secondly, all data base designers will be able to capitalize immediately upon the results of general studies and research work carried out in
terms of widely recognized concepts. Thirdly, even if different infological models are specified for different data bases, the general conditions for future integration of data bases, which are today designed and operated as separate systems, will be better if the different infological models are at least formulated within the same general conceptual frame-work.

As was mentioned in chapter 1, individual software manufacturers and organizations like the CODASYL have undertaken important efforts during the last couple of years in order to develop concepts and models for data base management. Unfortunately these efforts seem to have been too much hampered by "traditional" views and attitudes towards data management and file organization. This has led to lack of generality and simplicity, and above all it has conserved a computer-orientation in the concepts and models, which makes them more or less unintelligible to people who have not been trained in the data processing field. The approach to data bases advocated by the CODASYL and others may be called "the generalized datalogical approach". It aims at making life a little easier for so-called application programmers and data base administrators. This is no doubt a worthy cause, but it does not at all facilitate the involvement of the ultimate data base users and other computer novices in the data base design process, and such involvement is essential if future data bases are to be what people in general want them to be.

According to "the generalized datalogical approach" it seems to be tacitly assumed that all that can be said about data bases could and should be said in terms of representation oriented concepts like "file", "record", "term", "tree structure", "network", and "inverted list". The infological approach, on the other hand, emphasizes the distinction between

(a) real-world phenomena

(b) information about real-world phenomena

(c) data representations of information about real-world phenomena
According to the infological approach this distinction is basic to the understanding of the data base concept. As suggested by figure 5 in chapter 1 the distinction also facilitates data base design by making it natural to sub-divide the complex "reality + data base" mapping into a number of intelligible and less complex sub-mappings.

The general infological frame-work to be defined and analyzed in chapters 2 and 3 provides concepts by means of which particular infological models may be specified that cover the aspects (a) and (b) of the data base to be designed. Aspect (a) may be called "the object system aspect" and aspect (b) "the information aspect". In later chapters of the report we shall try to show that the general infological frame-work may also serve as a sound bases for the development of tools by means of which aspect (c), "the datalogical aspect", of the data base concept may be tackled, too.

In this chapter we start the presentation of the infological frame-work by defining and discussing a number of concepts in terms of which the designers of a data base should be able to specify formally the slice of reality, or the object system, about which they want to store information in the data base. In chapter 3 then, we shall see how the structure of this information may be formally described independently of any existing or planned data base internal representation of the information.

In the presentation of the infological frame-work we make a distinction between fundamental concepts, which are formally undefined, and derived concepts, which are formally defined in terms of fundamental concepts. Within the object system part of the infological frame-work there are four basic fundamental concepts: "object", "property", "object relation", and "time". The basic, formally undefined concept within the information sphere of the infological frame-work is "reference", which will be discussed in section 3.1.

Remark. The status of "reference" as a basic information concept does not at all imply that a reference is a basic,
or minimal, piece of information. On the contrary, it will be argued in section 3.2.1 that particular reference expressions, which we shall call "elementary messages", are to be regarded as the minimal information structures. This position is equivalent to the position taken by Langefors in \[48\] and \[49\].
2.2 Fundamental concepts

2.2.1 Objects

Intuitively, an object is something that we are interested in, something that we want to gather information about.

Objects may or may not be physical entities. Enterprises, departments, educations, professions, leisure activities, and car accidents are as good "object candidates" as are persons, buildings, areas, pets, and motor vehicles. It is when we specify a particular infological model (cf figure 5 of chapter 1) that we decide what phenomena to include as objects. What decisions we then arrive at will be dependent upon our objectives: what are the goals of the information system, of which the infological model is a part? What kind of object system do we want to control?

The specification of certain entities as objects is thus ultimately a question of judgement. What theory could do here is to provide rules of thumb or even more formal guidelines to these decisions. Infological theory could also help by systematically analyzing the consequences of looking at the same piece of reality through different infological glasses.

Being a fundamental concept of infological theory, "object" if formally undefined. Like fundamental concepts of other theories it gets partial meaning through other concepts, which are defined in terms of the fundamental concepts, and through axioms or postulates, which lay restrictions upon the concepts involved.

Whereas the object concept as such is fundamental, and thus formally undefined, each particular object, specified in a particular infological model, may either be formally undefined, atomic, or formally defined in terms of other specified entities. The latter kind of object instances are called compound objects.

Of course, the same real world entity may be specified as an atomic object in one infological model and as a compound object
in another. The latter strategy should be chosen, if one wants to make formally explicit an inherent structure of the phenomenon of interest. If, on the other hand, the inherent structure of the object is thought to be irrelevant to the problems, for which the data base is to be designed, then the object should be declared as atomic.

A compound object may consist of a set or a tuple of other objects, or of a constellation (see section 2.3.3). The object constituents of a compound object may in turn be atomic or compound. Any finite number of recursions is allowed.

Example. In a national information system, i.e. an information system which informs about social, economical, demographical and other conditions in a country, households and industrial concerns could be examples of compound objects consisting of individual human beings and enterprises respectively.

Example. In the information system of an enterprise, production teams and divisions may be regarded as compound objects, consisting of workers and departments respectively.

Example. Constellations\(^1\) of transaction or event type will often be strong candidates for compound object specification. A sales transaction, for instance, may be primarily thought of as a relational e-constellation

\(<\langle\text{seller, buyer, product}\rangle, \text{sell, time}\rangle\)

However, the transaction as such may very well be "something that we are interested in, something that we want to gather information about" (cf the first paragraph of this section), i.e. the sales transaction itself may be a strong object candidate. For instance, we may be interested in the quantity sold or the transportation mode for the consignment. Thus the sales transaction may be the natural object part of property type e-constellations like

\(<\text{sales transaction, quantity, time}\rangle\)

\(^1\) Cf section 2.3.1.
<sales transaction, transportation mode, time>

Note that the quantity and the transportation mode cannot in general be regarded as a property of any one of the object constituents of the sales transaction. For instance, the quantity could not be regarded as a property of the product, provided that different quantities of the product can be bought. Neither could it be regarded as a property of the buyer, provided that the buyer may buy different quantities on different occasions. Etc. Thus in connection with phenomena like transactions and events there are properties for which there is no feasible object smaller than the transaction itself, and this is the justification for compound objects consisting of constellations.

End of example.

An interesting problem area is formed by the questions of birth, change, and death of objects. When should a particular object be considered to be born? What changes of properties could a particular object stand without being killed or transformed into another object? Are dead objects of interest?

The identity problems, as the birth, change, and death problems may also be labeled, are particularly delicate in connection with compound objects consisting of a set of other objects. Example: a household consisting of individual persons. What happens if one of the constituent objects of such a set dies or disappears because it no longer fulfills the formal condition for being a part of the compound object? Often it is natural to let an object keep its life and identity as long as the separate changes are small and do not occur too closely in time. This might be a better solution than tying the life and identity of the object to one particular, "vital" property, provided that one is able to define formally what is to be regarded as "small changes" and "not too closely in time".

Example. Figure 1 illustrates the problem of identity of compound objects. If we tie the existence of $0_1$ to the existence of member object $a$, then the compound object $0 \simeq 0_1 \simeq \ldots \simeq 0_5$
$0_1 = \{a, b\}$

$0_2 = \{a, b, 1\}$

$0_3 = \{a, b, 1, 2\}$

$0_4 = \{a, b, 1, 2, 3\}$

$0_5 = \{a, b, 1, 3\}$

$0_6 = \{b, 1, 3\}$

$0_7 = \{b, 1, 3, 4\}$

$0_8 = \{b, c, 1, 3, 4\}$

$0_9 = \{b, c, 1, 3, 4, 5\}$

$0_{10} = \{b, c, 3, 4, 5\}$

$0_{11} = \{c, 3, 4, 5\}$

**Figure 1.** The problem of identity of compound objects ($0_i$ is a compound object existing at time $t_i$).
should be considered to have died between $t_5$ and $t_6$. But although $0_1$ and $0_{11}$ have no constituent objects in common, we may find it more natural to consider each of the transitions $0_1 \approx 0_{i+1}$ as such, i.e. $0 \approx 0_1 \approx \ldots \approx 0_{11}$.

**Remark.** The identity problem is a very old philosophical problem. It was tackled already by Heraclitus, a Greek philosopher of the sixth century B.C. See Wedberg [12]. A mathematician might find that the problem has something in common with the problems of analytical continuation of functions.
2.2.2 Properties

In the previous section we informally defined an object to be something we are interested in, something we want to know something about. Then one might ask: what is it that we want to know about the objects? The answers to such questions can be regarded to fall into two main categories. Intuitively, we may either want to inform ourselves about the properties of an object or about the object's relations to other objects. Accordingly, within the general infological framework, there are two fundamental concepts, which are called "property" and "object relation". In this paragraph we shall treat properties.

Intuitively, properties are generative in the following sense. If

\[ p_1: \text{"to be tall"} \]

and

\[ p_2: \text{"to be lean"} \]

are two properties, then there is also a property

\[ p_3: \text{"to be both tall and lean"} \]

Moreover, \( p_1 \) and \( p_2 \) intuitively generate properties like

\[ p_4: \text{"to be either tall or lean"} \]

\[ p_5: \text{"not to be tall"} \]

\[ p_6: \text{"to be either tall or not lean"} \]

\[ p_7: \text{"to be both lean and not tall"} \]

and so on. We shall formalize the obvious generating property of properties by requiring that the set of properties of any particular infological model should be closed under any property


generation rule specified for the model; the set of property
generation rules should be stated during the specification
process (of figure 5 of chapter 1) in the same way as the set
of objects, the set of properties, and the set of object
relations are stated during that process. Formally then, we
could phrase the closure restriction in the following way.

(R1) If \( p_1, \ldots, p_n \) are elements in the set of
properties, \( P \), of a particular infological model, and \( g \) is an
element in the set of property generation rules, \( G \), for the
same model, and \( g(p_1, \ldots, p_n) = p_{n+1} \), then \( p_{n+1} \in P \).

Although concepts like "object relation", "time", "constellation",
and "fact" have not yet been thoroughly discussed, we shall
give a few examples of property generation rules, which are
likely to be part of any particular infological model. The
examples may be skipped during the first reading of the chapter.
\( P \) denotes the set of facts, which is defined in section 2.3.1.

Example 1: Generation of conjunctive properties

If \( p_1 \) and \( p_2 \) are properties, then the conjunction generation
rule \( \land \) generates the conjunctive property \( p_3 = \land(p_1, p_2) \),
and for all objects \( o \) and all times \( t \)
\[
\langle o, p_3, t \rangle \in P \iff \langle o, p_1, t \rangle \in F \text{ and } \langle o, p_2, t \rangle \in F
\]

Example 2: Generation of disjunctive properties

If \( p_1 \) and \( p_2 \) are properties, then the disjunction generation
rule \( \lor \) generates the disjunctive property \( p_3 = \lor(p_1, p_2) \),
and for all objects \( o \) and all times \( t \)
\[
\langle o, p_3, t \rangle \in F \iff \langle o, p_1, t \rangle \in F \text{ or } \langle o, p_2, t \rangle \in F
\]

Example 3: Generation of negative properties

If \( p_1 \) is a property, then the negation generation rule \( \neg \) gen-
erates the negative property \( p_2 = \neg(p_1) \), and for all objects

\( \neg \) "iff" = "if and only if"
\( o \) and all times \( t \)

\[ \langle o, p_2, t \rangle \in F \text{ iff } \langle o, p_1, t \rangle \notin F \]

**Example 4:** Generation of time-dependent properties

If \( p_1 \) is a property, and \( t' \) is a time, then the generation rule \( g_t \) generates the time-dependent property \( p_2 = g_t(p_1) \), and for all objects \( o \) and all times \( t \)

\[ \langle o, p_2, t \rangle \in F \text{ iff } \langle o, p_1, t' \rangle \in F \]

**Example 5:** Generation of relation-dependent properties

If \( p_1 \) is a property, and \( R \) is an \( n \)-ary object relation, and \( i \) is an integer not less than \( 1 \) and not greater than \( n \), then the generation rule \( g_R^i \) generates the relation-dependent property \( p_2 = g_R^i(p_1) \), and for all objects \( o \) and all times \( t \)

\[ \langle o, p_2, t \rangle \in F \text{ iff } \langle o, p_1, t \rangle \in F \text{ and there are objects } o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n \text{ which satisfy } \langle o_1, \ldots, o_{i-1}, o, o_{i+1}, \ldots, o_n, R, t \rangle \in F \]

**Illustration:** The property "to be born in 1970 and have a father who is now, 1973, a millionaire" can be regarded as the conjunction of a time-dependent property and a relation-dependent property.

If \( P \) is the set of properties of a particular infological model, we shall call any subset of \( P \), which together with the property generation rules generate all elements of \( P \), a \( P \)-**generator**. A \( P \)-**basis** is a \( P \)-generator which is **minimal** in the sense that no element can be excluded from it without making the generated set of properties only a proper subset of \( P \).

When we are about to specify the properties of a particular infological model, we will certainly not find it practical to specify explicitly all elements of \( P \). Instead we will specify \( P \) implicitly by stating
(a) a set of property generation rules, and

(b) a P-generator

The P-generator, which we specify explicitly, may or may not be a P-basis. If it is a basis, then all its elements have to be fundamental properties, i.e., properties which cannot be generated from other elements in the specified P-basis. Proof: If one property in the specified P-basis were formally derivable from other properties in the P-basis by application of the generation rules, then the derivable property could be excluded from the P-basis without changing its status as a P-generator. But then the original basis would not be minimal, as defined above, and the basis would not be a basis. Reductio ad absurdum.

Thus, if we include derived properties, properties which are formally defined, in the set of properties, which we specify explicitly for a particular infological model, the specified P-generator will not be minimal, i.e., it will not be a P-basis. It may be practical for several reasons to specify explicitly formally derivable properties. In so doing we recognize them as autonomous entities, and we may assign distinct, atomic names to them (see chapter 3). This in turn facilitates man-data base communication as well as the introduction of quality control procedures utilizing redundancy.

Formal definitions are definitions expressed in terms of specified entities in a particular infological model, e.g., specified properties and specified generation rules. Thus formal definitions are internal to the infological model. Formally undefined entities are often operationally defined to those who serve and are served by the system. However, these definitions are informal in the sense that they are external to the infological model, i.e., they are not expressed in terms of entities which are specified in the particular

---

1 An example of such a quality control procedure: we register both "birth date" and "age now", where we know the value of "now", and use the obvious redundancy for a consistency check.
infological model. External definitions will usually be verbal, expressed in a natural language, and they should be known and uniformly interpreted by all human beings directly or indirectly using the information system (cf chapter 1).
2.2.3 Object relations

Whereas properties are tied to individual objects, object relations are tied to pairs or, more generally, n-tuples of objects. The object relations to be part of a particular infological model are specified in very much the same way as the objects and properties are specified, i.e., the decisions have to be guided by a kind of informed judgment. As infological theory develops, more formal criteria might be invented. There are dependencies, of course, between (i) the choice of objects, (ii) the choice of properties, and (iii) the choice of object relations, which we make, when we specify an infological model.

Example. If both persons and employers are specified as objects, we may specify an object relation "employ" characterizing certain \( \langle \text{employer, person} \rangle \) pairs. The property of being a civil servant, which we assume here to be equivalent to having the government as one's employer, could then be regarded as a derived, relation-dependent property (cf. section 2.2.2). If we specify persons but not employers as objects, the object relation "employ" could not be specified.\(^1\) Then the property of being a civil servant would have to be specified as a fundamental property, and any definition of the property would get the status of an informal definition (cf. section 2.2.2).

Every object relation is of a specific degree, a natural number greater than 1\(^2\). Object relations of degree 2 are called binary object relations and seem to be very frequent in most applications. For instance, if persons have been specified as objects in a particular infological model, binary object relations like "to be father of" may be specified. The object relation "sell", tied to \( \langle \text{seller, buyer, product} \rangle \) triples, could serve as an example of tertiary object relations. Fundamental\(^3\) object relations of greater degree than 3 do not seem to be very common.

---

\(^1\) We assume that there are "non-person" employers, e.g., incorporated enterprises.

\(^2\) Formally, it may sometimes be convenient to regard properties as object relations of degree 1.

\(^3\) "Fundamental object relation" is defined below.
Similarly as properties may generate other properties, object relations may generate other object relations, when generation rules are applied. The closure restriction for the set of object relations of an infological model could be stated in the following way.

(R2) If \( r_1, \ldots, r_n \) are elements in the set of object relations, \( R \), of a particular infological model, and \( g \) is an element in the set of object relation generation rules, \( G \), for the same model, and \( g(r_1, \ldots, r_n) = r_{n+1} \), then \( r_{n+1} \in R \).

The following examples of object relation generation rules may be skipped during the first reading of the chapter.

**Example 6:** Generation of conjunctive object relations

If \( r_1 \) and \( r_2 \) are \( n \)-ary object relations, then the conjunction generation rule \( \land \) generates the conjunctive object relation \( r_3 = \land (r_1, r_2) \), and for all object \( n \)-tuples \( \langle o_1, \ldots, o_n \rangle \) and all times \( t \)

\[
\langle \langle o_1, \ldots, o_n \rangle, r_3, t \rangle \in F \iff \langle \langle o_1, \ldots, o_n \rangle, r_1, t \rangle \in F \quad \text{and} \quad \langle \langle o_1, \ldots, o_n \rangle, r_2, t \rangle \in F
\]

**Example 7:** Generation of disjunctive object relations

If \( r_1 \) and \( r_2 \) are \( n \)-ary object relations, then the disjunction generation rule \( \lor \) generates the disjunctive object relation \( r_3 = \lor (r_1, r_2) \), and for all object \( n \)-tuples \( \langle o_1, \ldots, o_n \rangle \) and all times \( t \)

\[
\langle \langle o_1, \ldots, o_n \rangle, r_3, t \rangle \in F \iff \langle \langle o_1, \ldots, o_n \rangle, r_1, t \rangle \in F \quad \text{or} \quad \langle \langle o_1, \ldots, o_n \rangle, r_2, t \rangle \in F
\]

\( F \) denotes the set of facts. See section 2.3.3.
Example 8: Generation of negative object relations

If \( r_1 \) is an \( n \)-ary object relation, then the negation generation rule \( \neg \) generates the negative object relation \( r_2 = \neg(r_1) \), and for all object \( n \)-tuples \( \langle o_1, \ldots, o_n \rangle \) and all times \( t \)
\[
\langle \langle o_1, \ldots, o_n \rangle, r_2, t \rangle \in F \text{ iff } \langle \langle o_1, \ldots, o_n \rangle, r_1, t \rangle \notin F
\]

Example 9: Generation of time-dependent object relations

If \( r_1 \) is an \( n \)-ary object relation, and \( t' \) is a time, then the generation rule \( g_{t'} \) generates the time-dependent object relation \( r_2 = g_{t'}(r_1) \) and for all object \( n \)-tuples \( \langle o_1, \ldots, o_n \rangle \) and all times \( t \)
\[
\langle \langle o_1, \ldots, o_n \rangle, r_2, t \rangle \in F \text{ iff } \langle \langle o_1, \ldots, o_n \rangle, r_1, t' \rangle \in F
\]

Example 10: Generation of property-dependent object relations

If \( p_1, \ldots, p_n \) are properties, the generation rule \( g_r \) generates the property-dependent \( n \)-ary object relation \( r = g_r(p_1, \ldots, p_n) \), and for all object \( n \)-tuples \( \langle o_1, \ldots, o_n \rangle \) and all times \( t \)
\[
\langle \langle o_1, \ldots, o_n \rangle, r, t \rangle \in F \text{ iff } \langle \langle o_1, p_i, t \rangle \in F \text{ for } 1 \leq i \leq n
\]

Among the examples 1-10 there are two, example 5 and example 10, which show that property generation and object relation generation may very well be interdependent. For infological models, where this is the case, (R1) and (R2) have to be replaced with a combined closure restriction, however.

It could read as follows.

(R3) If, for a particular infological model, \( p_1, \ldots, p_m \) are elements in the set of properties, \( P \), \( r_1, \ldots, r_n \) are elements in the set of object relations, \( R \), \( g \) is an element in the set of generation rules, \( G \), and \( y = g(p_1, \ldots, p_m, r_1, \ldots, r_n) \) is generated when \( g \) is applied to \( p_1, \ldots, p_m, r_1, \ldots, r_n \), then \( y \in P \) or \( y \in R \).
Similarly as when one specifies the set of properties of a particular infological model, one does not explicitly specify the entire set of object relations of the model. Instead one states explicitly a generator, i.e., a set of object relations generating together with the set of properties and the set of generation rules the entire set of object relations. The generator may contain both fundamental and derived object relations. A derived object relation is an object relation which may be generated from a subset of the specified generator containing only fundamental object relations; a fundamental object relation is an object relation which is not derived. If the generator contains only fundamental object relations (and fundamental properties), it is called a basis.

Derived object relations, like derived properties, may be formally defined, i.e., defined in terms of entities which are internal to the infological model. For fundamental object relations only informal definitions may be stated. However, informal, verbal definitions, though expressed in terms of entities which are external to the infological model, can and should be operational and informative to the users of the data base.

We have said that derived properties and derived object relations may be formally defined, but we have said nothing about how to express the formal definitions. One way of doing this is to use first-order predicate calculus.

Example. If we have specified $p_1 =$ "to be female" as a fundamental property and $r_1 =$ "to be parent of" as a fundamental object relation, and if we have also specified appropriate generation rules, like those of examples 1 and 5 above, the

---

1 If we specify a common property and object relation generator, the subset may contain both fundamental properties and fundamental object relations.

2 Strictly, we cannot state such definitions or any other internal entities or any particular infological model until we have introduced the concept of names of entities. This is done in the next chapter.
object relation $r_2 = "\text{to be grandmother of}"$ could be specified as a derived object relation formally defined by the formula

$$r_2 \ x \ y \iff (p_1 \ x \land \exists z \ (r_1 \ x \ z \land r_1 \ z \ y))$$
2.2.4 Time

The fourth fundamental infological concept to be introduced is "time". The most convenient procedure for specifying the set of times of a particular infological model will probably be "per constellation type"\(^1\), i.e for each phenomenon to be covered by the infological model one specifies the times of potential interest.

The character of the times of interest may vary from one type of phenomenon to another within the same infological model. Sometimes points of time, without any duration at all, are relevant. Sometimes shorter or longer time intervals are more appropriate.

Example. Consider the situation described by the statement

"Smith was in London last Friday"

How should such a situation be treated within a particular infological model. It depends upon the goals of the model, of course. It seems fairly natural to treat "Smith" and "London" as objects and "geographical location" as a binary object relation. But what about the time, "last Friday"? For some purposes it will be quite sufficient to consider days as points of time, and then there is no problem. If, however, the goals of our infological model forces us to regard a day as a time interval in connection with the geographical location of people, then the statement above is ambiguous. It may either describe the situation that

"Smith was in London during the whole of last Friday",

or it may say that

"Smith was in London during some part of last Friday";

we have to specify carefully, if and how it should be possible

\(^1\) See 2.3.4 for definition of "constellation type".
to represent both kinds of situations within the particular infological model.

Example. For a situation like

"Sweden imported \( x \) tons of coffee in 1970"

it seems most natural to treat the time as an interval, particularly if object system situations like

"Sweden imported \( x_1 \) tons of coffee in January 1970"

are also of interest.

End of example.

By introducing "time" as a fundamental concept of the general infological framework, we recognize that most object systems, for the control of which we might like to design information systems, seem to be dynamic by nature, and that dynamic aspects, talked about in terms of time series, trends, prognoses etc, are often of primary interest. If we were only interested in static systems, or if we were only interested in one "time slice" at a time\(^1\) of a dynamic piece of reality, there would be no need to introduce "time" as a fundamental concept.

An object of a dynamic object system typically has a kind of life, during which it gets and loses properties, enters and leaves relational structures. When the object is born it gets at least one unique property, an identifying property, which it does not lose until it dies. As was pointed out in section 2.2.1 the birth and death problems, the problems of finding or identifying properties which are "natural" or at least suitable for the purpose of the model, are often immense in practice. Not even the birth and death of physical objects

---

\(^1\) Example: if the history of an object system has no prognostic value and has no interest in itself, one might only be interested in the current status, "the current time-slice" of the system.
like persons are always uncomplicated; however forgotten as
a human being, a person may be of administrative interest
for long after his physical death.

Although most interesting object systems are dynamical by
nature, there are of course even in such systems individual
properties and object relations which are stable in the sense
that they are not changed for the objects concerned as long
as the latter are alive. We have already mentioned the
identifying properties. Filial relations would be another
example. Note also the formal stability of time-dependent
properties and object relations generated by generation rules
like those of examples 3 and 9 above. As it may seem
paradoxical that time-dependent properties and object relations
are more stable than others, an illustrative example should
be given.

Example. Suppose A marries B at time t. For some time after
t, A will then have the property of being married to B.
However, this property may be highly instable. On the other
hand, the time-dependent property of A of having married B
at t is perfectly stable. A may very well escape B, but he
may never escape the fact that he once married B.

(This example could obviously be rephrased in terms of
object relations instead of properties.)
2.3 Derived concepts

2.3.1 Constellations

Definition. If \( x \) is an \( r \)-tuple of objects \( (n = 1,2,3, \ldots) \), \( y \) is a property or an object relation, and \( z \) is a time, then the triple \( <x, y, z> \) is called an elementary constellation, e-constellation. \( x \) is called the object component, \( y \) the predicate component, and \( z \) the time component of the e-constellation.

Special case 1. If \( o \) is an object, \( p \) is a property, and \( t \) is a time, then the triple \( <o, p, t> \) is called an e-constellation of property type.

Special case 2. If \( <o_1, \ldots, o_n> \) is an \( n \)-tuple of objects, \( r \) is an \( n \)-ary object relation, and \( t \) is a time, then the triple \( <<o_1, \ldots, o_n>, r, t> \) is called an e-constellation of relational type.

For a particular infological model we should distinguish between

\[
\begin{align*}
C &= \text{the set of all e-constellations} \\
V &= \text{the set of valid e-constellations} \\
F &= \text{the set of facts}
\end{align*}
\]

\( F \) is a subset of \( V \) which is a subset of \( C \). \( C \) contains all triples which are e-constellations according to the definition above. However, when we specify an infological model, we also specify restrictions which make certain e-constellations invalid; for instance we may specify that a particular property is only relevant for a particular kind of objects, that a certain object relation requires the time component to be a time interval and not a point of time, etc. An e-constellation having an object component of degree \( m \) and a predicate component of degree \( n \neq m \) will always be invalid, of course.

Thus \( V \) could be interpreted as the set of all "imaginable" states and events of the slice of reality which we are looking
at through a particular pair of "infological glasses". In chapter 1 we postulated the existence of an objectively true reality. Using this postulate we may define $F$, the set of e-facts, as the set of objectively true e-constellations. An e-fact then is an e-constellation that has occurred or will occur, depending on the time component.

Given the specification of a particular infological model, it will in principle be a "mechanical" task to tell the contents of $C$ and $V$. The model specification is not sufficient, however, to determine whether a particular element in $V$ belongs to $F$ or $\neg F$, i.e. whether a particular e-constellation is a fact or not. If we have implemented a data base corresponding to our infological model and fed it with messages about the object system, we will be able to tell for some, but not all, valid e-constellations whether they are facts or not according to the available information; we might also be able to say something about the likelihood that the available information correctly reflects the objectively true reality. Thus the character of the set $F$ is quite different from the more formal character of $C$ and $V$. 
2.3.2 Object groups

Definition. The object group $O(p)$, generated by a property $p$, the so-called group property, is the set of all objects which have, have had, or will have property $p$, i.e.

$$O(p) = \{o_i \mid \exists t: \langle o_i, p, t \rangle \in F\}$$

Definition. A set of objects $O$ is an object group if and only if there is a property $p$ which generates $O = O(p)$.

Definition. The time slice $O_t(p)$ of object group $O(p)$ is the subset of $O(p)$ characterized by

$$O_t(p) = \{o_i \mid \langle o_i, p, t \rangle \in F\}$$

Remark. Intuitively, $O_t(p)$ is the set of objects having property $p$ at time $t$. If $t$ is incompatible with $p$, which it is, for instance, if $t$ is a point of time, and the restrictions of the infological model requires $t$ to be a time interval, or vice versa, then obviously $O_t(p) = \emptyset$.\(^1\)

Theorem. It is easy to prove that any object group $O(p)$ is the union set of all time slices $O_t(p)$, i.e.

$$O(p) = \bigcup_t O_t(p)$$

For time-dependent properties\(^2\) $p = p(t_1, \ldots, t_n)$ all non-empty time slices. $O_t(p)$ are identical with the whole object group.

Example. The object group

"persons born in 1900, working in 1920 as miners, and being still alive"

\(^1\) Cf the definition of $V$, the set of valid e-constellations in the previous section.

\(^2\) Cf example 4 of section 2.2.2. See also the "marriage example" of section 2.2.4. Formally a time-dependent property may be defined as a property $p$, for which, for an "arbitrary but fixed" object $o_i$, all valid e-constellations $\langle o_i, p, t \rangle$ are either in $F$ or in $V \setminus F$. 
is generated by a time-dependent property \( p(t_1, t_2, t_3) \) where \( t_1 = 1900, t_2 = 1920, \) and \( t_3 = \) "current time".

End of example.

It follows from the definition that every property of an infological model may be a group property generating an object group. Two arbitrary object groups may be related to each other in three different ways: they may have no object in common, or they may partially overlap each other, or one may be a subgroup of the other.

**Example.** In the infological model for a university administration system "persons" is very likely to be an object group. This object group may contain important subgroups like "teachers", "students", and "administrative staff". These subgroups need not be mutually exclusive; e.g. there may be students, who are also teachers.

The "object group" concept may be used during the specification process. For instance the task of specifying the objects of the infological model cannot be carried out by enumerating explicitly all objects to be part of the model. Instead it could be recommended to search for a number of "important" and stable\(^1\) properties generating a set of mutually exclusive object groups which exhaust the set of objects of interest. Together these object groups, which we may call object types\(^2\), constitute a classification of the objects of the infological model.

Thus we do not specify the objects of an infological model in terms of individual objects like "Mr Alexander", "Jones & Jones Inc", "Miss Smith", "The Labour Party", "steel", "coffee", "Mrs Gold", "Ford Company", "Sweden", and so on, but rather in terms of object types like "persons", "enterprises", "political associations", "goods", and "countries".

\(^1\) Cf section 2.2.4.

\(^2\) The group property of an object type is said to be the type property of the object type.
When we proceed to specify the properties and object relations, it may be necessary to distinguish between different subgroups within the object types, because the same properties and object relations may not be of interest, "relevant", to all objects of an object type. We return to this question later.
2.3.3 Attributes

Definition. If

(a) \( O(p) \) is an object group generated by the property \( p \), and

(b) \( A = \{v_i\} \) is a set of properties, and

(c) for every time slice \( O_t(p) \), every object which is contained in \( O_t(p) \) is also contained in at least one of the corresponding time slices \( O_t(v_i) \) generated by the properties \( v_i \) in \( A \),

then

(d) \( A \) is said to be an attribute which is relevant to the object group \( O(p) \), and

(e) the elements \( v_i \) of \( A \) are called the values of the attribute \( A \).

Definition. A set of properties \( A \) is an attribute if and only if there is an object group for which \( A \) is relevant according to the preceding definition; the object group is called a relevance group of the attribute.

If we substitute "exactly one" for "at least one" in condition (c) above, we get the definition of a single-valued attribute. Single-valued attributes will also be called variables. Attributes which are not variables will be called multiple-valued attributes.

Suppose \( A \) is a single-valued attribute, i.e., a variable, with the relevance group \( O(p) \). Also suppose that the object \( o_i \) is a member of \( O(p) \) at time \( t \). Then, according to the definition, there is exactly one property, one value \( v_{ij} \), of \( A \), which \( o_i \)
has \( j \) at \( t \). We shall say then that the attribute, the variable, \( A \) takes the value \( v \), for the object \( o_j \) at time \( t \).

Attributes may be classified into different categories depending upon different structural similarities between the relevance group of the attribute and a set of numbers, e.g., the set of rational numbers, \( Q \). We shall look at a couple of examples of such categories.

Example 1. An attribute \( A \) belongs to category \( C_1 \) if and only if

(a) there is a homomorphic mapping \( f \) between \( \langle O(p), R_1, R_2 \rangle \) and \( \langle Q, <, = \rangle \), where

(b) \( O(p) \) is a relevance group of \( A \),

(c) \( R_1 \) and \( R_2 \) are binary object relations,

(d) \( Q \) is the set of rational numbers, and

(e) "<" and "=" are the binary "less than" and "equal to" relations respectively, defined as usual in \( Q \).

The condition (a) implies that if \( o_i \in O(p) \) and \( o_j \in O(p) \) then

(f) \[ o_i R_1 o_j \text{ iff } f(o_i) < f(o_j) \]

and

(g) \[ o_i R_2 o_j \text{ iff } f(o_i) = f(o_j) \]

---

1 An object \( o_j \) is said to have a property \( p_j \) at time \( t \) if and only if \( \langle o_j, p_j, t \rangle \in F \). Similarly, the \( n \)-ary object relation \( R \) is said to hold between the objects \( o_1, \ldots, o_n \) at time \( t \) if and only if \( \langle o_1, \ldots, o_n, R, t \rangle \in F \).
Being fundamental or derived object relations of a particular
infological model, \( R_1 \) and \( R_2 \) are supposed to have so-called
"empirical and systematic import". Without this restriction
condition (a) could always be trivially provided for with the
aid of "postconstructions".

Attributes belonging to category \( C_1 \) could be called comparative
attributes. The attribute "hardness" with the relevance group
"minerals" is a typical comparative attribute, for which the
object relations corresponding to \( R_1 \) and \( R_2 \) have obvious
empirical and systematic import; they could be informally\(^1\)
but operationally defined by reference to the scratch test\(^2\):

"a mineral \( x \) is called harder than another mineral, \( y \), if a
sharp point of \( x \) scratches a smooth surface of \( y \); and \( y \) are
said to be of equal hardness if neither scratches the other".

Example 2. An attribute \( A \) belongs to category \( C_2 \) if and only if

\[(a) \text{ there is a homomorphic mapping } f \text{ between } \langle 0(p), R_1, R_2, R_3, R_4 \rangle \text{ and } \langle Q, +, -, \times, / \rangle, \]
where

\[(b) \text{ } 0(p) \text{ is a relevance group of } A, \]

\[(c) \text{ } R_1, R_2, R_3, \text{ and } R_4 \text{ are tertiary object relations, } \]

\[(d) \text{ } Q \text{ is the set of rational numbers, } \]

\[(e) \text{ } +", "-", "\times", \text{ and } "/" \text{ are the addition, subtraction,}
\text{ multiplication, and division operators}\(^3\), \text{ defined as}
\text{ usual in } Q \]

---

\(^1\) Cf section 2.2.3

\(^2\) Hempel

\(^3\) Note that the arithmetical operators may be regarded as
tertiary relations; for instance \( \langle 1, 1, 2 \rangle \in "+", \)
\( \langle 1, 1, 0 \rangle \in "-", \langle 2.5, 2.5, 6.25 \rangle \in "\times", \text{ and }
\langle 7, 2, 3.5 \rangle \in "/".\)
Attributes belonging to category $C_2$ could be called quantitative attributes. The attribute "weight" with the relevance group "physical objects" is a typical quantitative attribute. Obviously, weight sums, weight differences, weight products, and weight quotients have both empirical and systematic import.

It should be evident from the examples above, how other categories of attributes, which one meets in the literature in connection with scales and measurement and related subjects, may be formally treated within the infological theory.

In section 2.2.4 we defined an identifying property as a property which is stable and unique for a particular object during the life-time of the object. We shall now define what is meant by an identifying attribute.

**Definition.** If $a$ is a single-valued attribute, i.e., a variable, and $O(p)$ is an object group, to which $A$ is relevant, and $O_{t'}(p)$ and $O_{t''}(p)$ are two time-slices of $O(p)$, and $o_i$ and $o_j$ are two different objects which are contained in both time-slices, and $A$ takes the value $v_{i}$ for $o_i$ at both $t'$ and $t''$, and $A$ takes the value $v_{j} \neq v_{i}$ for $o_j$ at both $t'$ and $t''$, and all this is true for any times $t'$ and $t''$ and any objects $o_i$ and $o_j$, then $A$ is an identifying attribute within the object group $O(p)$.

**Definition.** A generally identifying attribute is an attribute which is identifying within all object groups, to which it is relevant.

**Example.** The attribute "name" may be identifying within the group of persons constituting a school class, but it will probably not be identifying in an object group consisting of all persons living in a particular town. Thus "name" will not be a generally identifying attribute in an infological model where the two mentioned groups of persons are object groups, to which "name" is a relevant attribute.
2.3.4  Constellation types

Definition.\(^1\) If \(x\) is an \(n\)-tuple of object groups (\(n = 1,2,3,\ldots\)), and \(y\) is an attribute or an object relation, then the pair \(<x,y>\) is called an elementary constellation type. \(x\) is called the object component and \(y\) the predicate component of the e-constellation type.

Special case 1. If \(0\) is an object group, and \(A\) is an attribute, then the pair \(<0,A>\) is called an attribute e-constellation type.

Special case 2. If \(<0_1,\ldots,0_n>\) is an \(n\)-tuple of object groups, and \(R\) is an \(n\)-ary object relation, then the pair \(<<0_1,\ldots,0_n>,R>\) is called a relational e-constellation type.

Note the "natural" correspondence between e-constellations and e-constellation types. The attributive e-constellation type \(<0,A>\) and the property type e-constellation \(<o,p,t>\) are said to be corresponding to each other, if and only if \(o \subseteq 0\) and \(p \subseteq A\). Similarly the relational e-constellation type \(<<0_1,\ldots,0_n>,R>\) and the relational e-constellation \(<<o_1,\ldots,o_n>,r,t>\) are said to be corresponding to each other, if and only if \(o_i \subseteq 0_i,\ldots,o_n \subseteq 0_n\) and \(r = R\).

In section 2.3.1 we defined \(V\), the set of valid e-constellations, as the set of e-constellations which do not violate any formal restrictions of the particular infological model under consideration. Valid e-constellation types may be defined quite analogously, but in order to increase the usefulness of the definition, we choose a slightly different formulation.

Definition. An e-constellation type \(<x,y>\) is valid if and only if

\(^1\) Cf the definition of "elementary constellation" in section 2.3.1.
(a) \( \langle x, y \rangle \) is either an attributive e-constellation type \( \langle 0, A \rangle \) or a relational e-constellation type \( \langle \langle 0_1, \ldots, 0_n \rangle, R \rangle \), and

(b) the attribute A is relevant\(^1\) to the object group 0, if the e-constellation type is attributive, and

(c) the object relation R is relevant\(^2\) to the object groups 0\(_1\), ..., 0\(_n\), if the e-constellation type is relational

In the specification process (cf figure 5 of chapter 1) it may be much more convenient to specify directly the e-constellation types to be part of the infological model, than to specify separately

- the objects, or rather the object types and other object groups, of interest

- the properties, or rather the attributes, and the object relations of interest

- the restrictions, making most potential e-constellation types invalid and some valid

In the specification process, and particularly later, in the design process, it may be practical to be able to talk about different time-versions a constellation type. We shall therefore define such a concept.

---

\(^1\) See section 2.3.3 for definition.

\(^2\) An n-ary object relation R is relevant to the object groups 0\(_1\), ..., 0\(_n\), or rather to the object group n-tuple \( \langle 0_1, \ldots, 0_n \rangle \), if and only if

\[ \langle \langle 0_1, \ldots, 0_n \rangle, R, t \rangle \in V \]

for some time t and for any objects 0\(_i\) \( \in 0_i \); \( i = 1, \ldots, n \).
Definition. If $<x, y>$ is a valid e-constellation type, and $t$ is a time, and there is a valid e-constellation $<x_i, y_j, t>$ corresponding to $<x, y>$, then $<x, y, t>$ is said to be a valid time-version of the e-constellation type $<x, y>$.

The number of time-versions per e-constellation type, which the users consider to be of interest, may be an important design parameter.

As we shall see in later chapters, the set of e-constellation types, which has been specified for a particular infological model, will constitute a basis for the file formation part of the design process. Files corresponding to e-constellation types will be called elementary files, e-files. During the design process, several e-files may be consolidated into one large c-file for storage and retrieval efficiency reasons. We shall say that the c-files correspond to consolidated constellations, c-constellations. For instance, if $0$ is an object group, and $A_1, \ldots, A_m$ are attributes, then the tuple $<0, <A_1, \ldots, A_m>>$

with the object component $0$ and the predicate component $<A_1, \ldots, A_m>$ could be a valid c-constellation type, provided that $A_1, \ldots, A_m$ are all relevant to $0$.

---

1 Recall the definition of "corresponding" stated above.
3. Information entities

3.1 References

In the previous chapter we introduced the entities of the object system part of the infological framework. Thus we discussed the fundamental concepts of objects, properties, object relations, and times, as well as a number of derivable concepts like those of object groups, attributes, and e-constellations. In order to be able to store, communicate, and process information about the object system, we need another kind of fundamental entities by means of which we can refer to the object system entities. That is why we introduce in this chapter the concept of references. The object system entity which a particular reference refers to will be called the (object system) target of the reference.

Though "reference" like "object", "property", "object relation", and "time", is a fundamental – and thus formally undefined concept – in the general infological framework, we shall try in this section to convey an intuitive feeling for the reference concept, as we tried to do it for the four other basic infological categories in the previous chapter.

We use references when we refer to phenomena in reality. The references may be "mental", as when we think about reality, or more tangibly represented (by data\(^1\)) in the reality outside the human minds, as when we talk about reality, or when we use extensions like paper and pencil or computers to store and process knowledge about reality.

Note that a reference is neither identical with the entity to which it refers, nor with the entity, the set of data, which represents it. The same object system phenomenon may be referred to in several ways, and different data configurations may be used to represent the same reference. The latter part of this proposition is obvious, and in order to be convinced of the former part of it, we may consider the following two statements about some object system:

\(^1\) Cf chapter 1.
(i) "The youngest cousin of George married the fair-haired sister of Bill the same day as Mary's grandmother died"

and

(ii) "Jim and Anne got married on the 3rd of April 1965"

These statements report the same fact (or tell the same lie), provided that "the youngest cousin of George" refers to the same person as "Jim", Anne is identical with the fair-haired sister of Bill, and Mary's grandmother actually died on the 3rd of April 1965. Yet it does not seem reasonable to regard the two statements as identical pieces of information, identical messages.

Whereas it is easy to imagine a lot of datalogical variations of (i) and (ii), i.e., different ways of representing on the one hand message (i) and on the other hand message (ii)\(^1\), we thus maintain that there is no strictly datalogical, i.e., strictly representational, way of varying (i) so as to become (ii), or vice versa. We shall say that there are infological differences between (i) and (ii) which cannot be explained merely in terms of datalogical differences; on the other hand, as is also seen by the example, infological differences need not imply factual differences.

There are several reasons, why we should like to consider the two messages as infologically different, even though they might turn out to be factually coincident:

(1) To be conceivable, (i) presupposes a more comprehensive frame of reference with the receiver of the message (human being or computer) than does (ii).

(2) Even though they point to the same entities of the object system, the corresponding references (e.g.

---

\(^1\) Note the dilemma that we get into because we have to use one particular data representation of the messages (i) and (ii) in order to be at all able to talk and write about them. This is not a new philosophical problem, however.
"Anne" and "the fair-haired sister of Bill") in (i) and (ii) contain differences, which we cannot very well maintain to be **datalogical differences** or differences in representation.

(3) The processes used to handle the two messages in a human mind or in a computer are bound to contain basic, infological differences.

As was just pointed out there is a many-to-one relationship between the set of references of an infological model and the set of object system entities specified in the same model. Let us adopt the convention of using capital letters for references and ordinary letters for the objects, properties, etc, themselves. Then, for instance, in the example above we have

\[ \text{Jim} = \text{the youngest cousin of George} \]

whereas

\[ \text{JIM} \neq \text{THE YOUNGEST COUSIN OF GEORGE} \]

The messages

\[ m_1 = \langle \text{THE YOUNGEST COUSIN OF GEORGE, THE FAIR-HAIRED SISTER OF BILL}, \text{GET MARRIED, THE DAY WHEN MARY'S GRANDMOTHER DIED} \rangle \]

and

\[ m_2 = \langle \text{JIM, ANNE}, \text{GET MARRIED, 1965-04-03} \rangle \]

are infologically different, but they both refer to the same constellation

\[ c = \langle \text{Jim, Anne}, \text{get married, 1965-04-03} \rangle \]
3.1.1 Different classifications

A reference always refers to an object system entity or a set of object system entities, the object system target of the reference. Thus, for instance, a reference may refer to an object, an object group, a property, an attribute, an object relation, a time, a constellation, or sets of these entities among which the empty set, $\emptyset$, occurs.

The relationship between a reference and the object system may be characterized in different ways. Accordingly, there are several ways of classifying the set of references. In the subsequent sections we shall discuss three different classifications:

(1) Firstly, a reference may be classified as explicit or implicit, depending upon whether it refers to an object system entity directly or indirectly, via other entities. Section 3.1.2.

(2) Secondly, a reference may be classified as unique or ambiguous. To be able to discuss this categorization we have to introduce the concept of infological context, the environment in which a reference occurs. Because of its infological context, a reference may be expected to refer to a particular kind of entity, e.g., an object. If the actual target is "wider" than the expected target, then the reference is ambiguous. For instance, the expected target could be an object, whereas the actual target turns out to be an object group with more than one object member. Section 3.1.3.

(3) Thirdly, we may naturally classify the references according to the kind of object system entities they refer to, i.e., we may classify the references according to their targets. This will lead us to four basic categories of references, namely
(a) property references
(b) attribute references
(c) object relation references
(d) time references

Objects and object groups may always be referred to via properties\(^1\); thus object references and object group references do not constitute basic categories. Section 3.1.4.

---

\(^1\) For objects this is true under the assumption that there is always at least one identifying property for each object (cf 2.2.4 and 2.3.3). It seems reasonable to introduce this assumption as a postulate for most infological models.
3.1.2 Explicit and implicit references

If we compare the statements (i) and (ii) above, we find immediately that the references of (ii) are much more "straight-forward" than those of (i). For instance, "JIM" refers directly to an object system entity, whereas "THE YOUNGEST COUSIN OF GEORGE" refers indirectly to the same entity. The former is said to be an explicit reference, and the latter is called an implicit reference, or a reference expression.

As is seen from the example, a reference expression contains other references, sub-references. The sub-references in turn may be explicit or implicit. Beside the object system entities which the sub-references explicitly or implicitly refer to, a reference expression refers to a generation rule (cf chapter 2). More precisely, it refers to the rule, which, from the object system targets of the sub-references, generates the object system entity implicitly referred to by the reference expression. Thus the reference expression \( expr(\rho_1, \ldots, \rho_n) \) could be said to be isomorphic with the structure \( g(e_1, \ldots, e_n) \) consisting of the generation rule \( g \) and the object system entities \( e_1, \ldots, e_n \):

\[
expr(\rho_1, \ldots, \rho_n) \\
\uparrow \downarrow \\
g(e_1, \ldots, e_n)
\]

**Example.** Figure 1 illustrates the isomorphism for the reference expression "THE FAIR-HAIRED SISTER OF BILL" of statement (i) above.

In general, reference expressions need not be so sophisticated and natural-language-like as those of statement (i). It will probably be more typical for the expression structures to be obviously parallel to property and object relation rules of the same kind as example 1-10 of sections 2.2.2 and 2.2.3. For instance, the reference expression "LEAN AND TALL" could refer to the property \( \land (\text{lean, tall}) \) generated by the conjunction generation rule defined in example 1 of section 2.2.2.
Figure 1
An explicit reference will often refer to a fundamental object system entity, e.g., a fundamental property (cf. chapter 2), whereas an implicit reference will usually refer to a derived object system entity. However, as was discussed in section 2.2.2, it may facilitate man-data base communication and quality control through consistency checks to specify derived entities explicitly and assign explicit references to them. On the other hand, we saw in figure 1 an example of an implicit reference referring to a fundamental entity. Thus all combinations may occur:

<table>
<thead>
<tr>
<th>fundamental entity</th>
<th>derived entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit reference</td>
<td></td>
</tr>
<tr>
<td>implicit reference</td>
<td></td>
</tr>
</tbody>
</table>
3.1.3 Names and ambiguous references

As was suggested in the previous section, each reference expression possesses an inherent structure. This structure determines for any sub-reference of the expression, what kind of object system entity it refers to. For instance the structure of the reference expression

(i) "THE YOUNGEST COUSIN OF GEORGE MARRIED THE FAIR-HAIRED SISTER OF BILL THE SAME DAY AS MARY'S GRANDMOTHER DIED"

determines that "THE YOUNGEST COUSIN OF GEORGE" refers to an object, that "MARRIED" refers to an object relation, and that "THE SAME DAY AS MARY'S GRANDMOTHER DIED" refers to a time.

The structure of a reference expression may be stated explicitly, separately from the reference itself, but at least partially it may alternatively be implied by the sub-references of the expression. Suppose, for instance, that the sub-reference "MARRIED" of the reference expression (i) above is universally unique within a particular infological model, i.e., in whatever infological context "MARRIED" occurs, it refers to one and the same uniquely identified object system entity, in this case a binary object relation. Then wherever "MARRIED" occurs, it imposes a certain structure upon its infological context, i.e., upon the reference expression of which it is a sub-reference. More precisely the occurrence of the sub-reference "MARRIED" in a reference expression implies, under the conditions just stated, the co-occurrence in the same reference expression of two sub-references referring to objects.

An explicit reference which is universally unique will be called a universal name.

An explicit reference, which is not universally unique, but which in its infological context refers to one particular object system entity, will be called a context-dependent name, or a local name.
In communication - between people or between people and databases - it may be convenient to use references as though they were names, albeit one does not know\footnote{Even if one does know that a reference is not unique, it does not follow that one knows how to make the reference unique. The interactive procedure described here is applicable in such situations, too.} if they actually are unique. One then expects constructive feedback from the person (the database), should the reference not be unique within his (its) infological model. After one or more interactions it will hopefully be possible to replace the tentative name, or quasi-name, with a universal name or local name in the original reference expression.

Example. Suppose a person asks a database for the income of somebody. Then he might use "INCOME" as a preliminary, quasi-name substitute for a lot of slightly differently defined income attributes. Thus he might get the return question from the database, whether he means for instance "GROSS INCOME", with the definition ..., or whether he rather means "INCOME NET OF TAX", with the definition ..., or ... Note that procedures like this one make it possible for a user to communicate with a database, even if he is not perfectly familiar with the details of the particular infological model underlying the database.

In summary then, among the explicit references we distinguish between

(1) \textit{universal names}, which refer to a unique object system entity, independently of the infological context

(2) \textit{local names}, which are unique in a particular infological context

(3) \textit{quasi-names}, which are non-unique references, tentatively used where the infological context requires a unique reference

(4) \textit{ambiguous references}, the rest class
Remark. One and the same reference may be a universal name of one object system entity and an ambiguous reference with respect to another entity. Example: "PERSON" is likely to be unique as a property reference, and also as an object group reference, but ambiguous as an object reference. Cf also section 3.1.4, where the concepts of "reference categories" and "potential reference categories" are defined.
3.1.4 Different kinds of targets

Definition. If, for a particular reference ρ, it is possible
to determine, either from ρ itself or from the infological
context of ρ, that ρ refers to an object system entity belonging
to category x, then we shall call ρ an x reference.

In the definition, "x" may be replaced with "object", "object
group", "property", "attribute", "object relation", "time", or
the like. Thus there are object references, object group
references, attribute references, object relation references,
time references, etc.

When we know the infological context of a reference, we should
always be able to tell, whether the reference is an object
reference, a property reference, or something else. When a
reference is taken out of its context, this may not always be
possible. Thus there is need for a "potential reference cate-
gory" concept to be distinguished from the "contextually deter-
mimed reference category" concept defined above.

Definition. If, for a particular reference ρ, there is a valid
infological context ¹ making ρ an x reference according to the
previous definition, then ρ is a potential x reference.

Whereas a reference may never at the same time be for instance
an object reference and a property reference, it may very well
simultaneously be a potential object reference and a potential
property reference. Cf the "PERSON" example of section 3.1.3.

Note also that even when we know the infological context of a
particular reference, ρ, and thus know ρ to be, say, an object
reference in this context, it is perfectly legitimate to
disregard the infological context and label ρ as a potential
property reference. In fact we shall postulate that all object
group references and object references are potential property
references (cf 3.1.1, foot-note); cf also again the "PERSON"
example of section 3.1.3.

¹ A valid infological context of a reference ρ is a reference
equation expr(..., ρ, ...) containing ρ as a sub-reference and
referring to a valid constellation. Cf 2.3 and 3.2.
Postulate. If \( \rho \) is an object group reference then \( \rho \) is a potential property reference\(^1\).

Postulate. If \( \rho \) is an object reference then \( \rho \) is a potential identifying property reference\(^2\).

We shall distinguish between atomic and complex property references.

Definition. An **atomic property reference** is an explicit property reference, i.e. (of 3.1.2) a reference which refers directly to a property.

Definition. A **complex property reference** is a reference expression with the structure

\[ \langle \rho(\text{attribute}), \, \rho(\text{value}) \rangle \]

where \( \rho(\text{attribute}) \) is a sub-reference referring to an attribute, and \( \rho(\text{value}) \) is a sub-reference referring to a value, i.e. a member property of an attribute\(^3\).

References to so-called type properties, which were found in section 2.3.2 to be useful in the specification process, are usually good examples of atomic property references. For instance, "PERSON", "ENTERPRISE", "POLITICAL ASSOCIATION", "GOOD", and "COUNTRY" could refer to type properties\(^4\).

---

\(^1\) Recall from section 2.3.2 that any object group \( O(p) \) is generated by a property and that any property generates an object group.

\(^2\) An identifying property was defined in section 2.2.4 as a property which is stable and unique for a particular object during the life-time of the object.

\(^3\) Recall the definition of "attribute" in section 2.3.3.

\(^4\) Out of their infological contexts the stated references could obviously just as well be considered to refer to object groups. Thus they are good examples of references which are simultaneously potential property references and potential object group references.
As examples of complex property references we could take

(a)  \langle \text{WEIGHT IN KG, 60} \rangle
(b)  \langle \text{HEIGHT, 200 CM} \rangle
(c)  \langle \text{COLOUR, GREEN} \rangle
(d)  \langle \text{EMPLOYER, VOLVO INC} \rangle

In these reference expressions the sub-references "WEIGHT IN KG", "HEIGHT", "COLOUR", and "EMPLOYER" refer to attributes, whereas "60", "200 CM", "GREEN", and "VOLVO INC." refer to attribute members, values (cf 2.3.3).

Within its context a value sub-reference actually has the same target, a member property of an attribute, as the complex property reference, of which it is a part. Out of context, however, the two references have different degrees of uniqueness. For instance, \langle \text{WEIGHT IN KG, 60} \rangle is likely to be a universal name\(^1\), whereas its sub-reference "60" is certainly only a local name\(^1\) of the same property; if we did not know its infological context to be "WEIGHT IN KG" we could not guess what property it refers to.

Example (d) may need some explication. It represents a reference to the property \(p_1\) of being employed by Volvo Inc.

Figure 2a depicts, with a technique to be further discussed in chapter 5, a situation, where an object with the type property "person" has property \(p_1\). It seems reasonable to assume, however, that \(p_1\) is not a fundamental\(^2\) property, but a property which is generated by a rule similar to that of example 5 in section 2.2.2 from a) an object relation "employ", which holds between persons and companies, and b) the property \(p_2\) of having the registration name Volvo Inc.:

\[ p_1 = g("employ", p_2) \]

In figure 2b we have "blown up" the property \(p_1\) into its alleged fundamental constituents; the dotted line embraces the \(p_1\) property.

\(^1\) Recall the definitions in section 3.1.3.
\(^2\) The concepts of "fundamental" and "derived" properties have been defined in section 2.2.2.
Figure 2a

```
person
\|--|\< EMPLOYER, VOLVO INC. >
    \|--|\p_1
```

Figure 2b

```
person
\|--|\EMPLOY
    \|--|\company
        \|--|\p_2
            \|--|\< REG NAME, VOLVO INC. >
```

Legend:
- x object with type
- property
- object relation
If we return to the references then, we have

\[
\begin{align*}
\rho_{11}(p_1) &= \langle \text{EMPLOYER, VOLVO INC.} \rangle \\
\rho_{12}(p_1) &= \text{VOLVO INC.} \\
\rho_{21}(p_2) &= \langle \text{REG NAME, VOLVO INC.} \rangle \\
\rho_{22}(p_2) &= \text{VOLVO INC.}
\end{align*}
\]

It is easily seen that VOLVO INC. is not a universal name\(^1\). In the EMPLOYER context it denotes the property \(p_1\), which is a member of the "employer" attribute, whereas in the REG NAME context it denotes the property \(p_2\), which is a member of the "registration name" attribute.

As a matter of fact the complex references \(\rho_{11}\) and \(\rho_{21}\) are not universally unique either. Like \(\rho_{12}\) and \(\rho_{22}\) they obtain a well-defined target through their infological contexts only. Compare the reference expressions

(e) \(\langle \text{BILL}, \langle \text{EMPLOYER, VOLVO INC.} \rangle, 1970 \rangle\)

which, according to its structure, refers to a property type e-constellation\(^2\), and

(f) \(\langle \langle \text{BILL}, \langle \text{REG NAME, VOLVO INC.} \rangle \rangle, \text{EMPLOY, 1970} \rangle\)

which is seen to refer to a relational type e-constellation\(^2\). Actually both reference expressions report the same object system situation looked at through slightly different infological glasses. In (e) the structure determines the sub-reference \(\langle \text{EMPLOYER, VOLVO INC.} \rangle\) to be the complex property reference \(\rho_{11}\) which we analyzed rather thoroughly above. On the other hand, the structure of (f) determines \(\rho_{21} = \langle \text{REG NAME, VOLVO INC.} \rangle\) to be, not a complex property reference, as when we analyzed it earlier, but what we shall analogously call a complex object reference. Thus complex references referring to

---

\(^1\) Cf section 3.1.3.

\(^2\) Cf section 2.3.1.
<identifying attribute\textsuperscript{1}, value> structures are good examples of references which are simultaneously potential object references and potential property references; the infological context ultimately determines the nature of the target. Similarly complex references referring to <attribute, value> structures, where the attribute is not necessarily identifying, are potential property references and potential object group references at the same time.

As we have stated earlier, several different references may have the same object system target. We shall call such references synonyms. When we specify a particular infological model, we should consider to specify universally unique atomic names as synonyms to those complex references which can be predicted to be frequently used. This may facilitate man–data base communication. For instance, it is obviously more convenient to refer to a property by

"ARAB WOMAN"

than by

"ETHNIC CHARACTERISTIC = A AND SEX = W"

In the former reference expression both "ARAB" and "WOMAN" are assumed to be universal, atomic names, whereas in the latter expression both "ETHNIC CHARACTERISTIC = A" and "SEX = W" are complex property references.

Complex property references will no doubt be the most frequent implicit property references in "ordinary" data base applications. However, there are other kinds of implicit property references as well. Consider for instance the property sub-reference of the expression

"BILL IS AS OLD AS JIM’S YOUNGEST COUSIN"

\textsuperscript{1} Cf section 2.3.3.
The implicitness of the property reference "AS OLD AS JIM'S YOUNGEST COUSIN" is of the same character as the implicitness of the time reference "THE SAME DAY AS MARY'S GRANDMOTHER DIED" in the example used earlier in this chapter.
3.2 Messages

3.2.1 Complete elementary messages

**Definition.** If \(<x, y, z>\) is a reference expression, where \(x\) is an \(n\)-tuple of locally unique object references, \(y\) refers locally uniquely to a generating property\(^1\) or a generating object relation\(^1\), and \(z\) is a locally unique time reference, then \(<x, y, z>\) is called a complete elementary message, a complete e-message. \(x\) is called the object component, \(y\) the predicate component, and \(z\) the time component of the e-message.

A complete e-message is thus a particular kind of reference expression. It always contains an object sub-reference, a predicate sub-reference, and a time sub-reference. These sub-references may in turn be explicit or contain sub-references. The sub-references of an e-message are called conceptual terms, or just terms when there is no risk for confusion with data terms.

In a sense a complete e-message is a minimal information structure. If we remove any of its terms, it does not convey "definite" knowledge any longer; it might possibly tell that one out of a limited set of e-constellations holds. Cf also section 3.2.2.

If we compare the definition above with the definition in the beginning of section 2.3.1, we see immediately that a complete e-message always refers to an e-constellation which is uniquely determined for the particular infological model under consideration\(^2\).

---

\(^1\) A generating property is a property which is a member of the explicitly specified property generator of the particular infological model. Similarly a generating object relation is an object relation which is a member of the object relation generator. Cf 2.2.2 and 2.2.3.

\(^2\) In practice many problems arise because the sender and the receiver of a message are not perfectly aware of each other's infological models, or frames of reference. This is equally true even if the sender happens to be a data base.
In accordance with the functional mapping between complete e-messages and e-constellations we may define two important subsets of the set of all complete e-messages. They are:

(a) the set of meaningful complete e-messages, corresponding to the set $V$ of valid e-constellations, and

(b) the set of true complete e-messages, corresponding to the set $F$ of facts

Cf section 2.3.1. As $F$ is a subset of $V$, it follows that all true e-messages are meaningful.

Also in parallel with section 2.3.1 we shall state two important special cases of the definition given above.

**Special case 1.** If $\langle x, y, z \rangle$ is a reference expression, where $x$ is a locally unique object reference, $y$ refers locally uniquely to a generating property, and $z$ is a locally unique time reference, then $\langle x, y, z \rangle$ is called a complete elementary message of property type.

**Special case 2.** If $\langle x, y, z \rangle$ is a reference expression, where $x$ is an $n$-tuple of locally unique object references, $y$ refers locally uniquely to an $n$-ary generating object relation, and $z$ is a locally unique time reference, then $\langle x, y, z \rangle$ is called a complete elementary message of relational type.
3.2.2 Incomplete and consolidated messages

In section 3.1.1 we stated that if the actual target of a reference is "wider" than is expected from the infological context of the reference then the reference is ambiguous.

Definition. An incomplete elementary message, an incomplete e-message, is a reference expression which satisfies the conditions for being a complete elementary message according to the definition in the previous section, except that at least one of its sub-references is ambiguous.

For instance, an incomplete e-message may have an object component which is an object group reference rather than an object reference or a predicate component which is an attribute reference, rather than a complex property reference.

Note that queries may be regarded as incomplete messages. For instance, a query like

(i) "WHAT PERSONS HAD PROPERTY p AT TIME t?"

is seen to have an ambiguous object component, viz. the object group reference "PERSONS". The question

(ii) "WHAT AGE IS JIM?"

has an ambiguous predicate component, viz. the attribute reference "AGE". One might believe (ii) to have an ambiguous time component as well. It seems more correct, however, to regard (ii) as an example of a message that contains a conceptual term, which has no explicit counterpart in the data representation of the message. The structure of the incomplete e-message (ii) is more clearly seen if we represent it by

(ii') \(<\text{JIM}, <\text{AGE}, \text{?}>, \text{NOW}\>"

We said earlier that typical of an ambiguous reference is that its object system target is "wider" than is expected from the infological context of the reference. Alternatively one could
say that for any ambiguous reference there is an associated set, the *target set*, which contains more than one object system entity of the kind required by the infological context of the reference.

**Example.** The target set of the ambiguous "PERSON" reference of (i) contains all objects which are persons, i.e. all entities belonging to the object group referred to by "PERSON".

**Example.** The target set of the ambiguous "AGE" reference of (ii) contains all properties belonging to the attribute referred to by "AGE", e.g. \(<\text{age}, 1>\), \(<\text{age}, 2>\), \(<\text{age}, 3>\), etc.

A unique reference then could be defined as a reference which has an associated target set of cardinality 1, i.e. a target set containing exactly one member.

**Theorem.** If \(m = <x, y, z>\) is an e-message and the cardinalities of the target sets of \(x\), \(y\), and \(z\) are \(n_x\), \(n_y\), and \(n_z\) respectively, then the cardinality of the target set of \(m\) itself is equal to \(n_x \times n_y \times n_z\).

**Special case.** If \(m\) is a complete e-message, then \(n_x = n_y = n_z = 1\), and the target set of \(m\) contains exactly one element, one uniquely determined e constellation. Cf section 3.2.1.

**Illustration.** Suppose the attribute "age" of e-message (ii) could take three values: "young", "middle-aged", and "old". Then, if we assume that "Jim" is a unique reference, \(n_x = 1\), \(n_y = 3\), and \(n_z = 1\). It is obvious that there are \(1 \times 3 \times 1 = 3\) different e-constellations "satisfying" the incomplete e-message (ii), viz.

(ii:a) \(<\text{Jim}, <\text{age}, \text{young}>, \text{now}>\)

(ii:b) \(<\text{Jim}, <\text{age}, \text{middle-aged}>, \text{now}>\)

(ii:c) \(<\text{Jim}, <\text{age}, \text{old}>, \text{now}>\)
The e-constellations corresponding to the incomplete e-message (ii) are depicted in figure 3. The dotted lines indicate that only one of the three e-constellations is supposed to be a fact, but that we do not know which one this is. As a contrast, figure 4 shows three e-constellations, all of which are alleged to be facts. This situation could be referred to by the reference expression

(iii) "JIM IS A MAN WHO IS OLD AND DIVORCED"

or

(iii') \langle JIM, \langle SEX, M \rangle, \langle AGE, OLD \rangle, \langle CIVIL STATUS, D \rangle, NOW \rangle

This kind of reference will be called a consolidated message, or a c-message. The c-message above is obviously formed by a consolidation of the complete e-messages

(iv) "JIM IS A MAN",

(v) "JIM IS OLD", and

(vi) "JIM IS DIVORCED"

which have a common object component and a common time component. The predicate component of the c-message is a conjunctive combination of the predicate components of the e-messages. More generally a c-message could be any combination of e-messages. A priori any message, any piece of knowledge or information, will be considered to be a c-message, which, after analysis, may be broken down into e-messages, the minimal information structures.

Remark. The only formal difference between the property $p$, referred to by the predicate component of the c-message (iii), and the properties $p_1$, $p_2$, and $p_3$, referred to by the predicate components of the e-messages (iv), (v), and (vi), is that the latter are generating properties, whereas the former is a generated property; $p = \land (p_1, p_2, p_3)$. Cf section 2.2.2,
particularly example 1, and the definition and foot-note in the beginning of section 3.2.1.
3.2.3 Message types

**Definition.** If \( x \) is an \( n \)-tuple of object group references, and \( y \) is an attribute reference or an object relation reference, then the pair \( \langle x, y \rangle \) is called an **elementary message type** (e-message type) or an **elementary concept** (e-concept). \( x \) is called the **object** component and \( y \) the **predicate** component of the e-message type.

**Special case 1.** If \( \rho(0) \) is an object group reference, and \( \rho(A) \) is an attribute reference, then the pair \( \langle \rho(0), \rho(A) \rangle \) is called an **attribute e-message type**, or attribute e-concept.

**Special case 2.** If \( \langle \rho(0_1), \ldots, \rho(0_n) \rangle \) is an \( n \)-tuple of object group references, and \( \rho(R) \) is a reference to an \( n \)-ary object relation, then the pair \( \langle \rho(0_1), \ldots, \rho(0_n), \rho(R) \rangle \) is called a **relational e-message type**, or relational e-concept.

**Correspondence** between e-messages and e-concepts is defined in much the same way as we defined correspondence between e-constellations and e-constellation types in section 2.3.4. Thus the attributive e-concept \( \langle \rho(0), \rho(A) \rangle \) and the property type e-message \( \langle \rho(o), \rho(p), \rho(t) \rangle \) are said to be corresponding to each other if and only if \( o \in 0 \) and \( p \in A \). Similarly, the relational e-concept \( \langle \rho(0_1), \ldots, \rho(0_n), \rho(R) \rangle \) and the relational e-message \( \langle \rho(o_1), \ldots, \rho(o_n), \rho(r), \rho(t) \rangle \) are said to be corresponding to each other, if and only if \( o_i \in 0_i, \ldots, o_n \in 0_n, \) and \( r = R \).

An e-concept (e-message type) \( \langle x, y \rangle \) is defined as **meaningful** if and only if \( \langle x, y \rangle \) refers to a valid e-constellation type. Cf section 2.3.4.

When we design and use data bases, we shall often need to be able to talk about different time versions of the same e-concept. A meaningful time version of an e-concept is defined in the following way.

\[1\] Cf section 2.3.4 and section 3.2.1.
Definition. If \( \langle x, y \rangle \) is a meaningful e-concept, and \( t \) refers to a time, and there is a meaningful e-message \( \langle x_1, y_1, t \rangle \) corresponding to \( \langle x, y \rangle \), then \( \langle x, y, t \rangle \) is said to be a meaningful time version of the e-concept \( \langle x, y \rangle \).

For the same reason as we introduced the concept of c-constellation types in section 2.3.4, we shall introduce here the concept of consolidated message types (c-message types or c-concepts). A c-concept is a reference expression referring to a c-constellation type. For example, if \( \rho(0) \) refers to an object group, and \( \rho(A_1), \ldots, \rho(A_m) \) refer to attributes, then the tuple

\[
\langle \rho(0), \langle \rho(A_1), \ldots, \rho(A_m) \rangle \rangle
\]

with the object component \( \rho(0) \) and the predicate component \( \langle \rho(A_1), \ldots, \rho(A_m) \rangle \) will be a meaningful c-concept provided that the attributes \( A_1, \ldots, A_m \) are all relevant to the object group 0.
3.2.4 Infological distance and quality of messages

The division of valid constellations into facts and non-facts (section 2.3.1) and of meaningful messages into true and false messages (section 3.2.1) is basically a reflexion from the classical, two-valued logic. This view is very primitive, of course, insufficient and even misleading under many circumstances. There are many situations, where the dualistic "black-or-white-attitude" is clearly unsatisfactory. Two messages may both be false in the sense that none of them corresponds to a fact, but still it may be obvious that one of them should be considered "less false" than the other. There is a strong need for a more sophisticated measure of quality than that implied by two-valued logic.

As an extremely simple illustration we may consider the messages "John is 17 years old" and "John is 33 years old" under the assumption that John is actually \( \frac{34}{2} \) years of age. We have thus one elementary fact,

(a) \(<\text{John, <age, 34 years>, 1972}>\)

and two elementary messages

(b) \(<\text{JOHN, <AGE, 17 YEARS>, "1972"}>\)

and

(c) \(<\text{JOHN, <AGE, 33 YEARS>, "1972"}>\)

none of which reports a fact, i.e. both (b) and (c) correspond to e-constellations, which are not facts. An e-message corresponding to the e-fact (a) would be, for instance

(d) \(<\text{JOHN, <AGE, 34 YEARS>, "1972"}>\)

It is obvious that what might be called the infological distance, \( d \), between the e-message (c) and the hypothetical e-message (d) is smaller than the infological distance between (b) and (d).

The concept of infological distance is still too vague to be used as a basis for a quality measure, which goes beyond a rough,
qualitative classification. It does not seem completely hopeless, however, to make this embryo of an infological quality concept more operational or even to make it quantitative. One must not believe, though, that there is one sole, universally applicable quality measure. Different distances, $d'$, $d''$, etc., should be used in different situations. Neither within our infological model, nor elsewhere is it possible to construct an absolute measure of quality. The quality of a piece of information could not be decided upon by mere inspection of the messages involved. One and the same message may be of inferior quality, if it were to be used as the main information basis of an extremely important decision, and of superior quality as a minor contribution to the total information basis supporting a routine decision or even a more crucial commitment.

We have discussed the infological distance between messages. Corresponding to this distance there is a distance between the real-world phenomena informed about by messages. When we specify a particular infological model for a particular data base, we specify among other things what real-world phenomena or states-of-nature, will be mapped into what e-constellations. When several states-of-nature are mapped into one and the same e-constellation it is reasonable to assume that this mapping decision has been at least implicitly governed by the judgment that the particular states-of-nature are "similar to each other" or "at a small distance from each other". More precisely, if two recognizable states-of-nature, $s_1$ and $s_2$, are mapped into one and the same e-constellation, $c$, when a particular infological model of a particular data base is being specified, then there is an implied judgment that for all future decisions to be supported by the information contents of the data base, it will be of little or no importance which of the two states-of-nature, $s_1$ or $s_2$, actually prevails, if any one of them does. It will be sufficient to know whether $c$ is a fact or a non-fact.

Thus the decisions during the model specification process to map certain "similar" states-of-nature into the same e-constellations will at least implicitly limit the decision area for which the particular data base will be useful. This issue will
be further discussed in section 7.3.2.1. See also Marschak [26], and Emery [24], chapter 4.
3.2.5 Meta-information

As was pointed out in section 2.1, the infological approach to data bases stresses the distinction between:

(a) the real-world phenomena that we are interested in, the object system

(b) information about the object system

(c) data representing information about the object system

A complication to this scheme arises when a data base containing information about a slice of reality has actually been designed, constructed and implemented. At that stage the data base itself will certainly belong to the real-world phenomena that we are interested in, i.e. the data base itself will be part of the object system. For example, a decision-maker using a data base as his source of information should be interested to know something about the quality of the messages that he retrieves from the base. Accordingly, the data base should contain quality information and other information about the information contents of the data base. We shall refer to such information as "information on information". Information on information is made up by messages, the object components of which are themselves messages.

Information on information is one important category of meta-information. The latter concept also covers

- information about the basic constituents of the particular infological model underlying the data base; example: formal and informal definitions of attributes and object types

- information about the data representations of the information contents of the data base; example file descriptions

Remark. Data representations of meta-information will be called
**meta-data.** Subsystems of data bases containing meta-information and data representations thereof may by called **meta-data-bases.**

**Remark.** If we exclude from the object system the phenomena informed about by meta-information and represented by meta-data, we are left with a subsystem of the object system which we shall now and then refer to as "**the object system proper.**"
4. The infological data base

4.1 Introduction

In chapter 1 we discussed what could be called the natural data base, the human mind. We established that this data base has certain limitations. For instance, each natural data base

- is tied to one individual human being, which creates a communication problem in groups and societies of individuals

- has a limited storage capacity, and its contents may easily be distorted, which may lead to false deductions and bad decisions

- is organized in a way which makes accesses to "distant" parts of it subject to errors and very costly in terms of time and mental effort

We also established in section 1.4 that shortcomings of this kind have led man to create artificial extensions to his own mind in much the same way as he has created tools and machines to amplify his physical capabilities. These mental extensions, like spoken and written language, books, abacuses, slide-rules, and computers, have implied the use of data, and they have enabled human beings to store, process, and communicate knowledge more efficiently.

However, it is obvious that artificial data collections are subject to limitations similar to those stated above for natural data bases. In particular, when the size of a data collection increases, it becomes an imperceivable entity, and in order to be of any use at all, it has to be organized as a system consisting of perceivable components. Such a system will be called an artificial data base, or a designed data base, or just a data base. Like the "natural" concepts constitute the infological basis for the organization of the human mind\(^1\), the specified entities of a particular infological model\(^2\) form the infological ground for the design of a data base. This topic will be discussed further in subsequent chapters.

\(^1\) Chapter 1, figure 1.
\(^2\) Chapter 2 and 3.
4.2 The black box reservoir of information

The definition of the data base concept, which was given at the end of the previous section, should not be regarded as the data base definition. It seems premature as yet to state such a general definition. Besides, it seems justified and necessary to emphasize slightly different aspects of the data base concept to different categories of people: users, systemeers, computer specialists, researchers, and so on. If any one of these aspects should be considered more fundamental than the others, it has to be the users’ aspect, because ultimately it is for the users that we design and implement data bases. It is equally obvious, however, that the systemeers, etc, will need a more elaborate version of the data base idea than any user should need.

From a user’s point of view a data base may (and perhaps should) be regarded as nothing more and nothing less than an information reservoir, which, like his own frame of reference, may supply him with information for his decisions.

In order for it to be practically feasible for a user to view a particular data base as a black box reservoir of information, the internal structure of the data base must contain functions which are far more sophisticated than the functions of the so-called generalized data base management systems which are commercially available today.\(^1\) If we assume that such advanced data base systems can and will be constructed, the user will be relieved of a lot of problems and disturbing details which are truly irrelevant to him. The user need not know anything about the datalogical file structure, for instance.

However, no black box view or sophisticated data base management routines can of course relieve the user of problems which are not assignable to the data base as such. As was mentioned in the previous section there are many problems which are common to the utilization of all kinds of information

\(^1\) For a further discussion of required features, see chapter 7.
sources. For instance, in order to be able to utilize the information potential of the data base, the user has to communicate with the data base, like he has to communicate with human information sources, if he desires to utilize them. The communication may take place at a slow or at a fast rate, and with more or less effort on the user's part, depending on (a) the nature and complexity of the user's problem and information need, and (b) the design of the data base. ¹ The same factors determine how well the data base matches the quality requirements of the user. When a person utilizes a data base, he is up to essentially the same interpretation and confidence problems as when he uses any external² source of information, artificial or human. As always when he directly or indirectly uses an external source of information he must question different aspects of the information quality: "Are there uncertainties, which are critical to my problem or decision? Could the information in the data base be deliberately or inadvertently biased for some reason? What are the information sources of the data base? What particular infological model is underlying the data base? Do I mean the same thing by this or that as the data base and the informers of the data base?" A well designed data base should assist the user in his search for such "information on information"³, but the underlying problems are always there, be the information source a data base or something else.

Naturally the desiderata of the users are of fundamental importance, when a data base is to be designed. There are also other interests to be considered in the design process, e.g. the desiderata of the objects of the data base, i.e. the persons, enterprises, authorities, etc., about which information...

¹ People who say things like "there will be no need for clever designs in a couple of years, when hardware prices have gone down" seem to underestimate the dynamic aspect of (a). Information processing is certainly still a field where "the appetite tends to grow while one is eating", as the old proverb says.

² By "external" is meant here "external to the user's own mind". Note that probably most of the contents of the user's frame of reference will somehow, at least indirectly, depend on other sources than his own observation (cf chapter 1, figure 4)

³ "Information on information" consists of messages, whose object components are themselves messages. Cf section 3.2.5.
is planned to be stored, and the desiderata of those who are to implement and run the database. The goal-setting process will be discussed to some depth in chapter 7. However, it should be pointed out at once that substantial parts of the goal-setting discussions in connection with the design of a database can be carried out with great precision even with the "black box" attitude towards the design object, provided only that the participants in the discussions are reasonably well acquainted with the infological frame-work presented in the two preceding chapters.

A first visualization of the database black box, its environment, and the communication which is assumed to take place between the database and its environment, is shown in figure 1. As we stated in section 1 of this chapter, a database is an imperceivable system, consisting of perceivable subsystems, or components. Applying the "black box" approach to the database means that we do not care about the details of the subsystem structure and the nature of the components. Instead we discuss input and output to and from the system, and the relations between input and output. The input emanates from the environment, and the output is delivered back to the environment. The environment of a system may be defined as the set of conditions that are relevant to but not under the control of the managers of the system. However, we have to be careful when we use such a definition in general discussions of database design, because what is under the control of the goal-setters and designers will vary from one database design situation to another. A database is often a component in a broader information system, which may or may not be designed or redesigned in connection with the design of the database. Thus in a practical situation the boundaries between the controllable main system and the uncontrollable environment should perhaps be drawn in a different way than is done in the subsequent discussion.

The inputs to, and the outputs from, a database will be called database transactions with a common name. From an infological point of view the database transactions are conceptual.

1 Churchman [23], p 63.
messages. From another point of view they may be regarded as data representations of conceptual messages, i.e. as data messages; this view may be justified during late stages of the data base design process, and during critical phases of implementation and maintenance.

The input and output transactions of a data base system usually appear in pairs: an input transaction causes the appearance of an output transaction. If we call the data base an organism, an input transaction a stimulus, and the resulting output transaction a response, we recognize the basic parts of so-called S-O-R model.\(^1\)

A stimulus S resulting in a response R may also transform the organism: \(0 \rightarrow 0'\). The response R may in turn act as a stimulus upon the environment and produce another input transaction \(S'\) and an environment transformation \(E \rightarrow E'\). We shall call the septuple

\[(a) \quad <S, O, O', R, E, E', S'>\]

a closed interaction and the quadruple

\[(b) \quad <S, O, O', R>\]

an open interaction.

Sometimes a response may occur without being caused by an identifiable, preceding stimulus. This phenomenon is called spontaneous behaviour and is sometimes claimed to be typical of living organisms\(^2\). According to this criterion a data base may very well be considered a living organism; for instance, it may be impossible to trace a "time-to-reorganize"-signal from the data base back to any particular input transaction. Similarly we may imagine situations, where an input transaction does not cause any output, and where organism transformations may or may not take place. An automatically initiated and performed reorganization may exemplify organism transformation.

---

\(^1\) See for instance Wärneryd [13], p 67, 376.
\(^2\) See for instance Carro-Yancazas [22], p 17 f.
without either input or output. Etc. There are $2^3 - 1 = 7$
different main types of open interactions, some of which will
be treated in detail in chapter 5.
4.3 Environment subsystems

Still regarding the data base itself as a black box reservoir of information, we shall in this section define a workable subsystem structure of the data base environment. The structure, which is visualized in figure 2, is meant to serve as a basis for subsequent discussions of

(a) different kinds of data base interactions (chapter 5), and

(b) typical goals and missions of a data base system (chapter 7)

Figure 2 is a revised version of figure 1, where the data base environment has been broken down into eight subsystems. The arrows indicate the principal information and control flows between the environment and the data base and between the different environment subsystems.
Figure 2
4.3.1 Observed and controlled object system

The formal infological aspects of the object system were discussed in chapter 2. If we define the object system of a data base as the slice of reality which has any significance for the existence and functioning of the data base, we see that all systems indicated by figure 2, including the data base itself and its designers, are actually parts of the object system. When we refer to the object system in two of the subsystem boxes of figure 2, we think of the object system in a more restricted sense.

By the controlled object system we mean the slice of reality which is consciously affected by the decisions taken by the ultimate users of the data base, on an information basis which is at least partially supplied by the data base.

By the observed object system we mean the slice of reality referred to by the (non-meta\(^1\)) information in the data base.

Although the observed object system and the controlled object system may sometimes coincide, they need not always do so. Observations about one category of objects, say enterprises, may very well be used in decisions, which primarily concern another category, say persons. Even when the same object types are concerned, the information supply and consumption processes may affect different subsets of individual objects; example: planning for a whole community may be based upon information about the preferences of people belonging to a random sample.

Usually there is considerable overlapping between the observed and the controlled object systems. But differences may become important, for instance in the goal-setting process, where they may result in intricate conflicts of interest.

---

\(^1\) We recall from section 3.2.5 that meta-information, or "information on information consists of messages whose object components are themselves messages. Example: messages informing about the quality of other messages."
4.3.2 Information supply and consumption

The information supply subsystem comprises all kinds of resources and activities engaged in supplying the data base with information concerning the object system to be observed in order to facilitate decision making concerning the object system to be controlled (cf. the previous section). Thus the information supply function covers all processes, persons, and equipment engaged in

- observing and measuring particular phenomena in the object system

- filling in questionnaires

- interviewing

- coding, i.e. classifying or transforming the original observations, etc., into the categories of the particular infological model underlying the data base

- data registration, i.e. representing the observations and the coded information by data messages

- transforming and transmitting data

- preliminary quality control of input information and data

The information consumption subsystem comprises all kinds of resources and activities engaged in the utilization of the information in the data base. Thus the information consumption function covers all processes, persons, and equipment engaged in

- identifying, operatively, information needs in the decision processes for which the data base has been designed

- formulating the information requests in terms of the particular infological model underlying the data base
- transforming the formalized information requests into data transactions, which are submitted to the data base

- receiving and interpreting the answers from the data base

- analyzing the answers, thus making them adequate for effective utilization in decision processes

- influencing upon and making decisions concerning particular aspects of the object system

In many data base environments the information consumption subsystem will be the function which interacts most frequently with the data base. The interaction between the information supply function and the data base will usually be high-frequent, too, more frequent the more dynamic the data base is (cf section 4.5).

Remark. Depending upon the level of ambition, which is set for a particular data base, one and the same task, say coding, error elimination, or "intelligent analysis" of retrieved information, may be allocated to the data base system itself or to one of its environment subsystems; sometimes the task may advantageously be fulfilled in "conversational mode" as a joint effort by the data base and its environment.
4.3.3 Data base goal-stating, decision-making, and administration

Like most undertakings worth systematic study, the design, construction, and operation of a data base requires some kind of goal-stating and decision-making function. This function, like the others, is a system of resources and activities. The activities will be analyzed in chapter 7. The most important resources within the goal-stating and decision-making function are of course the human resources. In this connection it is important to remember, what is often stressed in modern organizational literature\(^1\), namely that those who really set the goals and make the decisions are not necessarily identical with the set of executives who have the formal authority to do so. The real goal-setters and decision-makers are those who have the power to establish or reject goals and to produce or resist change.

The data base goal-stating and decision-making subsystem will naturally be most active during the early stages of data base planning. When the data base has been designed, constructed, and put into operation, activity from this subsystem will only be required occasionally and mainly upon initiative from information supply, information consumption, or data base administration; these activities may concern major infological extensions desired by the consumers, exchange of hardware equipment for techno-economical reasons or for the sake of suppliers' or consumers' convenience. It is the task of the goal-stating and decision-making subsystem to pool and reconcile the interests of the other subsystems during all stages of life of the data base system.

The data base administration subsystem embraces all those planning, systemeering, programming, operation, and supervision activities in connection with the design, construction and running of the data base system, which by formal or informal decisions have been delegated to it from the goal-stating and decision-making subsystem. Among other things data logical and

\(^1\) See for instance Carzo-Yanouzas \([2]\).
infological restructurings of the data base are carried out by this function; datalogical restructuring may be for instance changes of the file structure of the data base in order to speed up response times or save secondary storage, whereas infological restructuring will mean such things as the introduction or elimination of an object type or an e-concept, or the changing of secrecy status of a particular attribute of the infological model underlying the data base. It should be noted that the data base designer himself is a part of the data base administration function, a circumstance which will inevitably complicate any attempt to arrive at an "objectively optimal" solution to a design problem or to "objectively evaluate" the performance of a particular data base.
4.3.4 The computer

The computer subsystem of the data base environment comprises all devices devoted to the running of the data base, as well as all people, processes, etc, engaged in managing and operating these devices. In principle the devices may be of any kind, even manual. Very often they will be components of an electronic computer system.

It may seem strange to separate the computer from the data base and regard it as part of the data base environment. There are several reasons for this, however. Firstly, it is not at all certain that we may buy a new computer or substantially reconfigure the old one at the same time as we design the data base. Thus the computer may very well be a condition which is relevant to but not under the control of the managers of the data base system, and then, according to the definition stated in section 4.2, we should consider it a part of the system environment, rather than a part of the system itself.

Secondly, the same computer system may have to be shared by several applications, of which the data base system to be designed is only one. Even if the data base designers in such a situation should have partial control over the computer configuration, it is convenient to consider the latter a separate system.

Needless to say, the formal separation of the computer system from the data base should not stop us from integrating the design of the two systems, whenever this is found to be desirable and feasible.
4.3.5 Other data bases

Of course the environment of a data base may contain other data bases. The interaction between different data bases of such a data base network may run so smoothly as to give the information consumers the impression that they are interacting with one data base. The question then arises, why we should at all distinguish between "our" data base and the other data bases in the network, the external data bases. The reasons are analogous to the reasons for separating the computer system from the data base (cf section 4.3.4) and will not be repeated here. When at all possible for organizational and other reasons, it may be advantageous to apply a uniform view to the design of all the data bases of a network. When this is not feasible, one could at least try to establish one common infological model, or a set of compatible models, between which transformations are easy to carry out.
4.4 Basic subsystems of the infological data base

In the previous sections we have regarded the data base as a black box reservoir of information. In this section we shall continue to look upon the data base from a strictly infological point of view, but we shall enter the black box and try to identify the principal infological subsystems of the data base. It should be stressed that the aim of this break-down effort is primarily to provide a deeper understanding of the data base concept as such. In chapter 7 we shall return to the break-down problem from the design point of view, and then of course other factors than infological intelligibility have to be considered. Thus the infological break-down and the design break-down need not necessarily coincide. However, the infological understanding of the data base will no doubt guide the designer in his systemeering task, and it will certainly provide a sounder basis for the discussions and negotiations to take place between users, managers, designers, and other interested parties, during the goal-setting process.

Now let us analyze what has to be inside the data base from an infological point of view. We have talked of the data base as a reservoir containing a set of information. What information? The set of messages which have been put into the data base minus the set of deleted messages might seem to be a strong candidate. But consider a data base, which has been fed with the e-messages

1. \( p \) is a well-known politician
2. \( q \) is a famous actress
3. \( p \) and \( r \) are brothers-in-law
4. \( s \) is the fiancé of \( q \)
5. \( r \) murdered \( s \) yesterday

1 For instance the design break-down must aim at facilitating programming and maintenance of the data base system.
2 It has been found convenient to use the single term "systemeering" (cf "engineering") to denote the combinedly analytical and creative task, which is often referred to by the phrase "systems analysis and design". Beside being convenient the term stresses the integrated entirety of the task. Those who perform the task will be called "systemeers" (cf "engineers").
Five elementary messages have been put into the data base. Nevertheless it seems reasonable to claim that the data base contains other messages as well. For instance, it could be argued that the data base also contains the newsworthy (incomplete) e-message

(6) well-known politician's brother-in-law killed fiancé of famous actress yesterday

although this message has not been inserted into the base.

Thus we expect a data base to have not only a reproductive capability, i.e. the capability of perceiving, retaining, and reproducing information on request, but we also assume it to have some deductive power. The origin of this deductive power, which has to be inside the data base itself, will be called the data base schema, or just the schema.

Messages which, thanks to the schema, are derivable from other messages in the data base need not themselves be explicitly stored in the data base. However, there are perfectly legitimate reasons, related to design goals such as retrieval speed and quality control, why one should sometimes want to store derivable messages redundantly. As we are not discussing design problems for the moment, we just establish that the data base beside the schema has another subsystem, which we shall call the data base nucleus, or just the nucleus, and which comprises all explicitly stored messages in the data base, be they redundant or not.

Together the schema and the nucleus subsystems constitute the memory function of the data base. However, like a highly receptive but uncritical and talkative human being may cause its environment a lot of trouble, a data base consisting of nothing but a perfect memory would be vulnerable, unreliable and sometimes even dangerous. Some kind of filter thus seems to be a necessary third component of the data base system even from a strictly infological point of view.
Our view of the principal subsystem structure of the infological data base is summarized in figure 3. Formally we may define the infological data base as a triple

- \( DB = \langle S, N, F \rangle \)

where

- \( S \) is a schema
- \( N \) is a nucleus
- \( F \) is a filter

Together \( S \) and \( N \) determine the set \( M \) of messages which are contained in the data base, the *information contents* of the data base.

We shall devote the rest of this section to investigating the three basic components of the infological data base somewhat more thoroughly.
Figure 3
4.4.1 The schema

From an infological point of view a data base schema is identical with the specification of a particular infological model, as defined in chapters 2 and 3. Thus a schema is a statement of a set of (references to) object types, attributes, object relations, generation rules, constellation types, internal and external definitions, etc.

The data base schema has two distinguishable sub-functions, which we may call the semantic mission and the deductive mission, respectively. We have already exemplified the deductive function above. In this capacity the schema "amplifies" the information explicitly stored in the nucleus N to become the total information contents M of the data base.

The deductive function of the data base schema is tied to the set of derivation rules, i.e., message generation rules, which determine how messages in the data base generate other messages. Naturally a lot of derivation rules are implied by those generation rules and formal definitions that we discussed in chapter 2, but there may also be other derivation rules.

For instance, some of the derivation rules may have the character of "empirical laws"¹, i.e., laws which are not logical tautologies but yet seem plausible on the basis of available empirical evidence. Sometimes these statements are probabilistic and involve references to statistical distributions. An advanced data base might even be able to induce empirical laws automatically from its own information contents. Naturally, such data bases could be particularly helpful in disciplines where data are abundant, although powerful theory is lacking.

Empirical laws, and even weaker correlations between attributes, etc., may be very useful for other than deduction purposes. For instance, the filter function may use them to protect the data base and its users against false messages. They may

¹ Cf chapter 1, figure 1.
also guide the data base administrator in his continuous search for more efficient datalogical structurings of the data base. An advanced data base might itself suggest that a file structure, in which the strongly correlated attributes $A_1$ and $A_2$ are both stored explicitly, should be replaced with a structure consisting of (i) an empirical law, and (ii) one or two files containing all instances of attribute $A_1$ and the "exceptional" instances of attribute $A_2$, i.e., those instances which are not compatible with the empirical law.

The laws of logic and statistical inference are obvious derivation rule candidates. However, it would not be practical to require from any data base that it should have the full amount of deductive power that logical and statistical laws permit. They have to be combined with rules which limit the number of derivation rule applications to be tried in different practical situations. Such limitations will of course restrict the deductive power of the data base, but on the other hand they will reduce the bad effects of erroneous input to the data base. If the data base has perfect deduction capabilities, the admission of a message, which, however far-fetched it may be, contradicts another message already contained in the data base, would completely destroy the data base as a source of information, as any message would then be derivable. Thus there has to be an appropriate balance between the deductive power of the schema and the purifying power of the filter.

The semantic function of the data base schema consists in conveying the true meaning of the messages contained in the data base to the user. This is certainly a non-trivial task, in which the references and the external (informal) definitions play important roles.

In order to get a comprehensive understanding of the semantic mission in its entirety, we must recall what was said about concepts like "reference person", "reference knowledge", and "compatible frame of reference" in chapter 1, section 5. There are many different persons involved in the creation, maintenance, and use of the information potential of a data
base, and almost certainly all these persons have non-identical frames of reference. Not even the frame of reference of one single person is invariant over time.

We face here a problem area which has caused great discomfort for many people who have seriously thought about the future of data bases. For many kinds of data bases the problems do not seem to be insurmountable in practice, however, provided that they are attacked with imaginativeness and with appropriate, timely attention during the data base design process.

Infological models and the idea of a semantic mission within the schema sub-system of the data base should be of some help in these efforts. A simple example of what could be done, when the problem area is consciously considered, is given by the ARKDABA system, a data base for interactive, real-time production of statistics on individuals and households, being designed at present at the National Central Bureau of Statistics in Sweden. Whenever during his conversation with the data base, the user becomes uncertain about the particulars of the infological model underlying the ARKDABA system, he may immediately call the system's attention to this by pressing a button, and the schema function of the data base then promptly supplies him with the appropriate meta-information, say the external definition or the relevance group(s) of the variable "net income". Similar action is automatically taken by the system, when the user inadvertently reveals that he has not got a sufficient understanding of the infological (or

---

1 For instance, Churchman has taken up the problem in \[23\], chapter 7. He suggests there an interesting "dialectical technique, according to which the data base does not simply provide straight answers to seemingly simple questions. Instead the data base would try to generate "deadly enemy proposals" based upon a picture of the world, a "Weltanschanung", which is contrary to that underlying the user's question. Thus the decision-makers would at least become self-conscious about their implicit assumptions.

2 Cf section 3.2.5.

3 Cf section 2.3.3.
other) assumptions upon which the system is based; then a
tutorial sub-system is invoked. ¹

¹ It seems natural to credit the filter function of the data
base system with the latter kind of action (cf section 4.4.3
below), as it aims at protecting the user from being misled
by the data base.
4.4.2 The nucleus

The nucleus of a database is a set of messages, which are sufficient to generate, in combination with the schema, the information contents of the database. If no message can be removed from the nucleus without changing the information contents of the database, we shall say that the nucleus is infologically minimal, or non-redundant. As has been said before, there may be datalogical as well as infological reasons for allowing the nucleus to be redundant.

Whereas the general idea of the nucleus as a kernel or subset of messages, from which the other messages of the database are derived, seems clearly conceivable even from an infological point of view, we cannot always give a strictly infological justification for considering a particular message as part of the nucleus or not. Several distinct sets of messages may independently of each other fulfil the infological condition, as stated above, for being a non-redundant nucleus, and any set containing one of these sets as a proper subset would be a feasible redundant nucleus. Selecting one of these redundant or non-redundant nucleus candidates as the nucleus of the database is ultimately a design decision where datalogical efficiency considerations inevitably come in. Thus we could say that the exact demarcation of the nucleus of a database is a datalogical problem, which has to be solved under certain infological constraints. More concretely it is the problem of deciding which of the messages should be stored in files, and which of them should be made retrievable from file stored messages by means of programs.

In principle, the database nucleus may contain any kind of messages: complete or incomplete, elementary or consolidated, explicit or implicit (cf chapter 3). In practice, however, many database nuclei will be infologically equivalent to a set of complete, elementary, explicit messages, although for datalogical reasons some of them will be represented by consolidated file entries. As examples of messages which are non-elementary and incomplete in a non-trivial way, we may
take messages containing universal and existential quantifiers, respectively. In practical cases such messages are more likely to occur as empirical laws, etc. (cf 4.4.1), in the schema part of the data base.

Any message in a data base nucleus may be true, false, or meaningless\(^1\). It is the task of the filter to keep the false and meaningless messages out of the data base. However, most data bases will contain such messages, firstly because no filter function can be expected to be perfect, and secondly, and more interestingly, because it would sometimes be an enormous waste of information to throw away messages, even though we know them to be false. Because, as was pointed out in chapter 3, the infological distance between a false message and a true one may be relatively short, as in cases of slight measuring errors, for instance. For many practical purposes such false messages will be as useful as true ones. When the information supply function fails to deliver isolated messages, e.g. because a respondent has filled in his questionnaire incompletely or not at all, it may even be justified to \textit{impute} ("guess intelligently") the missing message on the basis of available, relevant information. It has been said about this technique that the only thing we know for certain about the imputed messages is that they are false. Yet it may be a useful technique, for instance in connection with statistical data bases\(^2\), where isolated deficiencies usually "vanish" in the aggregation process. However, whenever false messages are admitted into the data base it becomes utterly important to keep the quality of the outgoing messages under control. This again is a filter sub-function.

---

\(^1\) Cf chapter 3.

\(^2\) A \textit{statistical data base} is a data base whose output messages are \textit{aggregated} from a lot of messages concerning individual objects (example: table cells); cf chapter 6.
4.4.3 The filter

The filter function of the infological data base should protect the data base and its users against false messages and messages which are not meaningful according to the specifications and definitions embedded in the data base schema\(^1\).

The actions of the data base filter may take several guises. For instance it could make the data base refuse input messages which are not meaningful within the particular infological model underlying the data base, i.e. messages which do not correspond to valid constellations (cf section 2.3.1 and section 3.2.1). Similarly, the filter could monitor the output messages, assembled by the data base, for oddities. Not only could the filter act upon the in- and out-going transactions. Whenever the data base system is idle, i.e. when it does not interact with its environment, it could, through its filter function, try to "purify" itself by carrying out different kinds of consistency checks, thus utilizing the redundancy that may exist in the base, as pointed out earlier.

Beside performing quality control kind of actions, the data base filter could also do things like preventing disclosure of confidential information to unauthorized interactors. In this capacity the filter may in effect make the same physical data base appear infologically different to different users.

**Remark.** A data base with a selective filter, as described in the previous paragraph, may be regarded as a special case of a multi-schema data base, i.e. a data base with several schemas, equivalent to several infological models, which make the same physical data appear as different infological data bases.

---

\(^1\) Remember that the specifications and definitions embedded in the schema imply a particular infological model (section 4.1.1), which in turn implies a classification of all imaginable messages into meaningful and not meaningful messages (section 3.2.1).
Remark. Among the multiple schemas of multi-schema data bases there has to be one schema which is more fundamental than the others, namely the schema which is assumed when the data base assimilates information from its environment. Cf the general impossibility of observing without categories in which to observe i.e. without a model of the observed.
4.5 The dynamic data base

Typically the information contents of a data base are ever changing. In chapter 5 we shall investigate more systematically how different kinds of interactions may affect the data base when it is seen as "the organism" in an S-O-R model (cf section 4.2 above). Here we shall only make few general remarks about the dynamic aspect of the infological data base.

In section 4.4 we defined the infological data base formally as a \(\langle\text{schema, nucleus, filter}\rangle\) triple. Thus by changing the schema, the nucleus, or the filter, we also change the data base. Naturally the typical, most frequent way of changing the data base will be to add or delete a particular message, i.e. to change the nucleus of the data base. In so doing, we change immediately the information contents of the base except in certain redundancy situations\(^1\). By changing the schema we may or may not change the information contents of the data base at once. For instance, the substitution of one definition for another will probably directly affect the set of messages contained in the base, whereas the introduction of a new attribute will not have any effects upon the information contents of the data base, until messages involving references to the new attribute arrive. A filter change, finally, will typically have indirect, mediate effects only. However, e.g. an alteration of the filter's selectivity towards confidential messages\(^2\) may immediately and drastically change the effective data base from a particular user's point of view.

Figure 4 provides an alternative starting point for the discussion of data base dynamics. It shows two different ways of classifying the set of all messages, \(M_\Omega\). According to classification A, \(M_\Omega\) consists of the three subsets

\[
A_1 = \text{the set of false messages, i.e. messages corresponding to valid constellations which are not facts (cf 2.3.1 and 3.2.1)}
\]

---

\(^1\) Cf section 4.4.2.

\(^2\) Cf section 4.4.3
**Figure 4** A cross-classification of the set of all messages ($M_\Omega$)
\[ A_2 = \text{the set of meaningless messages, i.e. messages corresponding to invalid constellations} \]

\[ A_3 = \text{the set of true messages, i.e. messages corresponding to facts} \]

According to the other classification, \( B, M_n \) consists of the subsets

\[ B_1 = \text{the set of unknown messages, i.e. messages which neither are at present in the information contents of the data base under consideration, nor have been so at any earlier point of time} \]

\[ B_2 = \text{the set of known messages, i.e. messages which are at present contained in the data base} \]

\[ B_3 = \text{the set of forgotten messages, i.e. messages which are not at present in the data base, but which have been so at some earlier point of time} \]

The set \( B_2 \) of known messages may be further subdivided into

\[ B_{21} = \text{the set of stored messages, i.e. messages which are contained in the data base nucleus} \]

\[ B_{22} = \text{the set of virtual messages, i.e. messages which are contained in the data base without being stored in the nucleus (cf section 4.4)} \]

The subsets \( B_{21} \) and \( B_{22} \) are exclusive. We may also define

\[ B_{23} = \text{the set of derivable messages, i.e. messages which are derivable from other messages in the data base by means of the derivation rules in the data base schema} \]

Note that \( B_{22} \) is a subset of \( B_{23} \). The data base (nucleus) is non-redundant if and only if \( B_{22} = B_{23} \) (cf section 4.4.2). If
\[ B_{22} = B_{23} = \emptyset, \] the data base has no deductive power, in
which case the schema has only got a semantic sub-mission
(cf section 4.4.1).

It is obvious that classification B is tied to the particular
data base under consideration. Classification A is not data
base independent either, however. We can see this if we analyze
what happens if we change the data base schema, i.e. if we
modify the particular infological model underlying the data
base. First of all such a change may imply a modification of
the set \( M_\Omega \) of all messages; for instance the introduction of
a new attribute or a new object relation will expand \( M_\Omega \).
A modification of the schema may also change the meaningfulness
of a particular message in \( M_\Omega \). Suppose, for instance, that
"person" is an object group and "weight" is an attribute, which
has been defined to be relevant to the "person" object group
only (cf section 2.3.3); then it is meaningless to talk of the
weight of an object which does not belong to the "person"
object group. However, if we modify the infological model and
the schema so that "weight" becomes relevant to another object
group as well, say "car", then messages informing about the
weight of cars suddenly become meaningful; our modification of
the data base schema caused a transition of messages from
subset \( A_2 \) to subsets \( A_1 \) and \( A_3 \).

**Remark.** For data bases which are based upon identical info-
logical models, and thus have identical schemas, the \( M_\Omega \) -sets
are identical, and so are the subsets according to classification
A, because (i) the common schema uniquely determines a
common subset \( A_2 \) of meaningless messages, and (ii) according
to the objectivity postulate (chapter 1) the same meaningful
message has to belong either to the subset \( A_1 \) of false messages
in both data bases or to the subset \( A_3 \) of true messages in
both bases. However, the classification B need not coincide
for two data bases, even if they are based upon the same
infological model and have a common schema, because one of
the bases may have got to know (or forgotten) a lot of messages
which the other data base has not learnt (forgotten) yet.
The remark led us to the subject of classification B. Obviously the assimilation by the data base of an unknown message implies a \( B_1 \rightarrow B_2 \) transition. If the message was already known by the data base, the addition transaction might lead to nothing or to a transition from \( B_{22} \) to \( B_{21} \cap B_{23} \); in the latter case the message is redundantly stored although it is derivable. Similarly a deletion transaction might lead to one of the transitions \( B_2 \rightarrow B_3 \), or \( B_{21} \cap B_{23} \rightarrow B_{22} \).

The information contents \( M(t) \) of a data base \( DB(t) \) at a particular point of time \( t \) can easily be identified as the subset \( B_2 \) of known messages.

From figure 4 we find immediately that

\[
M(t) = B_2 = (A_1 \cap B_2) \cup (A_2 \cap B_2) \cup (A_3 \cap B_2)
\]

**Remark.** The mission of the data base filter may be expressed as

(a) keeping down the number of messages in \( (A_1 \cap B_2) \);
   "consistency checking"

(b) keeping down the number of messages in \( (A_2 \cap B_2) \);
   "syntax checking"

(c) preventing confidential messages in \( (A_3 \cap B_2) \) from being disclosed to unauthorized users; "secrecy checking"

(d) preventing \( (A_3 \cap B_2) \rightarrow (A_3 \cap B_3) \) transitions, unless they are desired by the data base owner; "security checking"
5. Data base interaction

5.1 Introduction

As was noted in the previous chapter, the dynamic aspect of the data base is of utmost importance. The object system changes, and so do the contents of the data base. In fact, it is the ability of the data base to deliver relevant, accurate, and timely information about states and changes of states in the real world, which is the principal reason for its existence. It is obvious that this ability presupposes an extensive exchange of information, an extensive communication, between the data base and its environment. In this chapter we shall delve a little deeper into the nature of the different kinds of interactions which take place between the main data base system and the surrounding systems with which the data base communicates more or less incessantly during its life time.

In order to systematize the data base interaction problem area we shall use two points of departure, which were introduced in the previous chapter.

Firstly, in chapter 4, figure 2, we broke down the data base environment into eight subsystems. This subsystem structure is taken as a basis for a classification of interactions and interactors to be discussed in section 5.2.1 below.

Secondly, in section 5.2.2, we shall take up the formal definitions of closed and open interactions, stated in section 4.2, and then, in section 5.2.3, we shall use the so-called S-0-R model as another basis for classifying interactions and interactors.

Interaction between the data base and its environment takes place by means of data base transactions. In section 5.3 we shall analyze the formal structure of transactions and discuss a few important transaction types.

Finally, in section 5.4, we shall use the infological framework developed in chapters 2 and 3 in order to analyze and
graphically depict the structure of a typical retrieval query. In that connection we shall also touch on the problem of designing languages for data base interaction.
5.2 The interaction process

5.2.1 A classification of interactions based upon the subsystem structure of the data base environment

In chapter 4, figure 2, we identified eight environment subsystems surrounding the infological data base. Five of these systems, or functions, were found to be heavily and directly engaged in data and information exchange with the main data base system, namely

- the information supply function, covering all processes, persons, equipment, etc, engaged in observing particular phenomena in the object system, coding (classifying) the observations in terms of the particular infological model underlying the data base, representing the coded observations by data messages, transforming and transmitting the data until they finally reach the data base

- the information consumption function, covering all processes, persons, equipment, etc, engaged in identifying, operatively, information needs in the decision processes for which the data base is meant, formulating the information requests in terms of the particular infological model underlying the data base, submitting the requests to the data base, receiving, interpreting, and analyzing the answers from the data base, and influencing upon and making decisions concerning particular aspects of the object system

- the data base administration function, embracing all planning, systemeering, programming, operation, and supervision activities in connection with the design, construction, and running of the data base

- the computer function, covering the technological devices allocated to the data base, and the people, processes, etc, engaged in managing and operating these devices

- other data bases than the one under consideration
We shall say that each of these five environment subsystems defines an interactor category. In this chapter we shall focus our interest upon the first three functions of those listed above. The corresponding interactor categories will be called

- **information suppliers,**

- **information consumers,** or users, and

- **data base administrators**

As suggested by the list stated earlier, each of the three interactor functions may be further subdivided. Accordingly there are sub-categories of interactors under each of the three headings above. Many of these sub-categories, like coders and programmers may inherit names from closely related functions in pre-data base information processing. For other sub-categories, corresponding to more or less new functions there are no appropriate labels today. We shall feel free to introduce names for these functions and categories whenever needed. For instance, it is important to be able to talk about two sub-categories of data base administrators which we shall call **information structurers** and **data structurers,** respectively. The two functions are clearly distinguishable and care should be taken to avoid conceptual confusion.¹

The tasks of the **information structurers** are related to the infological model underlying the data base, which means that they are related to the schema and filter functions of the data base. The information structurers should be concerned with the practicality and consistency of the operative infological model, and whenever desirable they should undertake changes like the specification of new object groups, properties, attributes, object relations, definitions, and names. They could also be responsible for the appropriate

¹Particularly in the case of a small data base the two functions may be exercised by the same people. There are the same arguments for and against double assignments in data base administration as in any other kind of organizational activity.
distribution of information about what information is accessible from the data base, for activating potential information consumers not yet heard of, and so on.

The data structurers, on the other hand, should be concerned with tasks related to the internal, datalogical structuring of the data base. Whenever the infological or technological environment changes, the data structurer should be prepared to such things as altering the demarcation between the nucleus and the virtual part of the data base\(^1\), restructuring the files, and redesigning the strategies for efficient processing of transactions.

In parallel with the three main interactor categories we could also talk about three major types of data base interactions, namely

- **supplier interactions**,  
- **consumer interactions**, and  
- **administrator interactions**,  

each of which could be further sub-divided, following the sub-classification of the corresponding interactor categories. For instance, we can distinguish between information structuring and data structuring interactions.

\(^1\) Cf sections 4.4.2 and 4.5.
5.2.2 Open and closed interactions

In this section we shall start out from the formal definitions of open and closed interactions stated in 4.2. As was also stated in chapter 4, these definitions are related to the so-called S-O-R model, used by psychologists.

Thus we recall that a closed interaction was defined as a septuple

(a) \( \langle S, O, O', R, E, E', S' \rangle \)

where

- \( S \) is a stimulus, an input transaction from the environment to the data base

- \( O \) is the organism, the data base, acted upon and transformed into \( O' \) by the stimulus

- \( R \) is the resulting response, an output transaction from the data base to the environment

- \( E \) is the environment, or, if we like, an environment subsystem, or an individual interactor, acted upon and transformed into \( E' \) by the response

- \( S' \) is the next input transaction from the environment (subsystem), or the interactor, to the data base.

The last component, \( S' \), of one closed interaction coincides with the first component of the next one. Thus, as is illustrated by figure 1, we may in principle trace the complete history of a data base, including all internal \( (O \rightarrow O') \) and external \( (E \rightarrow E') \) transformations by following the chain of closed interactions; if we let \( E \) denote a subsystem or an individual interactor instead of the whole data base environment, we get the partial history with respect to that subsystem or interactor.
Figure 1  The interaction history of a data base
We also recall from chapter 4 that an open interaction was defined as the sub-quadruple

\[(b) \quad \langle S, O, O', R \rangle \]

of a closed interaction septuple.

The closed interaction concept seems to be theoretically important, because it focuses attention upon certain mutuality and symmetry aspects of the data base interaction process; during one and the same closed interaction the interactor and the data base are affected by and affect each other. In many data base projects it may be as important to consider the environment oriented complementary open interaction

\[(c) \quad \langle R, E, E', S' \rangle \]

as the data base oriented open interaction \(b\). The "complementary aspect" of the data base interaction process raises questions like

- how much easier is it for a user (information consumer) to interact with a data base in terms of an infological model than in terms of traditional, data- and hardware-oriented concepts like files, blocks, records, fields, and positions?\(^1\)

- what degree of conversation should be aimed at when one designs data base interaction languages for different interactor category; i.e., where is, for a particular kind of interactor and task, the break-even point between (i) an interaction chain consisting of an enormous lot of simple but

---

\(^1\) Admittedly, at the present stage there is no empirical evidence in support of the hypothesis that infologically oriented data base interaction languages would at all be superior, from the user's point of view, to the languages in use today. This hypothesis seems plausible, however, but it seems equally plausible that languages treating the data base like a "black box reservoir of information" (cf chapter 4) will cause the data base oriented part of the interaction process, \( \langle S, O, O', R \rangle \), to become more costly, and then the question arises, how much more efficient the complementary open interaction will be, if the interaction language is based upon the infological framework developed in earlier chapters of this report.
tedious interactions, and (ii) an interaction chain consisting of one or two very complex and error-prone interaction

- In what situations and to what degree should a data base interaction language be non-procedural (descriptive, result-oriented), and when is a more procedural language more efficient from the interactors' point of view?

- How often and in what doses should the data base automatically, during a chain of closed interactions, provide the information consumer with "information on information", like definitions and quality declarations, in order as far as possible to harmonize dynamically the user's frame of reference with that of the data base, so that the user and the data base assign the same meaning to the messages they exchange.

These and similar questions define extremely important, though at present on the whole unexplored research areas, which call for joint efforts by behavioral scientists, data base specialists, linguists, and others. Among other things such projects, if experimentally oriented, would shed empirical light upon some of the main hypotheses of this report.
5.2.3 A classification of interactions based upon the S-O-R model.

In this section we use the S-O-R model as a structuring tool in our efforts to systematize data base interactions. In a particular interaction

(S) a stimulus from the environment may or may not occur

(O) the organism, i.e. the data base, may or may not be transformed

(R) a response from the data base to the environment may or may not appear

As a situation characterized by a "null stimulus", \( S = \varnothing \), a "null transformation", \( O \to \varnothing \), and a "null response", \( R = \varnothing \), cannot very well be regarded as interactive, this leaves us with \( 2^3 - 1 = 7 \) basic interaction types. This is a classification scheme which is meaningful for both open and closed interactions; it is not meaningful for complementary open interactions, though, because it refers only to the S-, O-, O'-, and R-components, and not to the E-, E', and S'-components of the closed interaction septuple \( \langle S, O, O', R, E, E', S' \rangle \).\(^1\)

The seven basic interaction types may be grouped into several structures, one of which is displayed by figure 2.

The structure of figure 2 has two basic sub-structures, labeled "spontaneous behavior", characterized by the null stimulus, and "triggered behavior".

Spontaneous behavior may occur if we have designed the data base to take own initiatives. Such design could be very practical for missions, which have to be carried out now and then, but which have low priority at any particular point of time. There are a lot of maintenance tasks which fit this description. For

\(^1\) Cf section 5.2.2, where open, closed, and complementary open interactions are defined and discussed.
<table>
<thead>
<tr>
<th>S - O - R</th>
<th>INTERACTION TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Φ - Φ - Φ</td>
<td>Spontaneous behavior</td>
</tr>
<tr>
<td>Φ - 1 - Φ</td>
<td>modified data base</td>
</tr>
<tr>
<td>Φ - Φ - x</td>
<td>invariant data base</td>
</tr>
<tr>
<td>1 - Φ - x</td>
<td>Triggered behavior</td>
</tr>
<tr>
<td>1 - 1 - x</td>
<td>modified data base</td>
</tr>
<tr>
<td></td>
<td>- nucleus modification</td>
</tr>
<tr>
<td></td>
<td>- addition of message</td>
</tr>
<tr>
<td></td>
<td>- deletion of message</td>
</tr>
<tr>
<td></td>
<td>- substitution</td>
</tr>
<tr>
<td></td>
<td>- schema modification</td>
</tr>
<tr>
<td></td>
<td>- filter modification</td>
</tr>
<tr>
<td>1 - Φ - x</td>
<td>invariant data base</td>
</tr>
<tr>
<td></td>
<td>- query</td>
</tr>
<tr>
<td></td>
<td>- about the object system proper (^1)</td>
</tr>
<tr>
<td></td>
<td>- about the information (^1)</td>
</tr>
<tr>
<td></td>
<td>- modification failure</td>
</tr>
</tbody>
</table>

**Figure 2**  A classification of interactions based upon the S - O - R model

**Legend:**

"Φ" = null stimulus (transformation, response)

"1" = other than null stimulus (transformation, response)

"x" = any kind of stimulus (transformation, response)

\(^1\) Queries about the information request meta-information (cf 3.2.5), whereas queries about the object system proper request information about the observed object system (cf 4.2).
instance, in connection with any data base system there has to be a manual or automatic function, which collects information about different aspects of data base performance, and which now and then suggests more or less radical datalogical or infological restructurings of the data base; the restructuring may concern the nucleus, the schema, or the filter, or several of these subsystems. The spontaneous behavior may include the data base modification \((0 \rightarrow 0')\), or it may leave the data base invariant (null transformation) and only result in an output transaction, which may recommend some environment subsystem, for instance a particular subfunction of the data base administrator, to initiate some kind of triggered behavior, for instance a partial redesign of the file system.

If we, despite what has been said, should try to identify a particular, data base external originator of spontaneous behavior, the only candidate seems to be the designer of the data base system, whereas the particular, data base external originator of triggered behavior is of course the triggering stimulus from an operative process in an environment subsystem.

We distinguish between triggered behavior which modifies the data base and triggered behavior which leaves the data base invariant. Data base modifying behavior is initiated by a modification transaction; different kinds of modifications are suggested by some of the sub-headings in figure 2 and will be further discussed in section 5.3.1. The most typical interaction type which leaves the data base invariant is the query, which is initiated by a query transaction and will be further discussed in section 5.3.2. However, interactions which aim at data base modification but which are not successful, for instance because the interactor is not authorized to carry out the desired modification, formally belong to the same interaction category as queries.
5.3 Data base transactions

It is by means of data base transactions that the exchange of data, information, and control between the data base and its environment takes place; without data base transactions there would be no data base interaction. Formally a data base transaction has the structure

- $\text{dbt} = \langle \text{operator}, \text{parameter} \rangle$

where

- the operator is a stimulus which initiates a certain kind of behavior, or action, on part of the data base or the environment interactor, depending upon the direction of the transaction

- the parameter modifies the behavior of the data base or interactor by providing the process, initiated by the operator, with certain input

If the transaction is directed from the environment to the data base, i.e. if it is an input transaction, the operator could for instance initiate the addition or the deletion of a message; the operator would then put the data base into addition or deletion "mode", whereas the actual piece of information to be added or deleted would be supplied by the parameter part of the transaction.

If the transaction is instead directed from the data base to an interactor, i.e. if it is an output transaction, the operator could for instance put the interactor into receiving or replying mode, whereas the parameter would in those cases be the information to be received or the question to be replied to, respectively.

Naturally, the operator need not be represented by a specific string of characters or the like; just as often the stimulus may be inseparable from the string of characters representing
the parameter. For instance, a query operator may be explicitly represented by a question mark ("?") or implicitly represented by the grammatical structure of the message supplying the parametrical input to the query process.  

It seems natural to say that database transactions with identical operator parts belong to the same transaction type. It also seems natural to claim that there should be at least one transaction type for each interaction type. However, as we saw in section 5.2, there are several ways of classifying interactions, and for each way the classification may be "deep" or "shallow". Ultimately it is a design problem to define the set of transaction types (operators) for a particular database system. The problem is structured by the classifications given in 5.2.1 and 5.2.3. For instance, we may ask ourselves at goal-setting and design time what operators (if any) we should define for a particular environment subsystem, say information supply and a particular row in figure 2, say "query about the information".  

The number and scope of the operators we define for a specific task will also depend upon whether we choose to achieve a certain total amount of "communication variety" by means of a few operators allowing a wide range of parametrical input or by means of a larger number of operators, each of them with more rigid requirements upon the parametrical input.

---  

1 As we shall see in section 5.4.2, the parametrical input to a query process often consists of an incomplete message (cf also section 3.2.2).  

2 This particular combination might for instance make the designer recognize that the database system he is designing should embrace language, meta-information, and retrieval routines enabling a particular supplier category to ask questions about the information they should supply, e.g. how a particular variable is defined. Similar questions may of course be asked by several interactant categories, but probably each category will have its specific requirements as to the language, for instance.
5.3.1 Modification transactions

If we look back at figure 2, we find that there are several kinds of data base behavior which may be triggered by a modification transaction. First of all the modification may concern either the nucleus, or the schema, or the filter function of the data base. In this section we are particularly interested in nucleus modifications. Then we may distinguish between additions, deletions, and substitutions of information.

An information addition transaction has the structure

- \(<\text{ADD}, \text{message}\>"

where

- \(\text{ADD}\) is an operator which puts the data base into "information reception mode", and

- "message" denotes the particular message to be received by the data base

The message parameter of the information addition transaction may be of any kind and complexity. Let us assume, however, that it is equivalent to a set

\[
M = \{m_1, \ldots, m_n\}
\]

of \(n\) elementary messages (cf 3.2). Then it should be observed that this does not necessarily mean that the transaction adds exactly \(n\) e-messages to the information contents of the data base. On the one hand some of the e-messages in \(M\), say

\[
M_i = \{m_{i1}, \ldots, m_{in}\}
\]

may already be known to the data base as stored or derivable messages (cf chapter 4, figure 4). On the other hand, other messages than those contained in either \(M\) or
(3) \( B = \) the information contents of the data base

may be derivable from \( M \cup B \) and thus truly added to the data base by the transaction. Let

(4) \( AB = AB (B, M) \)

denote the set of e-messages, which due to the transaction, are transitioned from the set of unknown messages to the set of known messages\(^1\). \( AB \) contains as subsets both the set of messages in the transaction parameter which are really new to the data base,

(5) \( M_{i+1}, n = M \setminus M_i \)

and the set of messages which are derivable from \( M \cup B \) but not from \( B \) alone.

The cardinality of \( AB \),

(6) \( \# (AB) \)

could be taken as a measure of the quantity of information added to the data base by the transaction. Note that \( \# (AB) \) may be less than, equal to, or greater than \( n \). If \( \# (AB) = 0 \) the transaction adds no new knowledge at all to the data base.

Remark. The definition of information quantity suggested above implies among other things that the same transaction may add different amounts of information to different data bases. This seems perfectly reasonable, particularly if we recall that "information" was defined in chapter 1 as "new knowledge". What is "new knowledge" may certainly differ from one data base to another, be the data base "natural", i.e. a human mind, or "artificial". The argument also suggests that the stated definition of information quantity could be just as useful for communication between people as for communication with a data base. The magnitude of information conveyed by a certain message must be a function of the subjective frame of reference

---

\(^1\) Cf chapter 4, figure 4.
of the receiver as well as of the (objective) message itself.

An information deletion transaction has the structure

- \(<\text{DELETE}, \text{message}>\)

where

- DELETE is an operator which initiates an abolition process within the database "organism", and

- "message" denotes the particular message to be abolished from the database

We assume again that the message parameter is equivalent to a set \(M = \{m_1, \ldots, m_n\}\) of \(n\) elementary messages. The deletion transaction may imply different kinds of actions depending on the status of each e-message \(m_i\) \((i = 1, \ldots, n)\) within the information contents, \(B\), of the database:

(a) If \(m_i\) is not known to the database, \(m_i \notin B\), no action at all is needed. However, in this situation the deletion transaction from the interactor to the database would probably be followed by a "warning transaction" from the database to the interactor.

(b) If \(m_i\) is a stored, non-derivable message, \(m_i\) should be cancelled, which may cause both datalogical and infological problems. Firstly, if \(m_i\) is redundantly stored in several files, it has to be removed from all the files where it occurs. Secondly, if \(m_i\) is cancelled, all virtual messages in \(B\) which are not derivable from \((B \setminus \{ m_i \})\) will be automatically cancelled as well. If this was not the interactor's intention, the virtual messages concerned have to be stored explicitly before \(m_i\) is cancelled. Thirdly, by a deletion (like by an addition) the information and data contents of the database are changed. This implies that a lot of meta-information, "information on information" and "information on data" has to

---

1 Recall figure 4 of chapter 4 in the subsequent discussion.
2 Cf section 4.5.
be changed, too, if it is to remain correct.

(c) The worst situation appears if \( m_1 \) is a derivable message. Then either (i) we have to restructure the data base so that \( m_1 \) becomes a stored, non-derivable message, which may be cancelled as described under (b), or (ii) the data base has to have the "exceptional storage feature"\(^1\), i.e. the nucleus of the data base has to have a part in which messages are stored, which in cases of contradiction should have priority over the rest of the information contents of the data base; thus the "exceptional messages" should nullify their negations, even if the latter should happen to be derivable from other stored messages by means of the "ordinary"\(^2\) set of derivation rules of the data base schema.

If we let \( \Delta B = \Delta B (B, M) \) denote the set of e-messages, which due to the deletion transaction, are transitioned from the set of known messages to the set of forgotten messages\(^3\), we may again take \( \# (\Delta B) \) as a quantitative measure of the change in the information contents of the data base, caused by the transaction; this time the change is negative, of course. In connection with deletions, \( \# (\Delta B) \) will usually be greater than \( n \); it will be less than \( n \) if and only if at least one e-message in \( M \) is not known to the data base, case (a) above.

An information substitution transaction has the structure

\[
\langle \text{SUBSTITUTE}, \text{message 1}, \text{message 2} \rangle
\]

where

- \text{SUBSTITUTE} is an operator, and

- "message 1" denotes the message to be replaced with

- "message 2"

---

\(^1\) Cf. the discussion in section 4.4.1 about how strongly correlated attributes may be stored.

\(^2\) Formally the "nullification rule" may be regarded as an extra-ordinary, high-priority derivation rule, which is also part of the data base.

\(^3\) Cf. chapter 4, figure 4.
Formally a substitution is nothing but a deletion immediately followed by an addition. In practice, however, the two messages, the deleted message and the added one, are always related to each other in some characteristic way. For instance "message 1" may be a message, which has been erroneously added to the data base, and "message 2" may be the correct message, which should have been added in the first place.

Another typical example of information substitution is the traditional update situation, where a message like

- \(<\text{object } i, \langle \text{attribute } A, \text{value } j \rangle, \text{time } t_j \rangle\)

is replaced with

- \(<\text{object } i, \langle \text{attribute } A, \text{value } k \rangle, \text{time } t_k \rangle\)

where \(t_j < t_k\) = "current time". Such transactions are very frequent in connection with situational data bases, which aim at reflecting the "current situation" in the object system and nothing more. Even in historical data bases, in which the history of the object system is also reflected, update transactions may be common, however, namely if only a partial history of the object system is saved; it is then natural to define the partial history in terms like "the latest \(n\) time versions of message type ...", and then the addition of time version \(x\) becomes tied to the deletion of time version \(x-n\).
5.3.2 Query transactions

According to figure 2 the query type of interaction is a kind of triggered behavior which leaves the data base invariant. The triggering stimulus is contained in a query transaction

- $\langle \text{operator, parameter} \rangle$

which normally results in a non-trivial reply transaction from the data base to the interactor.

There are several kinds of query transactions. First of all, we may distinguish between queries about the object system proper\(^1\) and queries about the data base contained information about the object system proper\(^2\). However, if the meta-data bases containing the meta-information are designed according to the same infological principles as the data base proper - and there is no reason why they should not be - then there is no structural difference between the two kinds of queries.

If we are looking for structural differences between different kinds of queries, we may instead distinguish between

(a) yes/no-queries,

(b) retrieval queries, and

(c) process queries

What is typical of yes/no-queries is that the parameter part of such a transaction will always be a complete message\(^3\), $m$. If $m$ is contained in the data base, the obvious reply to the query is "yes", and if a message, which logically contradicts $m$, is contained in the data base, the latter will certainly reply "no" to the query. If neither $m$, nor its contradiction is contained in the data base some kind of qualified "I don't

---

\(^1\) Cf section 4.2
\(^2\) Cf section 3.2.5
\(^3\) Cf sections 3.2.1 and 3.2.2
know" response should be expected; what exactly will happen in such a situation is dependent upon our infological design of the data base schema (cf 4.4.1); a sophisticated data base might give an answer in terms of probabilities, for instance.

In retrieval queries the parameter will be an incomplete message, m. The reply is expected to contain messages, referring to all e-constellations, which satisfy m, i.e. all e-constellations contained in the target set of m. Naturally, the "I don't know" problem mentioned above is present in connection with retrieval queries, too, and once again the action taken by the data base in such situations will depend upon our infological design of the data base schema.

In process queries the operator or the parameter part will contain a processing request, meaning that not only should a specified set of messages be retrieved, they should also be processed in a certain way before presentation. For instance, the processing request could imply aggregation and statistical analysis. We shall return to the subject of process queries in chapter 6.

Let us return to the retrieval queries for some further considerations. They are particularly important, because in fact most transactions from an interactor to a data base will, among other things, imply processing that is equivalent to the processing of a retrieval query.

Many retrieval queries conform to the pattern

\[(1) \quad \text{"For all objects having the property } P, \text{ retrieve the values of the attributes } A_1, \ldots, A_m \text{ at the times } t_1, \ldots, t_m, \text{ respectively."} \]

---

1 For definitions of "incomplete message", "satisfy", and "target set", see section 3.2.2.

2 As was stated in the introduction to 5.3, we have a choice between specifying a few, very flexible operators or a larger number of operators with more rigid requirements upon the parametrical input.

3 In this connection it may be practical to consider "yes/no-queries" a special case of "retrieval queries". Cf section 3.2.2, "special case".
The parameter part of the retrieval query transaction (1) contains the \( m \) incomplete \( e \)-messages

\[
(2) \quad \langle P, \langle A_i, ?, t_i \rangle \rangle \quad (i = 1, \ldots, m)
\]

where \( P \) is an ambiguous object reference, and \( \langle A_i, ? \rangle \) is an ambiguous property reference\(^1\). Note that \( P \) may in fact be a very complex property involving a lot of \( \langle \text{attribute, value} \rangle \) -pairs, object relations, times, etc\(^2\). When \( P \) involves attributes, it may be difficult to distinguish between those attributes and the attributes \( A_1, \ldots, A_m \), the values of which are requested by the query transaction. As the distinction between the two, types of attributes may be essential, for instance in design discussions, we have found the following definition to be convenient.

**Definition.** Retrieval queries conforming to pattern (1) consists of two parts, one involving the property \( P \), and another involving the attributes \( A_1, \ldots, A_m \) and the times \( t_1, \ldots, t_m \). The former part will be called the \( \alpha \)-part of the query, any attributes involved will be called the \( \alpha \)-attributes, and so on. The latter part of the query will be called the \( \beta \)-part, the attributes involved will be called the \( \beta \)-attributes, etc.

Note that the classification of attributes into \( \alpha \)-attributes and \( \beta \)-attributes is only relevant in connection with a particular query. It is a functional, context-dependent classification, and one and the same attribute may be an \( \alpha \)-attribute in one query and a \( \beta \)-attribute in another.

**Example.** Compare the queries

\[
(3) \quad \text{"List name, address, occupation, and telephone number of all single males aged 25 and earning more than \$10000".}
\]

\(^1\) Cf sections 3.1.3 and 3.2.2.

\(^2\) Thus \( P \) may be a generated property. Cf section 2.2.2, examples 1-5.
(4) "List income, age, and registration number of all dentists whose names begin with a C",

In query (3) the $\alpha$-attributes are "marital status", "sex", "age", and "income", and the $\beta$-attributes are "name", "address", "occupation", and "telephone number". In query (4) the $\alpha$-attributes are "occupation" and "name", and the $\beta$-attributes are "income", "age", and "registration number". Thus, "income" and "age" are $\alpha$-attributes in (3) but $\beta$-attributes in (4), whereas "occupation" and "name" are $\beta$-attributes in (3) but $\alpha$-attributes in (4).

Remark. All times, both $\alpha$- and $\beta$-times, involved in (3) and (4) were tacitly assumed to be identical and equal to "current time", or "now". In general, a lot of different $\alpha$- and $\beta$-times may of course occur in a retrieval query conforming to pattern (1).

A more general retrieval query pattern than (1) is

(5) "For all object tuples $< o_1, \ldots, o_n >$ where the n-ary object relation $R$ holds between $o_1, \ldots, o_n$, retrieve the values $v_{ij}$ of the attributes $A_i$ ($i = 1, \ldots, n$) for the objects $o_j$ ($j = 1, \ldots, n$) at the times $t_{ij}$ respectively.

The reply to such a query would contain an $m \times n$ matrix of values. Naturally $R$ may be a generated object relation.¹

Example. In the query

(6) "Retrieve civic registration number, age, and income for all married couples where the husband's father is a fisherman and the wife's mother was born in France".

$R$ is generated from the object relation "married to" and the properties $p_i$: "to have a father who is a fisherman"

¹ Cf section 2.2.3, examples 6-10.
and $p_2$: "to have a mother who was born in France", where $p_1$ and $p_2$ in turn involve the object relations "father" and "mother" and the attributes "occupation" and "nationality". All the object relations, attributes, and properties mentioned so far belong to the $\alpha$-part of the query. The $\beta$-attributes of (6) are "civic registration number", "age", and "income".
5.4 Communication and illustration techniques

The infological framework presented in this report may be used for several purposes. For instance,

(a) it may serve as the conceptual basis of a general theory of data bases

(b) it may serve as a "common denominator" to all parties who are interested in the design of a particular data base

(c) it may serve as the basic structure of data base interaction languages.

In order to be useful for all these purposes, the infological framework must, of course, be theoretically sound. This is not sufficient, however. For purposes (b) and (c) above it is necessary and for purpose (a) it is clearly advantageous that we are able to illustrate and communicate infological structures by means of simple graphical and verbal techniques. In section 5.4.1 we shall use a retrieval query as an example of what kind of techniques may be developed. The subject will be further discussed in chapter 6. In section 5.4.2 we discuss more generally what kind of development may be expected to occur in the field of data base interaction languages over the years to come.
5.4.1 An example

Consider the retrieval query

(1) "What are the addresses of the textile-manufacturing enterprises with more than 15 employees and producing articles containing the poison with the industrial name POI?"

In order to be able to analyze the infological structure of this query, we must assume a particular infological model.
Let us assume that the following specifications have been made:

"Enterprises", "articles", and "poisons" are object types (cf 2.3.2). "Textile-manufacturing" is a property, which generates a subgroup of "enterprises". "Number of employees" is a single-valued attribute, i.e., a variable (cf 2.3.3), which is relevant to enterprises. "POI" is the value of an identifying attribute within the object type "poisons" (cf 2.3.3). "address" is an attribute, which is relevant to enterprises. "Produce" is an object relation with the relevance groups (i) "enterprises" and (ii) "articles" (cf 2.3.4). "Contain" is similarly an object relation between "articles" and "poisons".

Now let us try to fit query (1) above to the general retrieval query pattern (1) of section 5.3.2, i.e., to the pattern

(2) "For all objects having the property $P^a$, retrieve the values of the attributes $A_1^b, \ldots, A_m^b$ at the times $t_1^b, \ldots, t_m^b$, respectively."

We recall that the property $P^a$, may be generated from a set of so-called $a$-attributes, $a$-times, and $a$-relations.

Firstly, we observe that there are no explicit time references whatsoever in query (1). From the infological context we may assume, however, that all needed\(^1\) time references in (1) should

\(^1\) Note that in order to be meaningful, a query conforming to the pattern (2) must have an $a$-property, $P^a$, which is stable, which it may be, for instance, by being time-dependent (cf 2.2.2, example 4, and 2.2.4, example).
be tacitly understood to be "current time", "now".

The $P^a$-property of (1) is seen to be

(3) $P^a =$ "textile-manufacturing enterprise with more than 15 employees and producing articles containing the poison POI",

a property which is generated from the atomic\(^1\) properties

(4) $P_1^a =$ "to be textile-manufacturing",
$P_2^a =$ "to be an enterprise",
$P_3^a =$ "to be an article",
$P_4^a =$ "to be a poison",

and the complex\(^1\) properties

(5) $P_5^a =$ "no of employees $> 15$",
$P_6^a =$ "industrial name = POI",

involving the attributes

(6) $A_1^a =$ "no of employees", and
$A_2^a =$ "industrial name",

respectively. Among the entities generating $P^a$ are also the object relations

(7) $P_1^a =$ "produce", and
$P_2^a =$ "contain",

and the tacitly understood time

(8) $t^a =$ "now"

The infological precedents of the $P^a$-property of query (1) are indicated by the formula

\(^{1}\) By an atomic property we mean a property which is referred to by an atomic property reference (cf 3.1.4). Similarly, a complex property is a property which is referred to by a complex property reference (cf 3.1.4).
(9) \[ F^\alpha = g(P_1^\alpha, P_2^\alpha, P_3^\alpha, P_4^\alpha, A_1^\alpha, A_2^\alpha, R_1^\alpha, R_2^\alpha, t^\alpha) \]

However, this formula does not tell the structure of the generation expression \( g \), which contains references to generation rules like those stated in chapter 2. This instead could be done for instance by means of the predicate-logical formula

\[
F^\alpha x \iff (P_1^\alpha x \land P_2^\alpha x \land P_3^\alpha x \land \exists y \exists z \ (R_1^\alpha xy \land P_3^\alpha y \land R_2^\alpha yz \land P_4^\alpha x \land P_6^\alpha z))
\]

where \( x \) could be thought of as an arbitrary member of the target set of the \( \alpha \)-part of query (1), i.e. as an arbitrary member of the set of objects referred to by the property reference "\( F^\alpha \)" in its infological context (1).\(^1\)

So much for the \( \alpha \)-part of query (1). The \( \beta \)-part is much easier to analyze, since it only involves the one\(^2\) attribute

\[
(11) \quad A^\beta = "\text{address}"
\]

and the tacitly understood time

\[
(12) \quad t^\beta = "\text{now}"
\]

Our analysis shows that query (1) calls for the values of attribute \( A^\beta \) at time \( t^\beta = "\text{now}" \) for all objects \( x \) having, at time \( t^\alpha = "\text{now}" \), the property defined by (10). However, we are well aware of the fact that the formalism used to express the results of our analysis are not satisfactory from a communication point of view. The predicate-logical formula (10), for instance, has the following severe drawbacks:

- it is inconceivable to people who are not well acquainted with predicate calculus

\(^1\) Cf 3.1.4, 3.2.2, and 5.3.2.

\(^2\) Thus \( m = 1 \) if we compare query (1) to the pattern (2).
- it is not easy to survey even for those who are familiar with predicate calculus

- it would have become still more complex, had the a-part of query (1) contained times other than "now"; as "time" is not a fundamental concept of predicate calculus, it is not even possible to tell exactly how such times should have been expressed in (10)

In figure 3 we show a graphical representation of query (1), which seems easier to conceive at a glance than a predicate-logical expression, and which should thus be more apt for illustration and communication purposes.

The squares of figure 3 represent sets of objects, object targets, which are demarcated by the properties symbolized by the lines entering the squares from different directions. Vertical lines entering the squares denote relation-dependent properties (cf 2.2.2, example 5), i.e. properties which are generated from object relations. A horizontal line entering an object target square from the right denotes a property, which is also common to all the objects in the target set, but which is not relation-dependent. A horizontal line entering an object target square from the left denotes a property asked for by the query, a requested value of a particular attribute; naturally this property need not be the same for all objects in the target set.

The triangles in figure 3 denote properties. By means of generation rules, represented in the figure by circles with operator symbols in them, the "triangle properties" may be combined into derived properties, which in turn may generate other derived properties.

It must be admitted that the illustration technique demonstrated in figure 3 has its limitations. For instance, it is not obvious how object relations of higher degree than 2 should be represented. On the other hand the technique does not seem to be subject to the earlier stated drawbacks of the predicate-calculus formalism, at least not to the same
address (now) = ?

- being an enterprise (now)
  - being textile-manufacturing (now)
    - number of employees (now) > 15

- being an article (now)
  - being a poison (now)
    - industrial name (now) = POI

Figure 3
extent. With only little training it should be possible for all, who are interested in the design of a data base, as well as for all, who will directly interact with the data base at operation time, to understand and even to draw themselves diagrams like figure 3. We can also see that different sets of α- and β-times in the query to be represented will not cause any problems, if this technique is used.

As a matter of fact we may in many practical situations make the infological structure of a query still easier to survey by adopting a few simplifying conventions. These conventions are:

(a) the default time reference "now" is left out

(b) atomic type and group properties are represented directly in the squares rather than by means of separate triangles linked to the squares by straight lines

(c) logical conjunction is regarded as the default property generation rule, and the corresponding circles are left out

Applying these conventions to figure 3, we get figure 4, which is obviously much "cleaner".

In many practical situations graphical communication is not economically feasible. We then have to consider the problem of representing infological structures verbally instead of graphically. Natural language is one alternative which will be commented upon in the next section. The formal notation of predicate calculus is another possibility which was discussed earlier in this section. An alternative, which is believed to be more realistic, is shown in figure 5, which is essentially a "verbal reflection" of figures 3 and 4. Note, however, that we have had to introduce the auxiliary symbols A, B, and C.
Figure 4
<table>
<thead>
<tr>
<th>OBJECT GROUP</th>
<th>PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>enterprise</td>
</tr>
<tr>
<td></td>
<td>textile-manuf</td>
</tr>
<tr>
<td></td>
<td>no of employees &gt; 15</td>
</tr>
<tr>
<td></td>
<td>address = ?</td>
</tr>
<tr>
<td>B</td>
<td>article</td>
</tr>
<tr>
<td>C</td>
<td>poison</td>
</tr>
<tr>
<td></td>
<td>industrial name = POI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJECT RELATION</th>
<th>DOMAINS</th>
</tr>
</thead>
<tbody>
<tr>
<td>produce</td>
<td>A, B</td>
</tr>
<tr>
<td>contain</td>
<td>B, C</td>
</tr>
</tbody>
</table>

Figure 5
5.4.2 Data base interaction languages

The example of the previous section leads in a natural way to the general subject and problem area of data base interaction languages. The vivid discussions about the pros and cons in connection with generalized data base management systems and standardized generalized data base management systems have very much come to circle around this subject. For instance the CODASYL proposal [7] is best characterized as the specification of a standardized data base interaction language for a particular kind of data base interactors. Naturally such a language specification imposes other internal and external characteristics upon a data base (management) system which supports the particular language interface, but it is certainly not the same as a complete specification of such a system. Thus it seems to have been more or less neglected in the debate around the CODASYL proposal that

- a data base interaction language is only a subsystem of a complete data base (management) system

- the goals of the language interface and the design of it are dependent upon the goals and the design of the other data base functions, which in turn are dependent upon the overall goals of the data base system

- it is at least doubtful whether it is advisable to standardize the design of a subsystem of a data base system before there is some minimum amount of general data base theory

- different kinds of interactor categories may need different kinds of interaction languages

---

1 Cf our discussion in 4.3 and 5.2.1 of different categories of interactors. The data base interaction language suggested by CODASYL seems to be intended for a category of information consumers (users) who are capable of COBOL programming. It is certainly not intended for the categories which we have labeled "data structurers" and "information structurers", for instance.

2 In the data base subsystem structure suggested in chapter 7 the language interface, "interaction", is one of eight sub-systems.
- the bulk of data base interactors are only interested in the data base as a black box reservoir of information; they are interested to know what information is available, how fast and at what cost it may be retrieved, the true meaning and relevant implications of retrieved information, etc, and they are not interested in the data base internal file and data structures, however sophisticated they may be.

- on the other hand, an important minority of data base interactors, the data structurers (cf 5.2.1), need conceptual and language tools, which enable them to evaluate continuously data base performance against data base goals and to undertake datalogical restructurings in a convenient way.

If due attention is paid to these issues by data base decision-makers, interactors, scientists, systemeers, and other interested parties, it seems reasonable to guess that tomorrow's data base interaction languages will be quite different from those which are available on the market today. Let us consider three aspects of a data base interaction language and discuss a possible development for each of them separately. The three aspects, or dimensions, may be characterized by the following extreme point pairs:

(a) natural - formal

(b) procedural - result oriented

(c) reality oriented - computer oriented

As to dimension (a) it is easy, but probably not correct, to predict a dramatic development towards natural language - like data base interaction languages. The reason why this development will probably not be so striking, except possibly for very special applications, is not primarily the problems in connection with the design of natural language compilers and interpreters. The basic reason is that the applications for which data bases will be built are rather different from the every-day situations, where a natural language is at its best.
When we deal with complex decision problems, it is essential that we express ourselves with great logical precision, and logical precision is more characteristic of formal than of natural languages. On the other hand formalism due to shortcomings of computer hardware and software will be less important in the future. The resulting development will probably involve some movement in the natural language direction, more because the data base systems will support more "intelligent" and "flexible" conversation between the interactor and the data base, and less because the interaction languages will have more natural language - like vocabularies and syntaxes.

As for dimension (b), we may first establish that most programming languages in use today require an information request to contain three functional parts:

(1) an input description, describing (a part of) the data base

(2) an output description, describing the desired result of the request

(3) a process description, describing how the input should be transformed into the output

We can expect all data bases in the future to contain descriptions of themselves, so that input description will become superfluous. It is more interesting to discuss whether it will always be feasible to describe the desired output in such a way that no sub-steps of the output-producing algorithm will need explicit specification; if so the data base interaction languages will be perfectly result-oriented in the future, whereas today they are by and large procedural.

No doubt the development in this dimension will be fast and in the direction towards result-oriented languages. We have already seen examples of, how information requests may be formulated in a perfectly non-procedural way be means of infological models, and more complex situations will be treated in the next chapter. However, when we widen the range of information requests that we are able to communicate in a result-oriented way to the data
base, we will typically have to widen the vocabulary of the interaction language, too. In some situations, therefore, process descriptions may simply be the most practical way of describing desired results.

As for dimension (c), finally, development has been going on since the childhood of computers. This development has brought with it so-called high-level or problem-oriented languages like COBOL. Lately there has been much discussion about "data independence". There is no perfectly clear definition of "data independence", but the more changes may be undertaken in the mappings between the file system and the computer, without the user having to alter his mode of expression in order to retrieve a certain piece of information, the higher degree of data independence there seems to be. With this definition data independence is a necessary but not a sufficient condition for "reality-orientation", because even if the retrieval expressions of a data base interaction language are by and large invariant to changes of the kind mentioned, the mode of expression need not at all reflect "reality" but could just as well reflect the structure of some idealized or standardized computer. For instance, a data base interaction language which permits an information consumer to express himself in terms of an invariant chain, tree, or network structure of files and records certainly has a high degree of data independence, but it is "reality-oriented" only if the interactor usually thinks of reality in terms of chain, tree and network structures of files and records.

An extremely "reality-oriented" data base interaction language would be a language permitting every particular interactor to communicate directly, without any "model definition preludes", with the data base in terms of his own mental conceptualization of the slice of reality about which information is stored in the data base (cf chapter 1). Somewhat less ambitious would be to prepare the data base with a set of communication interfaces corresponding to "standard models" within the disciplines to which the interactors belong. For instance, economists have their standard models, sociologists have theirs, etc. The data
base system would then probably first translate expressions in terms of the subject matter models to "normal expression" in terms of some invariant model of the contents of the data base, e.g. a particular infological model of the kind discussed earlier in this report. Still less "reality-oriented", but yet ambitious in comparison with today's situation would be to make the different interactor categories agree upon one particular infological model, which would then become the basis of all communication with the data base. Skeletons for such languages could be provided by standard software manufacturers.

Thus we may conclude that we may expect future data base interaction language to be more conversational and more result-oriented, and that they will be based upon structures which are more attractive to "reality-oriented" data base interactors. Languages based upon the infological framework developed in this report could certainly be equipped with these general features.
6. Box structures of aggregate messages

6.1 Introduction

In chapter 5 we defined and analyzed so-called αβ-queries. An αβ-query requests the retrieval of a single elementary message, or of a consolidated message which is an unstructured conjunction of elementary messages. In this chapter we shall treat retrieval requests possessing one or more of the following characteristics:

(a) The query requests the retrieval of a structured conjunction of e-messages, e.g. a list of e-messages, sorted by one or more arguments.

(b) The query requests the fabrication, or aggregation, of aggregate messages from sets of e-messages. Aggregate messages are messages, the object parts of which refer to compound objects\(^1\), and the predicate parts of which are functions of predicate parts of messages, the object parts of which refer to individual member objects of the compound objects referred to by the aggregate messages. For a more concrete discussion, see section 6.3.3.

(c) The query requests the retrieval of a structured conjunction of aggregate messages, e.g. a table of averages, sorted by one or more arguments.

In the subsequent sections we propose two principal conceptual tools by means of which retrieval requests with the characteristics (a) - (c) may be systematically analyzed. The tools are (i) the box concept, to be defined and discussed in section 6.2, and (ii) the concept of an αβγ-query pattern, which will be treated in section 6.3. In section 6.4, finally, we shall make an attempt to illustrate how the tools may be utilized (i) for infological, representation-independent description of data base processes, (ii) in connection with certain quality and protection problems, and (iii) in the development of result-oriented, non-procedural data base interaction languages.

---

\(^1\)Cf section 2.2.1
6.2 A general theory of boxes

6.2.1 Basic concepts

We shall begin with a naive approach to box theory. Figure 1 visualizes a three-dimensional box, which is seen to be a structured set of elementary cells, e-cells. The elementary cells in turn are defined as sets, the elements of which form the contents of the e-cells. The elements of the e-cells may be differently interpreted in different applications of general box theory. When we apply box theory within the infological framework, which is the theme of this report, we shall make a particular kind of interpretation, which will be discussed in later sections of this chapter. In general box theory we leave the elements undefined.

In figure 1 we showed a three-dimensional instance of the box concept. In general the dimension, \( m \), of a box may be any non-negative integer. Naturally, boxes whose dimension is greater than three are difficult to visualize properly.

Each of the \( m \) dimensions of a box is said to be spanned by an axis, or variable. Unlike traditional coordinate systems there is only a finite number of values along each box axis. Thus the range of each box spanning variable \( V_i \) is a finite set of values \( R(V_i) = \{ a_{i1}, \ldots, a_{in_i} \} \), where \( i \in \{ 1, \ldots, m \} \), and \( n_i \) is a positive integer.

The idea of a set of variables spanning a box suggests in a natural way a symbolic representation of the elementary cells. They are denoted by tuples

\[
(1) \quad \langle j_1, \ldots, j_m \rangle
\]

where \( j_i \) is the second sub-script of one of the elements in \( R(V_i) \), \( i = 1, \ldots, m \).

We may now define boxes formally.
Figure 1  Three-dimensional box consisting of $2 \times 3 \times 4 = 24$ elementary cells
Definition. A box algebra is a tuple

$$\langle c, m, \vec{n} \rangle$$

where

(3) \( m \) is a non-negative integer

(4) \( \vec{n} \) is an \( m \)-tuple \( \langle n_1, \ldots, n_m \rangle \) of positive integers

(5) \( C \) is a set, whose elements are called cells, which are also sets

(6) there is a subset \( K \) of \( C \), called the kernel of \( C \), which is isomorphic with the set of integer \( m \)-tuples \( \langle j_1, \ldots, j_m \rangle \), \( 1 \leq j_i \leq n_i \), and whose elements are called e-cells

(7) \( c \in C \) if and only if \( c \) is a finite union of e-cells in \( K \)

Definition. A subset \( B \) of the set of cells \( C \) in a box algebra is a box if and only if

(8) \( \langle x_1, \ldots, x_m \rangle \in B \land \langle y_1, \ldots, y_m \rangle \in \vec{B} \rightarrow \langle z_1, \ldots, z_m \rangle \in B \) for all \( z_i \in \{ x_i, y_i \} \), \( i = 1, \ldots, m \)

and

(9) \( c \in B \) is and only if \( c \) is a finite union of e-cells in \( B \cap K \)

It is easy to prove for example that \( C \) itself as well as every elementary cell in \( C \) fulfil the box conditions.

Subsets of boxes, which are also boxes, are called sub-boxes. Other subsets are called cell collections.
If \( <C, m, \bar{n}> \) is a box algebra and \( K \) is the kernel of \( C \), then every sub-box of \( C \) contains a subset of \( K \), and that subset is called the **kernel of the sub-box**.

The kernel of a box is said to **generate** the box.

**Example.** Figure 2 visualizes a two-dimensional box spanned by the variables \( V_1 \) and \( V_2 \). The ranges of the variables are \( R(V_1) = \{ a_{11}, a_{12}, a_{13}, a_{14} \} \) and \( R(V_2) = \{ a_{21}, a_{22}, a_{23} \} \) respectively. The kernel of the box is a set of \( 4 \times 3 = 12 \) elementary cells \( <x, y> \), where \( x \in \{1, 2, 3, 4\} \) and \( y \in \{1, 2, 3\} \). A set of four cells

\[
(10) \quad K' = \{<1,1>, <3,1>, <1,3>, <3,3>\}
\]

is shaded in figure 2. It is easy to show that \( K' \) is the kernel of a sub-box of the original box. Note that for instance the set

\[
(11) \quad S = \{<1,1>, <3,3>\}
\]

does not generate a box. According to criterion (8), \( S \), if it were a box, should satisfy

\[
(12) \quad <1,1> \in S \wedge <3,3> \in S \implies <1,3> \in S \wedge <3,1> \in S
\]

which it obviously does not.

An important observation may be made from the example. Let us define two variables, \( V'_1 \) and \( V'_2 \), which differ from \( V_1 \) and \( V_2 \) only by their ranges:

\[
(13) \quad R(V'_1) = \{ a_{11}, a_{13} \} \subset R(V_1)
\]

\[
(14) \quad R(V'_2) = \{ a_{21}, a_{23} \} \subset R(V_2)
\]

The new variables with their restricted ranges are seen to span exactly the sub-box which is generated by kernel \( K' \), and
which is shaded in figure 2. This is not a mere coincidence. One may prove that the sub-boxes of a box, as defined above, and the boxes spanned by restrictions of the original variables are pairwise identical.

Note that all sub-boxes, even single e-cells, have the same dimension as the original box.

Beside the sub-boxes there are other boxes, which are closely related to a particular box. They are called projections. Projection is a box transforming operator, π, which like the restriction operator defined above reduces the number of elements in the range of at least one variable. Unlike restriction, however, projection does not discard the contents of any cells.

---

1 If the range of a variable V' is a subset of the range of variable V, then we shall say that V' is a restriction of V. A box B' is said to be a restriction of a box B if and only if there is a biunique correspondence between the variables spanning B' and B, and each variable spanning B' is identical with or a restriction of a variable spanning B.
Figure 2. The shaded e-cells from a kernel generating a sub-box which is a restriction of the original box. The contents of the restriction form a subset of the contents of the original box.

Figure 3. A partial projection of the box in figure 2. The contents of the projection form a set which is identical with the contents of the original box.

Figure 4. Complete projection of a two-dimensional box into a one-dimensional.
Example. Figure 3 shows a projection of the two-dimensional box in figure 2. The projected box has also two dimensions but the cardinality of the range of one of the box-spanning variables has been reduced: the values $a_{13}^\prime$ and $a_{14}^\prime$ of $V_1$ have "merged" into the single value $a_{13}$ of $V_1'$. $V_2$ has been left unchanged in this projection. The contents of the $e$-cell $<3, y>$ in the new box are identical with the contents of the non-elementary cell $<3, y> \cup <4, y>$ in the original box ($y = 1, 2, 3$). Every member element of a cell in the original box is thus to be found in some elementary cell of the projected box. This is not true for restrictions: e.g. the contents of $e$-cell $<4, 1>$ in the box in figure 2 are not to be found in the shaded sub-box.

The projection in the example above may be called a partial projection. In a complete projection all the values of least one variable are "merged", so that the variable is actually eliminated from the box structure. This implies dimension reduction: if an $m$-dimensional box is completely projected along $k$ variables, the dimension of the resulting box will be $m-k$.

Example. Figure 4 visualizes a projection which may be described by the formula

(15) \[ \pi_{V_1}(B) = B' \]

The two-dimensional box $B$ spanned by the variables $V_1$ and $V_2$ has been completely projected along $V_1$ into the one-dimensional box $B'$ spanned by $V_2$. E-cell $<1>$ of $B'$ contains the elements in $<1,1> \cup <2,1> \cup <3,1> \cup <4,1>$ of $B$ etc. The projection has reduced the structure but not the contents of the original box.
6.2.2 A convenient notation

In the previous section we presented the basic concepts of a general box theory. Before we start using box theoretical concepts within the infological theory of data bases, we shall introduce a convenient notation for representing certain important classes of subsets of a box.

It follows from the theory of boxes that any subset of a box is a set of unions of e-cells. As was pointed out in the previous section, any e-cell in an m-dimensional box may be denoted by an m-tuple

\[(1) \quad \langle j_1, \ldots, j_m \rangle\]

where every \(j_i\), \(i = 1, \ldots, m\), is an integer not less than 1 and not greater than \(n_i\), the cardinality of the range of variable \(V_i\), spanning the i:th dimension of the box. As any cell of a box has to be an e-cell or a finite union of e-cells \(^1\), an arbitrary cell may be denoted

\[(2) \quad \bigcup_{i=1}^{k} x_i = x_1 \cup x_2 \cup \ldots \cup x_k\]

where each \(x_i\) is an m-tuple \((1)\).

An arbitrary subset of a box may thus be denoted

\[(3) \quad \{y_j\}_{j=1}^{p} = \{y_1, \ldots, y_p\}\]

where each \(y_j\) is an e-cell union \((2)\).

The notation format defined in \((1)\) - \((3)\) is very general and very explicit, but in many practical and analytical situations of interest it will be very inconvenient and render bad justice to the structure, which is often inherent in a subset of a box, implied by the strong structure of the box itself.

\(^1\) See \((7)\) of section 1. An e-cell is a trivial union of an e-cell with itself.
A few simple examples will suggest important improvements of the notation, which are easy to implement, learn, and use.

Consider first the sub-box indicated in figure 5, which is a duplication of figure 2. According to format (3) the kernel of that sub-box, i.e., the set of e-cells generating the sub-box, would be denoted

\[(4) \quad \{<1,1>, <1,3>, <3,1>, <3,3>\}\]

Without introducing any ambiguity, however, this may be simplified to

\[(5) \quad \{<1 \lor 3, 1 \lor 3>\}\]

Using postulate (8) of section 1, one may prove that the kernel of any sub-box of an m-dimensional box may be represented by a single "disjunctive tuple" analogous to (5).

According to the disjunctive tuple notation the kernel of the original box of figure 5 would be denoted

\[(6) \quad \{<1 \lor 2 \lor 3 \lor 4, 1 \lor 2 \lor 3>\}\]

This may be further simplified to

\[(7) \quad \{<-, ->\}\]

where "-" represents a "full disjunction", i.e., a disjunction of all members of the range of a variable.

Analogously

\[(8) \quad \{<- , ..., ->\}\]

where <-, ..., -> is an m-tuple containing m "-" symbols may be used to represent the kernel set of e-cells of any m-dimensional box.
Figure 5 \[ \langle 1 \vee 3, 1 \vee 3 \rangle \]

Figure 6 \[ \langle 1 \vee 2 \vee 3 \vee 4, \rangle \]
So far we have only treated such subsets of boxes as contain e-cells only. Thus let us see how we may also simplify the representation of a few important classes of non-elementary subsets, i.e., subsets containing compound cells, unions of e-cells.

In the previous section we described projection as a box operator which transforms one box into another. For instance we saw in figure 3, which is duplicated here as figure 6, how a box spanned by the variables

\[(9) \quad V_1 \text{ with } R(V_1) = \{a_{11}, a_{12}, a_{13}, a_{14}\} \quad \text{and} \quad V_2 \text{ with } R(V_2) = \{a_{21}, a_{22}, a_{23}\}\]

was transformed into another box spanned by

\[(10) \quad V'_1 \text{ with } R(V'_1) = \{a'_{11}, a'_{12}, a'_{13}\} \quad \text{and} \quad V_2\]

where the projection operator has operated along the \(V_1\)-axis of the original box so that

\[(11) \quad a'_{11} = a_{11}, \quad a'_{12} = a_{12}, \quad \text{and} \quad a'_{13} = a_{13} \cup a_{14}\]

i.e., in such a way that the contents of the e-cells \(<3, y>\>

and \(<4, y>\>

have been merged into the e-cell \(<3, y>\>

of the

projected box \((y = 1, 2, 3)\).

However, we may also conceive of the projection operator as an operator which selects a subset of cells from a given box. Figure 6 would then show how a subset of 9 cells, namely

\[(12) \quad c_1 = \langle 1,1 \rangle \]

\[c_2 = \langle 1,2 \rangle \]

\[c_3 = \langle 1,3 \rangle \]

\[c_4 = \langle 2,1 \rangle \]

\[c_5 = \langle 2,2 \rangle \]

\[c_6 = \langle 2,3 \rangle \]

\[c_7 = \langle 3,1 \rangle \cup \langle 4,1 \rangle \]

\[c_8 = \langle 3,2 \rangle \cup \langle 4,2 \rangle \]

\[c_9 = \langle 3,3 \rangle \cup \langle 4,3 \rangle \]
have been selected from the box, which consists of

\[ 2^{4 \times 3} - 1 = 4095 \]

elementary and compound cells. We may observe that the selected subset contains 6 e-cells and 3 e-cell unions, or compound cells. When we considered projection a transformation rather than a selection operator, we conceived of figure 6 as visualizing a new box kernel consisting of 9 elementary cells.

Now let us turn to the question how to denote subsets of boxes selected by the projection operator. We shall introduce a convention which implies the subset (12) to be denoted.

\[ \{ <1 \lor 2 \lor 3 \times 4, \rightarrow \} \]

Hopefully, this example explains the convention now introduced. However, let us for the sake of clarification consider another couple of examples still referring to the original two-dimensional box, which was used in the earlier examples.

**Examples.** \( \{ <1 \times 2 \lor 3 \times 4, \rightarrow \} \) would denote a sub-set of 6 compound cells, namely

\[ c_1 = <1,1> U <2,1> \]
\[ c_2 = <1,2> U <2,2> \]
\[ c_3 = <1,3> U <2,3> \]
\[ c_4 = <3,1> U <4,1> \]
\[ c_5 = <3,2> U <4,2> \]
\[ c_6 = <3,3> U <4,3> \]

The subset is visualized in figure 7.

Figures 8-13 visualize the subsets

\[ \{ <1 \times 2 \lor 3 \times 4, 1 \lor 2 \times 3 > \} \]

\[ \{ <x, \rightarrow > = \{ <1 \times 2 \times 3 \times 4, \rightarrow > \} \]

\(^1\)Note the priority between the symbols: "\( \times \)" precedes "\( \lor \)".
Figure 7
\(<1 \times 2 \vee 3 \times 4, \rightarrow\)
Figure 10
\{<-, x>\}

Figure 11
\{<x, x>\}

Figure 12
\{<1 \times 2 \lor 3, 1 \times 2 \lor\}
(\text{shaded})

Figure 13
\{<1 \times 2 \lor 3, 1 \lor 3 \lor\}
(\text{shaded})
\( (18) \quad \{\langle -, x \rangle \} = \{\langle -, 1 \times 2 \times 3 \rangle \} \)

\( (19) \quad \{\langle x, x \rangle \} = \{\langle 1 \times 2 \times 3 \times 4, 1 \times 2 \times 3 \rangle \} \)

\( (20) \quad \{\langle 1 \times 2 \lor 3, 1 \times 2 \rangle \} \)

\( (21) \quad \{\langle 1 \times 2 \lor 4, 1 \lor 3 \rangle \} \)

respectively. Note that (17) - (19) correspond to the complete projections \( \pi_{V_1}(B), \pi_{V_2}(B) \) and \( \pi_{V_1,V_2}(B) \)\(^1\). Also note the introduction of the single "\( x \)" symbol, which is analogous to the introduction of the single "\( - \)".

The simplifications introduced so far permit a short-hand notation for subsets of a box which contain mutually exclusive cells only, i.e., whatever entities occur in the cells of the box they may only occur in one cell of the sub-set. In practice, however, one often wants to consider several, related classifications of the same set of entities at the same time. For instance, the structure of an ordinary statistical table may be considered to be built up from a basic structure, corresponding to the kernel of a box, and a few additional structures, which, like different levels of marginal sums, are functions of the kernel, both as far as structure and contents are concerned.

The additional symbol needed to represent these important "subsets of overlapping cells" will be written "\( \cdot \)" in this paper. Its use will be explained by a very simple table example.

Consider a box \( B \) spanned by the variables \( V_1 = \) "region" with \( R(V_1) = \{A,B,C\} \) and \( V_2 = \) "sex" with \( R(V_2) = \{M,F\} \). A "normal" table based on the structure and contents\(^2\) of this box could have the structure indicated in figure 14.

---

\(^1\) See section 1 for definitions of partial and complete projections and explanation of the "\( \pi_{V}(B) \)"-symbolism.

\(^2\) We do not discuss the contents of the cells in this section.
The table structure in figure 14 is a subset of the box B. B contains \(2^{3^2} - 1 = 63\) cells, 6 of which are elementary and form the kernel. The subset contains these kernel e-cells together with 6 compound cells, namely the 3 cells selected by the projection \(\pi_{V_2}(B)\), the 2 cells selected by the projection \(\pi_{V_1}(B)\) and the single cell selected by the projection \(\pi_{V_1 V_2}(B)\).

The table structure is thus the union of four subsets of the type which has been discussed earlier, i.e., subsets in which the entities of the box occur in one and only one cell. Note that in figure 14 three of the four subsets have been shrunk in order to make the whole look more like a typical table presentation. With the conventions used in earlier figures, figure 14 would have looked like figure 15, where the four basic sub-sets of the subset are visualized on the same scale.

According to the notation introduced before, the subset \(S\) of \(B\) corresponding to the whole table would be denoted as a union of four arguments \(S_1 \cup S_2 \cup S_3 \cup S_4\), where

\[
\begin{align*}
S_1 &= \{<-, ->]\} \\
S_2 &= \{<x, ->]\} \\
S_3 &= \{<-, x>\} \\
S_4 &= \{<x, x>\}
\end{align*}
\]

\(S_1\) is the kernel subset of \(B\) and is visualized by the upper left rectangle of figure 15. \(S_2\), upper right, is selected by the projection \(\pi_{V_1}(B)\), and \(S_3\), lower left, is selected by \(\pi_{V_2}(B)\). \(S_4\), finally, visualized by the lower right rectangle of figure 15 is selected by the projection \(\pi_{V_1 V_2}(B)\). Thus

\[
S = \{<-, ->, <x, ->, <-, x>, <x, x>\}
\]

We observe that the tuples of \(S_1\) and \(S_2\) differ only by their first component, which is "-" in \(S_1\) and "x" in \(S_2\). Let us therefore denote the union of \(S_1\) and \(S_2\) by
Figure 14

Figure 15
(27) \[ S' = \{\langle \cdot, -, \cdot \rangle\} \]

Similarly the tuples of \( S_3 \) and \( S_4 \) differ only by their first component and the union of them may accordingly be denoted

(28) \[ S'' = \{\langle \cdot, \cdot \rangle\} \]

Now \( S = S' \cup S'' \), but the tuples of \( S' \) and \( S'' \) differ only by their second component, and \( S \) may thus be represented by one single tuple

(29) \[ S = \{\langle \cdot, \cdot \rangle\} \]

More generally a set containing an \( m \)-tuple with an expression containing the "\( \cdot \)" symbol in its \( i \)-th place is defined by

(30) \[
\{\ldots, \langle x_1, \ldots, x_i = \text{expr}(\cdot), \ldots, x_m \rangle, \ldots\} = \\
= \{\ldots, \langle x_1, \ldots, x_n^1 = \text{expr}(\cdot), \ldots, x_m \rangle, \\
\langle x_1, \ldots, x_n^i = \text{expr}(x), \ldots, x_m \rangle, \ldots\} \]

The other places of the \( m \)-tuple, \( x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m \), may contain any of the symbols "\( - \)", "\( \times \)", and "\( \cdot \)”, or any expression using "\( \lor \)" and "\( \land \)" and any of the values of the range of the appropriate variable.

**Example.** The subset of the box figures 6-13 consisting of the four subsets which are shaded in figure 16 would be denoted

(31) \[ \{\langle 1 \cdot 2, 2 \cdot 3 \rangle\} \]

This may be developed in two steps, using formula (30):

(32) \[
\{\langle 1 \cdot 2, 2 \cdot 3 \rangle\} = \\
= \{\langle 1 \lor 2, 2 \cdot 3 \rangle, \langle 1 \times 2, 2 \cdot 3 \rangle\} = \\
= \{\langle 1 \lor 2, 2 \lor 3 \rangle, \langle 1 \lor 2, 2 \times 3 \rangle\}, \\
\{\langle 1 \times 2, 2 \lor 3 \rangle, \langle 1 \times 2, 2 \times 3 \rangle\}
\]

\(^1\) As has been pointed out before, some expressions containing "\( \lor \)" may be replaced by a single "\( \cdot \)".
Figure 16 \{\langle 1 \cdot 2, 2 \cdot 3 \rangle\}
Let us return to our simple table structure. Beside \( \{< *, * >\} \), visualized in figure 15, there are 8 other subsets of the box \( B \), which may be denoted by a single tuple containing "clean" 
"-", "x", and "." symbols. All these \( 3^2 = 9 \) different subsets of the original box are visualized in figures 17-25. In two dimensions it is naturally not particularly difficult to keep the different subsets apart, and to tell without any formal notation which of the structures one wants to be presented. In \( m \) dimensions, however, we may express by a single, "clean"\(^1\) tuple in our notation any one out of \( 3^m \) different subsets of the \( m \)-dimensional box, which we have defined to be of interest. It is very likely that the very subset, which we are particularly interested in, will be one of these \( 3^m \). On the other hand it is very unlikely that for large \( m \) we would be able to explain even to ourselves, exactly which of the \( 3^m \) subsets we want, without the aid of a stringent theory of table structures and a compact, formal notation to accompany it.

\(^1\) By a "clean" tuple we mean a tuple where all components are either "-" or "x" or "; no expressions and no numbers referring to single variable values occur. If "non-clean" tuples are allowed, the number of sub-sets expressible by a single tuple in our notation will of course increase dramatically.
6.3 Box theory and the infological framework

6.3.1 a8-queries and a8y-queries

We are now prepared to study how the box theoretical concepts and notation mode may be utilized in our infological approach to data bases. Chapter 5 brought us to the definition of a general retrieval query pattern:

\[(1) \quad \text{"For all objects having the property } P^\alpha, \text{ retrieve the values of the attributes } A_{1}^{\beta}, \ldots, A_{m}^{\beta} \text{ at the times } t_{1}^{\beta}, \ldots, t_{m}^{\beta}, \text{ respectively."} \]

We recall that the property \( P^\alpha \) may be generated from a set of \( \alpha \)-attributes, \( \alpha \)-times and \( \alpha \)-relations. Queries conforming to pattern (1) will be called a8-queries.

We claimed in chapter 5 that pattern (1) is general, not only in the sense that many retrieval queries are actually a8-queries, but also in the sense that more complex data base transactions may often be broken down into a8-queries. Very often the set of a8-queries, which constitutes the more complex transaction, has a structure, which may be neatly described in box-theoretical terms.

Consider for example requests like

\[(2) \quad \text{"List the names and telephone numbers of all employees by department."} \]

\[(3) \quad \text{"Compute average income 1971 for different social classes in Stockholm."} \]

The \( \alpha \)-parts of the a8-queries making up these and similar requests may be formalized in the following way:

\[(4) \quad P_{1}^{\alpha} = P_{o}^{\alpha} \land (v^{\gamma} = a_{1}) \]
\[\vdots\]
\[P_{n}^{\alpha} = P_{o}^{\alpha} \land (v^{\gamma} = a_{n}) \]
In (2) the common $\alpha$-property $P^\alpha_0$ is "to be an employee", and in (3) it is "to be an inhabitant in Stockholm". The common variable\(^1\), which we shall call a $\gamma$-variable, $V^\gamma$, is "department" in (2) and "social class" in (3). "n" is equal to the number of departments and the number of social classes, respectively, i.e the number of values of $V^\gamma$.

More generally the $\alpha$-parts of the set of $\alpha\beta$-queries making up a complex transaction may contain several, say $k$, common $\gamma$-variables. Then the set of $\alpha$-parts may be formalized in the following way:

\[(5)\]
\[
P^\alpha_1 \cdots 1 = P^\alpha_0 \land (V^\gamma_1 = a_{11}) \land \cdots \land (V^\gamma_k = a_{k1})
\]
\[
P^\alpha_1 \cdots 2 = P^\alpha_0 \land (V^\gamma_1 = a_{11}) \land \cdots \land (V^\gamma_k = a_{k2})
\]
\[
\vdots
\]
\[
P^\alpha_1 \cdots n_k = P^\alpha_0 \land (V^\gamma_1 = a_{1n_k}) \land \cdots \land (V^\gamma_k = a_{kn_k})
\]
\[
P^\alpha_{n_1} \cdots 1 = P^\alpha_0 \land (V^\gamma_1 = a_{1n_1}) \land \cdots \land (V^\gamma_k = a_{k1})
\]
\[
\vdots
\]
\[
P^\alpha_{n_1} \cdots n_k = P^\alpha_0 \land (V^\gamma_1 = a_{1n_k}) \land \cdots \land (V^\gamma_k = a_{kn_k})
\]

There are $\prod_{i=1}^{k} n_i$ rows in (5); $n_i$ = the number of values that $V^\gamma_i$ can take.

Example. In the request

\[(6)\]
\["Compute the average length of life for human beings by nationality and sex."
\]

$P^\alpha_0$ = "to be a human being", $V^\gamma_1$ = "nationality", $V^\gamma_2$ = "sex", $k = 2$, $n_1$ = the number of nationalities, $n_2 = 2$.

Remark. In the formulas above we have not made explicit reference to "time", because we thought it would only obscure the main line of analysis here. However, we may easily think

\(^1\) In this chapter we shall only deal with single-valued attributes. According to the definition in section 2.3.3 such attributes are called variables.
of all involved properties and variables as time-dependent. Similarly, the properties and variables may very well be relation-dependent. For further explanation of the underlined terms, see section 2.2.2, example 4 and example 5.

Definition. A data base transaction, which may be dissolved into a set of αβ-queries, the α-parts of which conform to pattern (5), will be called an αβ-query.
6.3.2 Object boxes

In section 6.2.1 we defined the cells of a box as sets of entities, but we did not say anything about the nature of these entities. Such interpretations we explicitly left outside the realm of general box theory. In connection with aγ-queries we shall consider boxes, the cells of which contain objects, facts, and messages, respectively. Let us begin with the object boxes, the cells of which contain objects.

It is natural to relate an αγ-query to an object box B with the following characteristics:

(a) B is spanned by the γ-variables \( V_1^γ, \ldots, V_k^γ \), and possibly by "time". If "time" is among the box-spanning variables, the dimension of B is \( k+1 \), and B is called a time-series box. If "time" is not among the box-spanning variables, the dimension of B is \( k \), and B is called a time-slice box. For time-slice boxes a time value is usually tacitly understood. A time series box consists of one or more time-slice sub-boxes.

(b) An e-cell \( \langle j_1, \ldots, j_k, t \rangle \) in B contains all objects having

- property \( P^α_o \) at time \( f(t) \), and

- property \( (V_1^γ = a_{1j_1}) \wedge \ldots \wedge (V_k^γ = a_{kj_k}) \) at time \( g(t) \)

where \( f \) and \( g \) are functions of the time variable, and where \( a_{ij_i} \) is a value in the range of \( V_i^γ \), \( i = 1, \ldots, k \). The properties will be referred to as "the α-property" and "a particular combination of γ-properties", respectively.

1 If B is a time-slice box the time component will not be present in the tuple.
Remark. Very often $f$ and $g$ are simply both the identity function, but there are exceptions to this rule. For example, if we want to follow the development of a certain cohort of objects over time\(^1\), $f$ will be constant for all time slices of the time-series box $B$. More generally $f$ and $g$ may be vector-valued functions with one $f$ component for each "sub-property" involved in $P^\mathcal{O}_i$, and with one $g$ component for each of the $k$ $\gamma$-variables, $V_1^\gamma$, ..., $V_k^\gamma$.

With the definition above each e-cell of the box $B$ will contain those objects which \textit{actually}\(^2\) have a particular property in common. Such an object box will be called a \textit{factual object box}, or an \textit{fo-box}. However, if we use a source of information, like a data base, we are forced to resort to the \textit{alleged object box}, the \textit{ao-box}, each e-cell of which will contain those objects which have a particular property in common according to the information available (in the data base). As this information may be more or less true\(^3\), there may be serious or less serious discrepancies between the fo-box and the ao-box corresponding to a particular $\alpha\beta\gamma$-query.

Remark. There is nothing requiring that the objects in an object box be atomic\(^4\). They may equally well be compound. For example, the objects belonging to a particular object box may be identical with the object tuples belonging to a particular object relation. The $\gamma$-variables used to classify the object tuples into box cells may either be variables which are relevant to the compound objects, the object tuples, or variables which are relevant to component objects of the object tuples.

\(^1\) Studies of this kind are known as "cohort studies" in statistics.
\(^2\) Cf section 2.3.1.
\(^3\) Cf section 3.2.1 and section 3.2.4.
\(^4\) Cf section 2.2.1.
6.3.3 Fact and message boxes

When we discussed αβγ-queries in the two previous sections, we said nothing about the β-parts of them. Nor did we relate the β-part of an αβγ-query to the β-parts of the individual αβ-queries which together constitute the αβγ-query. Let us therefore once again consider the following two examples, which represent two different types of αβγ-queries:

(1) "List the names and telephone numbers of all employees by department."
(2) "Compute average income 1971 for different social classes in Stockholm."

The object boxes of both these queries are one-dimensional. Let us call them B₁ and B₂ respectively. An e-cell in B₁ contains all employees belonging to a particular department. For each of these objects query (1) requests the values of the β-variables $\bar{v}_1^{\beta} = "name"$ and $\bar{v}_2^{\beta} = "telephone number"$.

An e-cell in B₂ contains all Stockholmers belonging to a particular social class. Query (2) does not request the value of the β-variable $\bar{v}^{\beta} = "income"$ for each of these objects. Instead query (2) requests values of a variable $\bar{v}^{\beta} = "average income"$. This variable has two important characteristics:

(a) The relevance group² of $\bar{v}^{\beta}$ does not contain individual Stockholmers, but compound objects³ each of which is a set of Stockholmers. There is one such compound object for each of the cells⁴ in the object box corresponding to query (2).

(b) $\bar{v}^{\beta}$ is a known function of $v^{\beta}$, and the value of $\bar{v}^{\beta}$ for a particular compound object $\{o_i\}$ is expressible in terms of the values of $v_i^{\beta}$ for the individual objects $o_i$ in this compound object:

---

¹ Cf section 6.3.1, (2) and (3).
² Cf section 2.3.3
³ Cf section 2.2.1
⁴ Both elementary and non-elementary. Cf section 6.2.1.
\[ \#\{ o_i \} \]
\[ \bar{\nu}^\beta (\{ o_i \}) = \exp (\bar{\nu}^\beta (o_1))_{i=1} \]

Facts and messages involving variables which satisfy the general counterparts of (a) and (b) above will be called aggregate facts and aggregate messages.

Recalling the definitions in chapter 2 an (elementary) aggregate fact, or a-fact, may be formally defined as an e-fact \( \langle x, y, z \rangle \), where \( x \) is a compound object which is a set of atomic or compound objects, \( y \) is a complex property \( \bar{\nu} = \bar{a} \) fullfilling (a) and (b) above, and \( z \) is a time.

Similarly, an (elementary) aggregate message, or a-message, may be defined as an e-message \( \langle x, y, z \rangle \), where \( x \) refers to a compound object which is a set of atomic or compound objects, \( y \) refers to a complex property \( \bar{\nu} = \bar{a} \) fullfilling (a) and (b) above, and \( z \) refers to a time.

The \( \beta \)-part of an \( \alpha \beta \gamma \)-query like (2) thus involves two related sets of e-facts:

(a) A set of aggregate facts containing one a-fact \( \langle \{ o_i \}, \bar{\nu}^\beta = \bar{a}, t \rangle \) per cell in the object box corresponding to the query, and per \( \beta \)-variable in the \( \alpha \beta \gamma \)-query. The facts in this set, or some of them, are requested by the query.

(b) A set of elementary facts containing one e-fact \( \langle o_i, \nu^\beta = a, t \rangle \) per object in the object box corresponding to the query, and per \( \beta \)-variable in the \( \alpha \beta \)-queries into which the \( \alpha \beta \gamma \)-query may be dissolved. The facts in this set are not requested by the query but they are logical precedents of the facts which are requested.

For each \( \alpha \beta \gamma \)-query of type (2) we may now define two more boxes:

(a') An aggregate fact box, or af-box, having the same structure as the object box corresponding to the
query, but containing instead of objects the a-facts in the set (a) defined above. Each cell in the af-box will contain one a-fact per β-variable in the query, and the object part of all these a-facts will be the same compound object containing exactly those objects which are contained in the corresponding cell in the object box.

(b') An elementary fact box, or ef-box, having also the same structure as the object box corresponding to the query, but containing the e-facts in the set (b) defined above. Each cell in the af-box will, for each object in the corresponding cell in the object box, contain one e-fact per β-variable in the αβ-queries into which the αβγ-query may be dissolved.

Example. The af-box corresponding to the αβγ-query (2) above will in a particular e-cell contain the a-fact that a particular social class in Stockholm had in 1971 a particular average income. A particular e-cell in the ef-box corresponding to the same query will instead contain a bunch of e-facts, namely the e-facts that Stockholmer p₁ belonging to social class y had in 1971 a particular income, Stockholmer p₂ also belonging to social class y had in 1971 another income, and so on.

Remark. For αβγ-queries like (1) it is not possible to define an af-box. The ef-box corresponding to (1) will in each of its e-cells contain the e-facts that particular individuals, belonging to one and the same department, have particular names and telephone numbers. This box has thus two β-variables, whereas the ef-box corresponding to (2) has only one β-variable, "income".

Similarly as we have defined elementary and aggregate fact boxes, we may also define for each αβγ-query of type (2) an elementary message box, or em-box, and an aggregate message box, or am-box. There is a difference between fact boxes and message boxes which is analogous to the difference that we found in the previous section that there is between factual and alleged
object boxes. For example, a cell in an ef-box will contain facts concerning the $\beta$-variables for the objects which factually have the $\alpha$-property and a particular $\gamma$-property combination in common. A cell in an em-box will instead contain more or less correct information concerning the $\beta$-variables for objects which allegedly, according to the available information, have the $\alpha$-property and a particular $\gamma$-property combination in common.

We can see that there are three principally different ways in which a message box may be incorrect:

(a) The $\alpha$-information demarcating the set of objects belonging to the corresponding object box may be incorrect.

(b) The $\gamma$-information classifying the set of objects belonging to the corresponding object box may be incorrect.

(c) The $\beta$-information upon which the interest is focused by the $a_{\beta\gamma}$-query and which form the contents of the message box may be incorrect.

Remark. This analysis also suggests that if we should like more mnemonic terms than "$\alpha$-property", "$\gamma$-variables", and "$\beta$-variables", we could choose "demarcation property", "classification variables", and "focus variables", respectively.

Let us finally say something about "time" in connection with fact and message boxes. The fact and message boxes corresponding to a particular $a_{\beta\gamma}$-query has been defined to be isomorphic with the object box corresponding to the same query. According to the definition of the object box in section 6.3.2 this implies that "time" may be a dimension in the fact and message boxes, which will then be called time series boxes. For example, a particular e-cell $\langle j_1, \ldots, j_k, t \rangle$ in a $(k+1)$-dimensional time series em-box will contain:
information concerning the $\beta$-variables $v^\beta_1, \ldots, v^\beta_m$
at time $h(t)$ for all objects allegedly having

- the $a$-property $P^a_\rho$ at time $f(t)$, and

- the particular combination of $\gamma$-properties,
  \[(v^\gamma_1 = a^1_{j_1}) \land \ldots \land (v^\gamma_k = a^k_{j_k}),\]
at time $g(t)$

where $h$, like $f$ and $g$ are functions of the time variable, and
where the other entities are defined as in section 6.3.2.

Like $f$ and $g$, $h$ will often be the identity function, but this
is a rule with exceptions. In general, $h$, also like $f$ and $g$,
may be a vector-valued function. Then $h$ will have one component
for each of the $m$ different $\beta$-variables involved in the box.
6.4 Applications of infological box theory

6.4.1 Infological descriptions of data base processes

In this section we shall see how the infological interpretation of general box theory may be used to give non-computer-oriented people an adequate understanding of the processing which successively transforms the physically stored information contents of the data base, i.e. the contents of the data base nucleus\(^1\), into a requested output message. As was discussed in section 4.4.1 these transformations may be very complex and involve a lot of deduction operations as well as statistical inference. It is certainly not necessary for data base goal-setters, decision-makers, and information consumers\(^2\) to understand how the computer processes performing these transformations actually work. However, it is essential that they are aware of what important and maybe questionable information transformations are implied by the working of the computer processes. For example, this is essential for a correct conception of the meaning and quality of the infologically complex messages which will probably stream out of the data base once it has got into operation.

The processing which ultimately results in a reply to an \(a\gamma\)-query, or a more complex transaction built up from \(a\gamma\)-queries, may be conceived of as a series of box transformations. How this may be done is shown in principle by figure 26. We shall try to give an exemplifying explanation of this figure.

What the data base system has to accomplish first when it receives an \(a\gamma\)-query is to analyze it in order to find a set of boxes, probably \(em\)-boxes, the contents of which are physically stored in the data base nucleus, and from which the requested box, probably an \(am\)-box, may be constructed by successive deduction and inference using logical and statistical box operators. We shall not get deeper into the analysis phase of the processing of the \(a\gamma\)-query. Instead we shall have a

\(^1\) Cf section 4.4.
\(^2\) Cf section 4.3.
measure & store

em-box

formal transformation

inclusion of any stored variable as a y- or z-variable is allowed

A

1

2

logical deduction

B

1

2

3

statistical inference

logical deduction

C

1

2

3

imputation

em-box

formal transformation

D

1

2

3

statistical inference eg blow-up

display

Figure 26

[] indicates error source
somewhat closer look at the reverse, synthesizing phase. It is this part of the processing which is schematically visualized in figure 26.

Thus figure 26 shows how a typical uβγ-message, for example a statistical table, is successively built up from initial, physically stored em-boxes. According to the figure each member of the initial set of em-boxes may be transformed in many different ways before a resulting am-box is finally displayed. The total transformation depends upon what route is followed each time a choice symbol (A, B, C, D) occurs in the flow.

There are two basic loops in the flow

(a) \[ \text{em} \rightarrow \text{em}' \rightarrow \text{em}'' \rightarrow ... \], and

(b) \[ \text{am} \rightarrow \text{am}' \rightarrow \text{am}'' \rightarrow ... \]

between which transitions occur if route A2 or C3 is chosen.

An initial box could be a box with the following structure:

\[ \alpha: \quad A; \text{ time } = f(t); \]
\[ \beta: \quad \#; \]
\[ \gamma: \quad "\text{object type}"; \text{ time } = g(t); \]

A (lambda) is the "general" property, the property of being an object. \( \beta = \# \) means that we have not selected any \( \beta \)-variable for the objects in this box; which is actually the typical characteristic of an object box.\(^1\) As the classifying \( \gamma \)-variable of the box we have chosen "object type"\(^2\) which is relevant\(^3\) to all objects in the data base. Thus each time slice \( t \) of the box contains all objects which are objects at a particular time, \( f(t) \), and these objects are sorted into different e-cells of the box, depending upon which object type they belong to at \( g(t) \).

\(^1\) Thus object boxes can be regarded as degenerate special cases of fact and message boxes.

\(^2\) Cf section 2.3.2.

\(^3\) Cf section 2.3.3.
The first transformation of the initial box could be a formal transformation performed by the restriction operator defined in section 6.2.1. We shall probably restrict the range of the \( \gamma \)-variable "object type" to one particular of its values, e.g. "person" or "enterprise". Then the box could be formally transformed again, so that the selected type property becomes the \( \alpha \)-property of a new box. At the same time one could include other \( \beta \)- and \( \gamma \)-variables from the data base nucleus. Let us assume that these transformations gives us a box with the following structure:

\[
\begin{align*}
\alpha: & \quad \text{"person"; time = \ldots ;} \\
\beta: & \quad \text{"income", "occupation"; time = \ldots ;} \\
\gamma: & \quad \text{"age", "sex"; time = \ldots ;}
\end{align*}
\]

This is an \( \varepsilon \)-box containing messages about the income and occupation of persons classified according to age and sex.

So far we have circled around the loop A1-B1-A1 in figure 26 a number of times. Let us now choose route B2, "logical deduction", instead. Suppose that the data base schema\(^1\) contains a formal definition of "social class" as a deterministic\(^2\) function of "income" and "occupation". We may then transform our box into the following one after application of the "logical deduction" box operator:

\[
\begin{align*}
\alpha: & \quad \text{"person"; time = \ldots ;} \\
\beta: & \quad \text{"income", "social class"; time = \ldots ;} \\
\gamma: & \quad \text{"age", "sex"; time = \ldots ;}
\end{align*}
\]

If the data base schema had instead contained a formal definition of "social class" as a stochastic\(^2\) function of "income", saying that a person with income \( x \) belongs with probabilities\(^3\) \( p, q \), and \( (1-p-q) \) respectively to social class 1, 2, and 3, we could

\(^1\) Cf section 4.4.1.

\(^2\) We distinguish here between deterministic functions, according to which things may be deduced, and stochastic functions, according to which things may only be inferred. In the latter case there is some known or unknown degree of uncertainty introduced.

\(^3\) These probabilities may either be part of the database schema, and thus constant until they are explicitly altered, or they may, as statistical messages, be the dynamic result of another flow of box transformation processes.
have inferred, rather than deduced, a box with the structure defined above. In so doing we would have chosen route B3 in figure 26, and we would of course have introduced a more or less serious error into the box.

The next step could be another formal transformation making "social class" one of the classification variables. This would give us the box

\[
\begin{align*}
\alpha &: \ "person"; \ time = \ldots \ ; \\
\beta &: \ "income"; \ time = \ldots \ ; \\
\gamma &: \ "age", \ "sex", \ "social\ class"; \ time = \ldots \ ;
\end{align*}
\]

Let us assume that the requested box, which we are gradually synthetizing from the initial em-boxes, is the am-box corresponding to the table "average income for persons by age, sex, and social class". Then we are now ready for the transition A2 in figure 26 which takes from the em-box loop to the am-box subflow. This transformation could result in the am-box

\[
\begin{align*}
\alpha &: \ "person"; \ time = \ldots \ ; \\
\beta &: \ \Sigma_{income}, \ \#; \ time = \ldots \ ; \\
\gamma &: \ "age", \ "sex", \ "social\ class"; \ time = \ldots \ ;
\end{align*}
\]

The $\beta$-variable $\Sigma_{income}$ satisfies the conditions (a) and (b) stated for the variable $\gamma^\beta$ in section 6.3.3. Thus

(a) The relevance group of $\Sigma_{income}$ does not contain individual persons, but compound objects each of which is a set of persons. There is one such compound object for each of the cells in the box.

(b) The $\beta$-variable $\Sigma_{income}$ in the am-box is a known function of the $\beta$-variable "income" in the previous em-box. The value of $\Sigma_{income}$ for a particular compound object $o_i$ is expressible in terms of the values of "income" for the individual objects $o_i$ in this compound object:
\#\{o_i\}
(1) \sum_{i=1}^{\#\{o_i\}} \text{"income" (}o_i\text{)}

The \#-variable \# in the am-box also satisfies the conditions (a) and (b) stated for the variable \#^B in section 6.3.3. However, it is not a function of any particular \#-variable in the previous em-box, but it is simply defined as the frequency function. It counts the number of objects in each cell of the box. Thus its relevance group contains the same compound objects as the relevance group of \(\sum_{i=1}^{\#\{o_i\}} \text{"income" (}o_i\text{)}\) and the value of \# for a particular compound object \(\{o_i\}\) is expressible as

\[\sum_{i=1}^{\#\{o_i\}} 1\]

Finally we may follow the route C1-D2 in figure 26 in order to get the am-box

\(\alpha:\) "person"; time = ... ;
\(\beta:\) \(\sum_{i=1}^{\#\{o_i\}} \text{"income" (}o_i\text{)}\)
\(\gamma:\) "age", "sex", "social class"; time = ... ;

where the \#-variable \(\sum_{i=1}^{\#\{o_i\}} \text{"income" (}o_i\text{)}\) is deducible from the \#-variables \(\sum_{i=1}^{\#\{o_i\}} \text{"income" (}o_i\text{)}\) and \# in the previous box according to the formula

\[\sum_{i=1}^{\#\{o_i\}} \text{"income" (}o_i\text{)} / \#\]

The simple example above has not illustrated all possible routes in figure 26. E.g. route C3, "imputation", implies a transformation back to an em-box, where individual variable values, missing in a preceding em-box have been calculated from aggregate values of an am-box. Imputation is a special case of statistical inference. So is "blow-up", a transformation which is in a sense the inverse of sampling. Sampling is a formal transformation which reduces the object contents of an em-box without changing the structure of it as it has been defined by the \(\alpha\)-property, and the \(\beta\)- and \(\gamma\)-variables. Blow-up expands back the original object contents into an em-box of similar structure appearing later in the flow.
Box theory in the shape of figure 26 thus offers a method of analyzing in detail the different processing steps involved in the fabrication of a complex message. We do certainly not claim that the operative data base processes should in detail follow the basic scheme resulting from this kind of analysis. However, the scheme is a kind of specification of a "normal algorithm" corresponding to the request and the infological contents and infological structure of the data base. Depending upon the datalogical structure of the data base a lot of scheduling work has to be carried out by the data base system before we have an efficient, executable program. We shall return to this issue in chapter 9.
6.4.2 Quality and protection problems

If an organization organizes its information collection, storage, processing, and retrieval activities in accordance with the database philosophy, this means integration from many important points of view. For example, the mere conceptualization of such entities as "the database", or "the information flow" of an organization means an important step towards integration, even if it is only a change of attitude. However, the database philosophy also implies disintegration from at least one point of view. Without a database, internal or external, the decision-makers or decision-influencers\(^1\) feeling an information need have to arrange themselves for all the above-mentioned activities including measurement, registration, processing and presentation. This is a very costly and time-consuming way of satisfying an information need but it must be admitted that the preconditions for integrating information supply\(^2\) with information consumption\(^2\) for a particular, well-defined information need are better with this approach, i.e., when data collection is made "on demand" and not "for known and unknown future needs".

The relatively weak ties between information supply and information consumption activities in a database environment will imply certain quality problems. The information in the initial boxes to be used for the fabrication of a complex message, as discussed in the previous section, will be heterogenous as to its quality. Thus there may be different kinds\(^3\) of errors in different initial boxes, and even within the same initial box. As is shown by figure 26 there are additional error sources within the flow of box transformations leading from the initial em-boxes to the terminal em-box.

On the other hand it is usually a requirement that the database system be able to deliver some information about the quality of a requested message with the same speed as it delivers the message itself. This implies that we must

\(^1\) Cf chapter 7.
\(^2\) Cf section 4.3.
\(^3\) Cf the analysis in section 6.3.3 of the total discrepancy between a message box and a fact box into \(-\), \(-\), and \(-\)-components.
(a) store information about the quality of the original measurements in the database itself

(b) develop methods and procedures for estimating automatically and in real time the errors generated by transformations of the types which have been indicated by in figure 26

(c) develop methods and procedures for estimating automatically and in real time the combined effect of a series of derivation and inference transformations like those of figure 26.

It should not be surprising if infological box theory turned out to be a useful tool in these efforts.

Inadvertent disclosure of confidential information is another database problem, to the solution of which infological box theory seems capable of contributing. For a clever database intruder it may be possible to retrieve messages which, when seen in isolation, reveal only innocent information about object groups, but which, when they are combined and possibly processed by an external computer, may reveal confidential information about individual objects. In order to keep this danger under control it is necessary for the database system to keep a record of all extractions which have been made from the database during a certain time interval backwards in time. However, such a record would very soon grow to an intolerable size unless a very compact recording method is used. This is where infological box theory comes in. Particularly the notation introduced in section 6.2.2 may turn out to be useful. It is both compact and lends itself easily to systematic and continuous search for potential disclosures which the database filter function has to pursue. For a preliminary report on these problems see Olsson [160].

---

1 Cf section 4.4.
6.4.3 Descriptive data base interaction languages

As was said in section 5.4.2 we should expect future data base interaction languages to be more result-oriented than today's programming languages. An information consumer should be able to state his requests descriptively rather than constructively. He should be allowed to state only what information he wants, and he should not have to tell the data base system how his request should be processed. This requirement is very natural from the infological point of view according to which the information consumer should be able to view the data base as a black box of information. Unfortunately this view is not widely spread in the computing world today. The CODASYL Data Base Task Group, for example, found it unrealistic to require that it should be possible to express non-parametric information needs in a non-procedural way.¹ With a sound infological framework as a basis it seems perfectly feasible to develop non-procedural data base interaction languages permitting the expression of relatively complex information needs. Naturally we cannot expect such an effort to require less resources than the development of today's "problem-oriented" programming languages and their compilers. Anyhow, the investment seems more than justified, not least because it could considerably shorten the effective distance between the data bases and their ultimate users.

We shall try to supply some further evidence in support of the hypothesis that the development of result-oriented data base interaction languages is really feasible. We shall sketch a fragment of a particular infological model and study how one could express rather complex information requests of øßγ-type concerning this model in a result-oriented, though formal, way.²

¹ According to a statement by T W Olle as interpreted by the author. See also [74] and [75]. Parametric data base interaction languages only permit the expression of a low number of pre-defined information needs. Such languages will only make a small percentage of the data base information contents available to the interactors.

² Recall from section 5.4.2 that a data base interaction language may be characterized along several dimensions. For example, a result-oriented language may be relatively "formal", or relatively "natural language like".
The structure of the model fragment is visualized in figure 27. There are nine object types with the type properties"person", "household", "region", "country", "enterprise", "commodity", "trademark", "thing", and "deal", respectively. Some object types are hierarchically related. This has been denoted by "Bachman arrows" in figure 27. For example, each country is a compound object consisting of regions. The deals are examples of compound objects of transaction type. Each time a particular enterprise, the buyer, purchases a particular commodity, the product, from another enterprise, the seller, a deal object is born. "Volume" and "value" are examples of variables which are relevant to the deal objects.

The model fragment contains several object relations. For instance, there is an "OWN" relation between persons and things, and a "LOCATION" relation between enterprises and regions. There are three object relations which hold between objects of the same type, "person", namely "MOTHER", "FATHER", and "MARRIED".

Naturally there are a lot of variables and other attributes. However, figure 27 is only meant to show the general structure of the object system, and no attributes have therefore been visualized in it. This could have been done in separate figures for each object type. We shall only mention that "occupation", "age", "sex", "social class", and "marital status" are examples of variables which are relevant to persons. The "volume" and "value" variables which are relevant to deals have been mentioned already. Naturally, there is also an indentifying variable for each object type. As we shall see in the examples below, there are also a lot of meaningful relational attributes which are derivable from the entities which are explicitly specified.

1 Cf section 2.3.2.
2 Cf Bachman [201] and section 3.2.2.
3 Cf section 2.2.1.
4 Cf the illustration technique described and used in section 5.4.1.
5 Cf section 2.3.3.
6 Cf section 2.2.2 example 5. A relational attribute is an attribute which consists of relational properties. Cf also section 2.3.3.
Figure 27 Visualization of the structure of a fragment of a particular infological model
Now we shall study four examples of information requests referring to the particular infological model which we have just outlined. The requests are all of the general αβγ-type discussed earlier in this chapter. We shall express them in a result-oriented way by means of a technique which is essentially based upon the infological box theory introduced in previous sections.
Example 1

We want a table showing for 1970 the average income for all persons by age, sex, and social class.

Rows (1) and (3) in figure 28 define the appropriate object box corresponding to this request. Row (2) defines the e-message contents of the box, and row (4) describes the demanded output box with the notation introduced in section 6.2.2.

Request (4) will only give us averages in the e-cells of the output box. Row (5) shows an alternative request, which would give us in addition to the kernel table, the figures showing the average income in the person population as a whole and in each of the social classes.

Request (6) would result in an output transaction showing the accumulated income in each e-cell of the box kernel and, moreover, the grand total and every subtotal of every order.

We have assumed that µ and Σ are functions which are automatically recognized by the data base system. This is a prerequisite for the strictly result-oriented, or descriptive, expression mode. Suppose instead that only summation (Σ) and count (#) had previously been defined to the system. Then the requestor should have to use the less result-oriented, more constructive, or procedural, expression mode (7). Similarly more complex functions may be defined to the data base system, either once and for all, or temporarily as need arises.

Remark. The ordering of the components of the tuple arguments of the PRESENT clauses reflects the ordering of the variables in the declaration in row (3). This convention will be used in the subsequent examples, too.
\[ \alpha: \quad \text{person; time} = 1970; \]  
\[ \beta: \quad \text{income; time} = 1970; \]  
\[ \gamma: \quad \text{age, sex, social class; time} = 1970; \]  
\[ \text{PRESENT } u(\text{income}) | < -, -, -> \]  
\[ \text{PRESENT } u(\text{income}) | \{ < -, -, ->, < \ast, \ast, \ast > \} \]  
\[ \text{PRESENT } \Sigma(\text{income}) | < \ast, \ast, \ast > \]  
\[ \text{PRESENT } u(\text{income}) = \Sigma(\text{income}) / \# \{ < -, -, ->, < \ast, \ast, \ast > \} \]  

\textit{Figure 28}
Example 2

We want information about Guatemala's trade, export and import by commodity and trade partner, during the years of 1960, 1965, and 1970.

Rows (1), and (3) - (6) in figure 29 define the object box corresponding to this request. It is a time series box containing trade transactions, called deals within our model, sorted by

(a) the country in which the buying enterprise is located; row (3)

(b) the country in which the selling enterprise is located; row (4)

(c) the commodity involved in the trade transaction; row (5)

(d) the year in which the transaction took place; row (6)

In rows (3) and (4) we have used a particular technique for referring to generated, relational attributes. The basic rule of this technique is the following. If $O_1$ is an object group, generated by a group property which is referred to by "gprop1", and $O_2$ is another object group, generated by a group property which is referred to by "gprop2", and "var" refers to a variable which is relevant to $O_2$, and "REL" refers to an object relation with the relevance groups $O_1$ and $O_2$, then

"gprop2 (REL).var"

refers to a relational attribute which is relevant to $O_1$. We may apply the rule repeatedly thereby generating names of

1 Cf sections 2.3.2 and 3.1.
2 Cf section 2.3.3.
3 Cf section 2.3.4.
\[ \alpha: \text{deal}; \]  
\[ \beta: \text{volume, value}; \]  
\[ \gamma: \text{enterprise(BUYER).region.country, } \]  
\[ \text{enterprise(SELLER).region.country, } \]  
\[ \text{commodity, } \]  
\[ \text{time (year)}; \]  
\[ \text{PRESENT } \sum_{\text{volume, value}} | \]  
\[ \{ \langle \bullet \text{ except Guatemala, } \]  
\[ \text{Guatemala, } \]  
\[ \bullet \} \} \]  
\[ \{1960, 1965, 1970\} \} ; \]  
\[ \langle \text{Guatemala, } \]  
\[ \bullet \text{ except Guatemala, } \]  
\[ \bullet \} \]  
\[ \{1960, 1965, 1970\} \} ; \]  

**Figure 29**
relational attributes involving a chain of object relations. For example, the relational \( \gamma \)-variable referred to in row (3) of figure 29 is tied to the "deal" object group via the chain \texttt{BUYER-LOCATION-c}, where "c" denotes the relation between regions and countries. With strict adherence to the rule stated above this relational \( \gamma \)-variable should be called \texttt{"enterprise\{BUYER\}.region\{LOCATION\}.country\{c\}.id"}

where "id" is assumed to denote the identifying attribute of the "country" object group. However, we have applied two simplifying rules in (3). Firstly, when there is only one object relation which directly relates one object group to another, we may exclude the "(REL)" part of the generated reference without risk of ambiguity. For example, \texttt{LOCATION} is the only object relation relating enterprises to regions. Secondly, any final sub-reference ".id", where "id" refers to an identifying attribute, is left out.

Row (2) in figure 29 defines the em-contents of the box, and rows (7) - (15) describes the em-contents of the demanded output transaction. In the \texttt{PRESENT} case we have introduced a couple of extensions to the notation technique introduced in section 6.2.2. For example, if a variable can take the values \( a, b, c, \) and \( d \), and no others, we shall write

" \( e \) except \( c \)"

instead of

"\( a \ e \ b \ e \ d \)"

The meaning of the latter expression was defined in 6.2.2, formula (30). This convention has been used in rows (8) and (13) in figure 29, where we eliminate the home trade in Guatemala from the output box. Finally, in rows (11) and (15) we have adopted the convention of writing

"\( \{x, y, z\} \)"

instead of

"\( x \lor y \lor z \)".
Example 3

We want to know the absolute and relative frequencies of car-owning employees in car-producing companies who are loyal in the sense that they have a car produced by the company in which they are employed.

Rows (1) and (3) in figure 30 define the one-dimensional object box corresponding to this request. As only frequencies are asked for there are no e-message contents in the box, row (2) is empty. According to row (4) we want the output message to tell beside the total number of car-owning employees in car-producing enterprises only the number and percentage of loyal dittos, i.e. the number and percentage of car-owning employees in car-producing enterprises for whom the variable "loyalty" takes the value "yes".

Now it happens that we have used references in our description of the information we want which have not earlier been defined to the data base system, namely "carowner", "carworker", and "loyalty". This is perfectly legitimate and should not cause a wild outburst from the system. Instead the latter asks the interactor to define the new entities. See row (5) in figure 30. Note that the system has "understood" that only one value, "yes", of the "loyalty" variable needs to be defined.

In the definitions in rows (6) - (9) we have introduced the well-known "∃" symbol in order to eliminate the ambiguity which would otherwise be there, because some of the object relations implicitly involved in the generating expressions are not functions. For example, if an employee in a car-producing enterprise owns several cars, he should be counted as loyal if at least one of them has been produced by "his" company.

In rows (8) - (9) the expression mode is rather similar to traditional predicate calculus. It should be observed that one of the simplification rules stated in example 2 has been slightly generalized so that we write
\( \alpha: \) person and carowner and carworker; \( (1) \)

\( \beta: \) \( (2) \)

\( \gamma: \) loyalty; \( (3) \)

PRESENT \( \#: \% | \langle \{ \text{yes, } x \} \rangle \); \( (4) \)

\( \rightarrow DEFINE \) carowner, carworker, loyalty = yes; \( (5) \)

carowner: \( \exists \) thing.trademark.commodity = car; \( (6) \)

carworker: \( \exists \) enterprise.trademark.commodity = car; \( (7) \)

loyalty = yes: \( \exists x \) (enterprise.trademark = x and thing.trademark = x) \( (8) \)

\( (9) \)

**Figure 30**
"trademark = x"

instead of

"trademark.id = x"

where "id" is assumed to refer to the identifying attribute of the "trademark" objects.
Example 4

A "pendulator" is defined as a person living at one location and working somewhere else. We want to know something about the "pendulation pattern" for the regions a, b, and c. For example, we want to see if there are any differences between occupations, sexes, and marital statuses.

Rows (1), (3), and (4) in figure 31 define a five-dimensional object box which seems to be relevant to the somewhat vague request above. Once again we are only interested in frequencies, which explains the empty β-row, row (2), in figure 31. The output message is precisely defined in the PRESENT clause; rows (5) - (7). We have introduced there the symbol "other" to denote the same thing as "x except a, b, c". Thus we shall not only learn how many persons pendulate between the regions a, b, and c, but we shall also get, for example, the total number of persons pendulating from region b to some other region than a and c, as well as the total number of persons, pendulating from some other region than a and c into region b.

In row (9) we have defined the previously undefined property "active", which is part of the α-property in the αβγ-query; row (1). With the conventions introduced earlier it would have been sufficient to define "active" by "∃ enterprise" because EMPLOY is the only object relation which directly relates persons to enterprises; cf figure 27.

Remark. There is one ambiguity in example 4 which has not been eliminated by the formulation in figure 31. Suppose that a person is employed by two enterprises, located in different regions. Then the relational attribute enterprise.region would not fulfill the condition for being a variable, and it would not be clear into which e-cell of the object box that the double-working person should be sorted.

1 Cf example 2.
\( \alpha: \text{person and active}; \)  

\( \beta: \)  

\( \gamma: \text{occupation, sex, marital status, region, enterprise.region}; \)  

PRESENT \# |<*, *, *,
{a, b, c, other},
{a, b, c, other}> \rightarrow \text{DEFINE active;}

active: \exists \text{enterprise (EMPLOY)};
Conclusions

The examples 1-4 provide a sketch of a data base interaction language by means of which very complex and differing information requests may be expressed in a formal, result-oriented way. Hopefully, the sketch has shown that further research and development work in this area would really be worthwhile.

In the discussion of the examples we have tacitly assumed that the user formulates his information request in close interaction and co-operation with the data base. Note that this does not imply that the user has to get the actual answer to his information request within a couple of seconds. Even if the actual retrieval process will take hours or days it may be very efficient for the interactor to get some very fast feedback during the request formulation phase. Typically the interactor will only have partial knowledge about the infological model underlying the data base and about the language he is using for interaction with the base. The data base system should then be so designed as to assist the interactor constructively and in such a way that it is reasonably certain that the interactor and the data base have identical opinions of the meaning of the information request when the actual retrieval process is started.

Suppose, for example, that the interactor hardly knows anything about the particular infological model underlying the data base he is going to interact with. Then the data base system could serve him a list of the basic object types. The interactor would then identify a few of these object types as possibly relevant to his information need. The next step from the data base system would be to display a network like that in figure 27, restricted to the object types selected by the interactor. The network would only show object groups and object relations. If the network is very small the attributes which are relevant to the respective object groups could perhaps be visualized in the same picture. Otherwise the interactor should be able to demand separate pictures showing the relevant attributes for one of the object groups at a time.
If the user is not sure about how an entity is defined he should be able to get a more or less detailed definition displayed to him.

When the interactor has defined what he believes to be the appropriate object, box and PRESENT clause, he should immediate get feedback from the data base system showing what kind of output, e.g. what kind of table, he will get when the request has been processed. Then the interactor could make his final adjustments, changing the layout of the presentation format, for instance.

The designers of future data base interaction languages certainly have enormous possibilities of demonstrating how powerful and yet "human" automatic data processing could be.
7. Data base systemeering

7.1 Systems and systemeering

7.1.1 Introduction

On several occasions in the previous chapters we have talked about the data base as "a system", and we have used the term "systemeering" to denote the analytical and creative task of designing (data base) systems. However, we have not said much about what a system actually is, and what kinds of activities are inherent in the systemeering task. We shall devote part of this chapter to discuss these important issues. Using the results of this discussion we shall then proceed to treat the more specific problem of designing a data base. We propose a set of general goals, which we believe to be well worth careful analysis in all data base projects. We also make an attempt to find a subsystem structure of a data base which could be used as a first functional description of the data base in many practical design situations. Finally we discuss the problem of translating the overall data base goals into more specific goals of each data base subsystem.
7.1.2 Different approaches to systems

Although "systems theory", "systems thinking" and "systems approaches" to different phenomena have been presented by many authors, there is not at present a unanimous approach to systems. This is probably due to the fact that "the systems approach" is much more a philosophy, a general attitude towards complex problems, than it is a formal method. Naturally, then, different authors emphasize different aspects of the system concept, and the results and recommendations arrived at more often point to fallacies to be avoided than to programs of action.

Most system thinkers seem to accept the view that a system is a set of parts, or components, which are related to each other in some way, and which may themselves be systems, as well as the original system will normally be a part of another system, the supersystem.

One may focus the interest upon different parts of the rather vague definition just stated. For instance, one may emphasize the existence of a supersystem and warn against sub-optimization which may easily occur if one disregards the other systems which are parts of the same supersystem and inevitably related to the system under consideration. According to this line of thinking it is most important of all to study "the system as a whole", the totality rather than its parts. The obvious drawback of this approach, if strictly adhered to, is, of course, that it will be very difficult to get anything done at all: the supersystem will turn out to be a subsystem of a super-supersystem, etc; the complexity of our design problem will increase rather than decrease.

Other system thinkers, like Langefors [46], stress instead the former part of the definition stated above, i.e. the view of a system as a set of subsystems. They point out that most systems to be designed are originally imperceivable and have to be broken down into a set of perceivable subsystems if we are to be able to carry out the design and construction work
in an organized way, and if we want to be reasonably confident
that we shall end up with a system which satisfies our design
goals.

Needless to say, the two lines of thinking introduced here
are not at all incompatible with each other, but they stress
two sides of the same coin. When we set the goals of the
system to be designed we have to consider the effects upon
the supersystem. However, we cannot expect these considerations
to take place in a strictly formalized way but rather by means
of negotiations between goal-setters representing the interests
of the fellow systems and the supersystem. On the other hand
we can usually neither define a precise goal structure, nor
design and construct the desired system until we have broken
down the original problem into manageable subproblems.
7.1.3  A fundamental principle of systems work

In order to get a general idea of the character of the system engineering task we shall review here "the fundamental principle of systems work" established by Langefors [48]. The principle, which, according to Langefors, is valid for all analysis, design, and management of systems, reads as follows:

"Partition the systems work into separate tasks, a through d,

a. Definition of the system as a set of parts.
List all parts from which the system is regarded as built-up.

b. Definition of system structure.
Define all interconnections which make up the system by joining its parts together.

c. Definition of the systems parts.
For each single part (or group of similar parts) separately define its properties as required by the system work at hand and do this in a format as specified by the way the system structure is defined (in task b).

d. Determination of the properties of the system.
Use the definitions as produced by the tasks a, b, and all separate tasks c, all taken together. Compare with specifications wanted for the system and repeat a, b, c, and d until satisfied."

The fundamental principle has several important implications, only a few of which will be commented upon here.

The principle is both recursive and iterative. It is recursive in the sense that in exactly the same way as the original system is broken down into a set of subsystems and a set of interconnections, each subsystem will be further broken down. We will end up with a hierarchy of systems, containing the original imperceivable system at the top, and a set of
perceivable systems to be constructed at the bottom\textsuperscript{1}.

The iterativity of each recursion of the fundamental principle is clearly seen from task (d) and is due to the fact that there is not at present any formal algorithm for breaking down a system into a set of subsystems and a set of interconnections. Even if there were such an algorithm we could like to execute it several times in order to find "the best" subsystem structure, e.g. the structure which minimizes the set of interconnections.

For a long time ahead of us we can at best hope for formal methods which assist us, but which do not solve all the problems for us, when we try to find the subsystem structure of a system. The same is true for task (c), i.e. the task of transforming the goals of the system into goals for each of the subsystems\textsuperscript{2}.

Nothing is said explicitly in the fundamental principle, as formulated above, about the character of the goals, or required properties, of the system and its subsystems. Elsewhere, however, Langefors distinguishes between external and internal properties of a system. It seems justified to require that the goals of a system should be expressed in terms of external properties of the system, i.e. properties which are meaningful to a person viewing the system as a black box.

\textbf{In this connection we want to stress that}\textsuperscript{3}:

\textsuperscript{1} Note that the same subsystem may occur repeatedly in the hierarchy, and not only on the same level. This suggests that tasks (a) through (d) should in each recursion be followed by a task (e) where one explicitly investigates whether any subsystems defined in the current recursion are identical or have been identified in earlier recursions. Failure to recognize identical subsystems will cause (i) repeated breakdown of the same system (not necessarily with the same result), and (ii) repeated construction of the same system; in a complex system the costs incurred because of this kind of double work may be considerable.

\textsuperscript{2} The goals of the (original) system are not explicitly mentioned in the quotation above.

\textsuperscript{3} Keep an eye on figure 1 during the rest of this section.
Figure 1  An example of a goal hierarchy and a system hierarchy which are not isomorphic. A few remarks:

(i) The same goal, 2, and its subgoals, 4, 5, and 6, appear twice in the goal structure, which is thus a network rather than a tree.

(ii) Similarly the same system, D, and its subsystems appear twice in the system structure.

(iii) The same system may contribute to several goals, and the same goal may be contributed to by several systems as exemplified by the dotted lines.

(iv) Goals which are contributed to by systems which are related because they share scarce resources (cf section 7.1.4) are potentially conflicting goals.
(1) the task of breaking down a goal into subgoals

(2) the task of breaking down a system into subsystems

(3) the task of transforming the goals of a system into goals of a subsystem

are three different, though related, systemeering tasks. Ideally, of course, we should like the goal–subgoal hierarchy and the system–subsystem hierarchy to be isomorphic in the sense that there would be an equal number of nodes and levels in both hierarchies, and an arbitrary system on an arbitrary level in the system hierarchy would contribute to one and only one goal on the same level in the goal hierarchy. Unfortunately, such isomorphisms seem to be very rare in real life. When proposed they should be questioned.

Another complication should also be noted. The bottom level in the goal hierarchy will contain the most precise, operational goals. Ideally these goals should be available when we start the system breakdown, because otherwise we cannot define the external properties of the system and subsystems precisely and operationally. However, it is neither feasible nor advisable to require that the setting and spelling out of the goals should be completed before design begins. The goal-setting and design processes are highly interrelated and should stimulate each other. This will be further discussed in subsequent sections of this chapter. Thus, not only will each step in the system breakdown have to be iterated as described by (d) in the quoted formulation above of the fundamental principle of systems work, but the whole breakdown will have to be reviewed as the preliminary, crude goals are replaced with more operational specifications.
7.1.1 The system and its environment

What we stated about the system concept in section 7.1.2 was in essence that

(a) on the one hand a system is a set of related subsystems, and

(b) on the other hand a system is a subset of a super-system

As we have hinted at in the two previous sections, many important conclusions may be drawn only by considering these two aspects of the system concept. However, both (a) and (b) are recursive, and in practical situations we have to limit the endless hierarchy both upwards and downwards, and then we shall need other than recursive descriptions, at least of the initial and terminal systems. This is the point of departure for our complementary discussion in this section of the nature of a system and its environment.

A general description of a system which seems to be useful in many practical design situations\(^1\) is the following (cf figure 2):

We divide the parts of a system into two general categories, resources and processes (activities). Both "resource" and "process" have the character of fundamental concepts (cf 2.2) which we leave formally undefined. However, we shall informally comment upon the two concepts and the interconnections between them, in an attempt to guide the systemeer to determine, for

\(^1\) The description should look familiar to designers of data processing systems. In its general interpretation, however, it is equally applicable, for instance, to the design situation which a government faces, when it is about to produce a political program for the development of a nation.
Figure 2  The system as a set of related resources and processes

Legend:  

- process  - resource  

- use of resource  

- exchange of stimuli (initiation etc)  

----- joins two states of the same resource
a particular system, which parts are to be considered as resources, and which as processes.\footnote{Recalling the fundamental principle of systems work (section 7.1.3), we thus assume that the systemeer has performed task (a) and is about to start with task (b). Even in his search during (a) he may be better off, knowing that he is looking for "resources" and "processes" rather than just some kind of enigmatic "parts". Naturally, there may be systems, which do not easily lend themselves to descriptions in terms of resources and processes as depicted in figure 2. Then the systemeer should abandon this imposed pattern and use the more general, but vaguer, formulation of task (b), as quoted in 7.1.3.}

A resource may be any kind of asset, physical or abstract, monetary or intangible, etc. People, disk storage, health, computer programs, knowledge, leisure, CPU-time, and energy could serve as examples of resources from different kinds of systems.

A process is an abstract entity producing change by transforming a set of resources, the input resources, into another set of resources, the output resources, and by initiating, suspending, activating, and terminating other processes.

A resource may be more or less flexible depending upon (i) the range of other resources into which it may be transformed by means of internal or external processes\footnote{An internal process is a process which is a part of the system under consideration. An external process is a process which is a part of the environment, of the considered system.} and (ii) the "easiness"\footnote{This "easiness" is related to the inverse of the drain of other resources caused by the transformation. For instance, a computer may be transformed into money, but not very "easily".} with which the transformation takes place. Money and general knowledge are examples of flexible resources, whereas e.g. computers and special skills are more inflexible.

The relationship between a process and a resource may be of several kinds; for instance a process may

- use a resource exclusively for a certain period of time and then release it

- share a resource with other processes
- return a resource in its original state

- return a resource in a different state

- create a resource

- destroy a resource

Each process of a system should contribute to at least one system goal. If the same resource is a direct or indirect precedent of two different processes with different goals, then the goals of the two processes are potentially conflicting goals. For instance, if in figure 2 process \( P_2 \) contributes to goal \( G_i \), and process \( P_5 \) contributes to goal \( G_j \), then there is risk of conflict between \( G_i \) and \( G_j \), because the same resource \( R_3 \) is a precedent of both \( P_2 \) and \( P_5 \).

On the other hand, if a succedent resource of one process is a precedent resource of another process, then the goals of the two processes may reinforce each other. For instance, in figure 2 the goals of \( P_4 \) may reinforce the goals of \( P_3 \) and \( P_5 \).

The environment of a system consists of those parts of the supersystem which are external to the considered system, i.e., the "fellow" systems of the considered system. For instance, in figure 1 the environment of system \( J \) consists of systems \( I \) and \( D \).

If it is feasible to regard a particular system as a set of related resources and processes, instead of regarding it as a set related but undifferentiated parts, we gain the advantage that when we perform task (a) and (b) according to the fundamental principle of systems work (cf. 7.1.3), we will not get just any structure, but a structure which in a natural way lends itself to further subdivision. This is so, because the processes partition the system in a very natural way into subsystems. If we accept this partitioning rule, the processes become the systems on the next lower level of the breakdown, and conversely the partitioned system becomes a single process on the next higher level. Naturally, the subdivision of a process

\footnote{Cf. 7.1.3, in particular figure 1.}
will normally have to be accompanied by a finer classification of the resources used and produced by the process, as well as of a finer subdivision of the goals of the system (cf 7.1.3).

As has already been pointed out the suggested resource/process view of a system is in no way original; it is well-known to data processing people. The interesting thing is the seemingly general applicability of the structure. For instance, just recently sociologists have proposed a model of society in similar terms.

Another thing, which is interesting from the point of view taken in this report, is that the resource/process approach, which we have found useful when the task is that of breaking down a system, for whatever purpose, seems to be perfectly compatible with the info logical approach recommended here, when the task is the particular one of designing a data base for the system. Thus, if an object system has been broken down in terms of resources and processes, it is feasible to transform the resulting model of the object system into a particular info logical model of the object system, i.e. a model in terms of objects, properties, attributes, etc. According to the author's experience it is often practical to regard a particular resource as the value of an attribute for an object, whereas a resource category may be specified as an e-constella tion type, i.e. an \langle object group, attribute \rangle pair (cf 2.3.4).

Processes may be specified in terms of object relations (between the objects to which the input and output resources are tied), compound objects of "event type" (cf 2.2.1), or properties (of the objects to which the input and output resources are tied), depending somewhat on the purpose of the data base.

---

1 See, for instance, Holmberg [25]: "I stället för slutrapport", Tiden 1972:1.
7.1.5 Systemeering techniques

As was said in section 7.1.3 there is not at present, and probably will not ever be, any formal systemeering technique, which solves all the problems of the systemeer and his principals. However, there exist partial formal techniques which have turned out to be of great help to the systemeer in his complex task. For instance, there is the precedence analysis technique for finding, describing, and manipulating the structure of an information system\(^1\). There are also semi-formal techniques for file consolidation and process grouping\(^2\) supporting the transformation of an (abstract) information system into a (realized) data processing system.

Naturally, all design tasks that we want to computerize have to be formalized. As we shall see later in this report, computerized design is not only desirable but simply necessary for certain functions of an advanced data base system, which has to be able to modify itself automatically and on its own initiative. We shall have to develop new formal techniques as well as modify the existing ones for these partially new purposes. A few attempts in that direction are made in chapters 8 and 9.

However, we cannot escape that there is a truly creative element in every systemeering task worth its name. Nobody will probably be able to tell us exactly how to find the "natural" or "best" subsystem structure of a given system. Therefore, a systemeer should be very careful before he says something like "if you just define the goals precisely, I'll design the optimal system for you", because suppose some day his principal would really turn up with a set of precise goals\(^3\), then still

\(^1\) Cf Langefors\(^{[48]}\) and Lundberg\(^{[52]}\).

\(^2\) Cf Langefors\(^{[48]}\), Arrás\(^{[112]}\), Sundgren\(^{[138]}\), and Nunemaker\(^{[136]}\).

\(^3\) His principal is very unlikely to this, of course, and the systemeer knows this or should know it. If the goal-setting and design processes are viewed together, it would probably be "executively suboptimal" to complete goal-setting before design is started. As was stated in 7.1.3 the tasks of (i) breaking down the goals into subgoals, (ii) breaking down the system into subsystems, and (iii) transforming the goals of the system into goals of the subsystem, probably have to be carried out in parallel. Cf also 7.2.
the systemeer would find that there is no unique way of partitioning the system, transforming the precise system goals into precise goals of the subsystems, designing the subsystems, etc.

It is sometimes claimed that a system, e.g., an organization, should be broken down into subsystems in a way which makes any subsystem contribute to one and only one goal. Certainly such a principle would make life a lot easier for the systemeer and for those who are to evaluate subsystem performance. It has not been proved, however, that this principle would be optimal from an overall point of view, or that it is at all feasible.

Another fallacy is to believe that goal-setters and systemeers should ideally start their work in a perfectly open-minded mood. Again this would not be "executively optimal". It seems reasonable that the goal-setter should put attention to goal structures which have been relevant in similar situations, and that the systemeer should not always start from scratch in the sense that no presumptions about at least the general structure of the system to be designed. For instance, a database systemeer, like an aeroplane design engineer, will and should be equipped with some prejudice about the general appearance and functional parts of the system he is planning for. It is the task of research and education to supply him with "good prejudice", i.e., good models to start his work from and to use in the communication with goal setters and decision makers. As a matter of fact this report is the result of an attempt to develop such prejudice.

One final remark should be made before we leave this section. We have tacitly assumed that the systemeer of a system will finally come up with one system structure, and that this structure will be adequate not only for the construction of the system, but also for operative management purposes. Naturally this is the most attractive solution for many reasons, but it should be pointed out that, at least theoretically, there is the possibility that one partitioning of the system is

\[1\] "Executive" versus "modal" optimization is discussed by Langefors in [26] "System för företagsstyrning".
most adequate during the construction phase, whereas as quite another structuring is preferrable from operative management point of view. In some such situations it may perhaps be feasible to use different descriptions of the same physical system for the different purposes.
7.2 Systemeering and decision-making

Even if we have found a feasible subsystem structure of a particular system, for instance a data base, it is not certain that we have uniquely determined a solution to the design problem. There may be several feasible designs, among which we have to choose one for construction, implementation, and operation. We have then a decision situation, the general characteristics of which will be discussed in this section.

We adopt a simple model of the decision process. According to this model, every imaginable decision in a given situation may be described as a particular assignment of values to a set of variables. Some of these variables may be influenced or completely controlled by the decision-maker; they are called controllable variables or decision variables. Others, the uncontrollable variables, are out of the decision-maker's control; they may be in the hands of competitors, "nature", or overall-policy makers, for instance. The actual values of the uncontrollable variables may be known or unknown to the decision-maker. A set of value assignments to the controllable variables is called a (possible) course of action. During the decision process one course of action, ideally the best one, is selected from the set of all possible courses of action. Which is the best course of action is determined by the explicit and implicit goals governing the decision process.

The basic decision model is visualized by figure 3. The decision process uses information about the goals, the uncontrollable and the controllable variables, and it produces a decision, a normative set of information. Thus the decision process is an information process using and producing information resources. The dotted arrows in figure 3 have been introduced in order to indicate that we assume the decision-maker's knowledge about the uncontrollable and controllable variables to be growing all the time. For instance, he may prognosticate the value of certain uncontrollable variables, whereas the

\[\text{1 Some version of this model can be found in many texts on decision-making. See for instance Miller and Starr [30], chapter 4, and Langefors [26], chapter 3.}\]
Figure 3: The decision process
values of the controllable variables may be successively determined by means of partial decisions.

**Remark.** In connection with partial decision-making we can illustrate the potential benefits of adhering to the fundamental systemeering principle quoted in 7.1.3. Suppose that a particular systemeering task involves $n$ controllable variables, each of which may be set to $m$ different values. If we regard the design of the whole system as one indivisible decision, we shall have to survey $m^n$ possible courses of action. If instead we manage to partition the system into $p$ subsystems each of which involves only $n/p$ controllable variables, we replace the total design decision with $p$ partial design decisions, for each of which we only have to survey $m^{n/p}$ courses of action. For large $m$ the reduction of the number of courses of action from $m^n$ to $m^{n/p}$ will probably more than outweigh the breakdown efforts.\(^1\)

In order to be able to evaluate design proposals easily, one should naturally prefer system performance and all system goals to be expressible in terms of one and the same quantitative measure. However, most practical design situations, they may concern data bases or other systems, seem to be irreducible multi-goal situations, i.e. situations where it is impossible, "ex ante" to transform all components of the goal structure into a common utility or cost measure. At best an irreducible multi-goal structure may contain partial comparison criteria, which enable the designer to eliminate alternatives without having to involve the decision-maker. When there is no explicit, formal comparison criterion automatically determining the choice, the goal-setters have to make an intuitive decision. Such decisions may, "ex post", impose certain weights upon the different goal components.\(^2\)

Suppose that an irreducible goal structure contains $r$ independent goals, i.e. $r$ goals which are not expressible in terms of each

\(^1\) Cf. Ashby [21], chapter 13.

\(^2\) If the goal setting and design process is going to be active for a long period of time, it may improve the quality of decision making if the decision makers are supported by a mechanism, which records intuitive decisions as well as formal decision criteria and searches for the implicit goal structure. At least such a mechanism will pin-point inconsistencies.
other. Then at least \( r - 1 \) of the goals are not expressible in monetary terms, i.e. the goal structure contains at least \( r - 1 \) intangible goals\(^1\), or intangibles. It may be possible in a particular design situation to establish upper and lower monetary boundaries, but it will not be possible to establish cost transformation rates, prices, for intangible goods and evils.

The intangibility of a goal may be more or less intrinsic. Sometimes it may be possible in principle to develop a procedure for measuring the dollar value of a marginal contribution to a particular goal; in practice, however, it could very well turn out that the cost of measuring would exceed the value or cost to be measured. Thus even for such goals we have to accept their intangible character and develop other methods than conventional cost/benefit-analysis to deal with them.

One consequence of the multi-goal situation is that we had better regard design in terms of satisficing rather than optimizing. Satisficing is a principle of control, which maintains that the aim of a decision-maker in a decision situation is not to obtain the greatest possible value of his utility function but to obtain a certain value which he considers as satisfactory. Barnard, Simon, and others have found that the satisficing principle gives the best explanation of observed executive behaviour, and Lungefors [27] has put forward theoretical arguments in support of the satisficing principle in administrative decision situations.

As we cannot maximize or minimize more than one variable at a time we would anyhow be forced to apply the satisficing principle in order to handle \( r - 1 \) of the \( r \) independent goal dimensions in a multi-goal situation. This is what is done in the ordinary optimization models used in economics and engineering, where one variable, e.g. utility or cost, is maximized or minimized at the same time as a number of other variables are kept within certain boundaries.

\(^1\) Cf Mc Kean [29].
Using the satisficing principle for all components of the goal structure is efficient in a design situation, as it helps us to direct our efforts to areas where they are best needed. By using optimization models we risk spending a lot of design resources in order to achieve marginal improvements of the value of the target variable, when it would be more efficient to shift one or more of the constraint variables from one satisfaction level to another.

In trying to direct executive resources at every stage of the decision process in the most urgent direction, control by satisficing has much in common with management by exception, a well-known administrative principle. The satisficing principle is also related to the so-called theory of the second best in public finance.¹

Now we leave the discussion of the general characteristics of design goals and turn our attention to the interrelations between goal-setting, design and decision-making. Figure 4 is a more elaborate version of figure 3, stressing the iterative character of systemeering as well as the mutual influence which should exist between the design, goal-setting, and system demarcation processes. The latter is a process which determines the boundary between the system to be designed and its environment (cf 7.1.4). Accordingly this process determines which of all "variables" or "conditions" are given by the environment (the uncontrollable conditions) and which are controllable from the designer's point of view. The system demarcation process is governed partially by "higher-level" goal-setting like "policy-making", and partially by conditions which are truly uncontrollable in the sense that they are hardly under any one particular decision-maker's control, for instance "available methods", "current technique", or "the distribution of arrival times and interaction lengths for terminal users of a data base".

Before the goal-setting process starts there should be (or perhaps has to be) a truly intuitive, more or less irrational, brainstorming-like process generating desirable goals (desirables for short). The desirables may very well be logically inconsistent

¹ Musgrave [3].
or at least inconsistent with regard to the uncontrollable variables. It is the task of the goal-setting process to reconcile the contradictory desirables into a consistent structure of goals and comparison criteria. The goal structure should be feasible, which means that it should be believed to be possible by mere variation of design variable\(^1\) assignments to generate during the design process a non-empty set of design proposals satisfying the goals and the uncontrollable conditions.

From the point of view of the design process the uncontrollable variables and the goals are similar in the sense that they both act as constraints and implicitly or explicitly rule out a vast number of "imaginable" designs as infeasible. Very often, however, these constraints will not be sufficient to determine a unique solution to the design problem. In general\(^2\) they may be considered to partition the set \(S\) of all "imaginable" designs into four mutually exclusive subsets, \(S_1, S_2, S_3,\) and \(S_4,\) where

\[
S_1 = \text{the set of designs not satisfying the uncontrollable conditions}
\]

\[
S_2 = \text{the set of designs not satisfying the goals}
\]

\[
S_3 = \text{the set of feasible designs each of which is inferior to at least one other feasible design according to the formal comparison criteria, which are part of the goal structure}
\]

\[
S_4 = \text{the set of superior designs, i.e. feasible designs, which have no competitor that is better according to the formal comparison criteria; it is the task of the design process to state explicitly the contents of } S_4
\]

If \(S_4\) is empty, then \(S_3\) is empty, too, and the design problem has no solution; the goals have to be revised or the project dropped.

\(^1\) "Design variable" = "controllable variable".

\(^2\) Cf figure 5.
Figure 5  Subsets of the set $S$ of "imaginable" designs
If $S_h$ contains exactly one design, this design should be realized, unless the goals are revised.

If $S_h$ contains more than one design, the goal-setting process has to be iterated once more in order to sharpen the formal comparison criteria or make an intuitive choice among the alternatives.

Failure to recognize, that revision of goals and even of system boundaries (the leftmost dotted arrows in figure 4) should be regarded as perfectly legitimate in systemeering projects of some complexity, tends to cause a lot of hard feelings in practice. However, not only should the systemeer be prepared to such revisions; sometimes he may very well be in the best position to propose them.
7.3 How to systemeer a data base

In 7.1 and 7.2 we have discussed systemengineering in general. Now we turn our attention to a particular kind of system, the data base, in order to see if and how the general systemengineering principles are applicable, and first of all, naturally, in order to learn something about how to attack data base design projects.

We may learn from 7.1.3 that in order to design a data base system we have to carry out, in parallel,

\(1\) the task of breaking down the goals of the data base, thus defining a goal structure

\(2\) the task of breaking down the data base into sub-systems, thus defining what we shall call a functional structure of the data base

\(3\) the task of transforming the goals on a higher functional level into equivalent goals on a lower functional level

Now, since a data base is a representative system, i.e. a system representing another system\(^1\), we have to consider the structure of the represented system, the object system, as well. According to the general thesis, propagated for throughout this report, the structure of the object system of a data base should be specified in terms of a particular infological model. Thus in connection with data base systemengineering we have to add a fourth task,

\(4\) the task of specifying a particular infological model\(^2\),

and naturally this task has to be related to and coordinated with the other tasks. Just as we should not expect or even desire the goal structure to be precisely defined before we set

\(^1\) Cf the definition of data in chapter 1.
\(^2\) Cf chapters 2 and 3.
out to find a feasible functional structure, we should not
either expect or desire a detailed infological specification
to be available when the goal-setting and design processes are
initiated.

Since we have discussed rather thoroughly in earlier chapters
the infological structure of a data base, we shall now concentrate
our efforts upon (i) the goal structure and the functional
structure and (ii) the integration of goal, functional, and
infological aspects of the data base system.
7.3.1 The purpose of a data base

Generally speaking, the purpose, or overall goal, of a data base is

(G) to provide an effective source of information for decision influencers

A decision influencer is a person or group of persons influencing upon a decision by own initiative or as a matter of professional duty. A decision maker is a decision influencer with the authority of finally making a decision.¹

The decisions for which information is supplied by the data base may concern anything: every-day situations of private persons, the effective running of a company, or the well-fare of a community, just to mention a few examples.

As was mentioned in section 7.2, a goal structure should contain comparison criteria which make it possible at decision time to judge the relative merits and deficiencies of design proposals. A minimum requirement would be that the goal structure should provide the decision maker with some rational argument for choosing one proposal before another. The formulation above hardly satisfies this requirement. It has more the character of a compact definition of the data base concept. An entity which does not fulfil (G) should not at all be regarded as a data base.²

¹For a discussion of "influence", "power", "authority" and related concepts see Carzo-Yanouzes [22].

²We disregard in this context from a few imaginable uses of data bases where decision are in no way involved. Example: "gossip data bases" with the only purpose of satisfying people's curiosity.
7.3.2 A high-level goal structure

In order to attain a data base goal structure, which is somewhat more discriminating, and thus more informative to the designer, we shall break down the overall goal (G) into four subgoals, labeled

\( G_1 \) "usefulness"

\( G_2 \) "non-harmfulness"

\( G_3 \) "economy"

\( G_4 \) "viability"

The meaning of these subgoals will be discussed below. They no doubt represent four aspects of the data base decision problem, which deserve explicit consideration, and they raise questions like

\( 1 \) for whom and for what should the data base be useful?

\( 2 \) could the data base have bad side-effects?

\( 3 \) is the data base project economically feasible and to what extent should the organization's scarce resources be put into it?

\( 4 \) for how long should the projected data base be viable, and how shall we see to it that it remains useful, non-harmful, and economically feasible during this period?

As was pointed out in 7.1 there are no formal techniques for breaking down goals and systems. Thus the breakdown of \( \langle G \rangle \) into \( \langle G_1, G_2, G_3, G_4 \rangle \) is in a sense arbitrary, and we cannot exclude that there are other breakdowns which could serve equally well or better.
Obviously the four subgoals are not independent of each other. In fact, if we "stretch" the usefulness goal it may be found to cover non-harmfulness, economy and viability as well, and similarly economy in a wide sense includes the three other goals. However, what is only revealed by "stretching" a goal will easily be forgotten. Thus if we want to call and maintain attention to a somewhat far-fetched aspect of a goal, we had better state it explicitly as a subgoal.

Now we turn to a discussion of each one of the components in the goal structure <G1, G2, G3, G4>.
7.3.2.1 Usefulness

Naturally, a data base must positively promote its "raison d'être", i.e., it must be useful to the decision influencers, supplying them with information, which will make decision-making within a particular decision area systematically better than it would have been without the assistance of the data base.

This is not the place to go deep into the problem of what should be meant by "systematically better decision making". We simply postulate that the decision makers themselves or their principals have methods for evaluating the qualities of the decisions made. It seems natural to require that the behaviour of the decision makers should be improved systematically, i.e., on the average, by the aid of a useful data base.

We limit the scope of the data base to "a particular decision area". We thereby express the insight that it could never be claimed that a particular data base should improve all kinds of decision made in the universe. In order to direct the continued goal setting work one should try at an early stage to make an operational formulation of the decision area of the data base. The formulation should be operational in the sense that it should be possible to determine for every proposed information request whether it belongs to the scope of the data base or not.¹ Thus we do not require the statement to contain an explicit, ex ante enumeration of applications for which the data base is to be designed.

The scopes of different data bases may of course have varying widths; typically a company data base will have a fairly narrow scope, whereas a national or international data base will have a wide scope, for instance. Today widening scopes of the data base seems to be the trend in most organizations, be they large or small.

¹ Of the often stated requirement that a data base should be "prepared to meet known and unknown information needs". The decision area statement should give an answer to the consequence question: "what kind of unknown information needs?"
Within its scope a useful data base should be reasonably complete, i.e. information requests, which, according to the operational criterion introduced above, do belong to the decision area of the data base, should not only be acknowledged by the data base, but with few exceptions cause output from the data base to be produced, which has positive information value to the decision influencer.

The completeness requirement does not imply that the data base should necessarily be capable of answering precisely those questions which are submitted to it, provided that they are within the scope of the data base. The requirement rather implies that the data base should be capable of responding fruitfully to all such requests. One could say that if the data base cannot answer a question put to it, and which belongs to its scope, then it must at least answer another question of related interest to the requestor. Different data bases could have different densities: a dense data base will find answers to questions, which are more closely related to the submitted questions, than will a sparse data base.

The concepts which have presented themselves during our analysis of the usefulness goal may be illustrated by a few simple pictures. In figures 6-8 the dots represent data base messages and the inverse of the distance between them is a measure of their "infological similarity". Many requests to the data base will hit "white areas" of the target; if the data base is complete, there will be a decision-relevant message at a short distance from the point of impact; on the average this distance will be smaller in a dense data base than in a sparse one.

To summarize then, a useful data base is a data base which systematically improves decision making within a specified decision area. This area could be wide or narrow, but it should be stated in operational terms as defined above at an early stage of the goal setting process. A useful data base must be

\footnote{Cf the discussion of "infological distance" in section 3.2.4.}
Figure 6  a) incomplete data base with wide scope;
b) complete data base with narrow scope

Figure 7  a) complete, sparse data base;  b) complete, dense data base

Figure 8  Wide, complete data base, generally sparse but locally dense
reasonably complete within its scope. A dense data base is more useful, at least potentially, than a sparse one.

Other subgoals of the usefulness goal are discussed later in this chapter. For instance, \textit{timeliness of response} will of course have decisive influence upon the usefulness of a data base.

\textbf{Remark.} In the terminology of Marschak\(^1\) a dense data base has a \textit{fine information structure}, whereas a sparse data base has a \textit{coarse information structure}. Marschaks model uses the matrices drawn in figure 9. The \textit{payoff matrix} \(u_{ij}\) shows the resulting utility, if the decision-maker implements action \(a_i\) when the "state of nature", or the "prevailing constellation"\(^2\), is \(c_i\). The \textit{information structure matrix} \(p_{ik}\) shows the conditional probability that the data base will contain message \(m_k\) provided that the constellation \(c_i\) prevails. Suppose now that the column \(m_k\) of the information structure matrix contains several "1" probabilities, and that the corresponding column \(m_k\) of the payoff matrix contains \textit{different} values in the corresponding rows. This means that the data base would represent with the same message several constellations, yielding different payoffs provided that a certain course of action is undertaken; the information structure is then said to be \textit{too coarse}. If, on the other hand the data base would represent with different messages constellations which have identical payoffs for all actions, the information structure is said to be \textit{too fine}. Note that an information structure may be \textit{simultaneously too fine and too coarse}.

\(^1\)Marschak \([23]\). See also Emery \([24]\), chapter 4.

\(^2\)Of section 2.3.1.
**Action**

\[
\begin{array}{cccc}
\text{Constellation} & c_1 & u_{11} & \cdots & u_{1s} \\
& \vdots & \vdots & \ddots & \vdots \\
& c_i & u_{i1} & \cdots & u_{is} \\
& \vdots & \vdots & \ddots & \vdots \\
& c_n & u_{n1} & \cdots & u_{ns} \\
\end{array}
\]

**Figure 9a** Payoff matrix \( (u_{ij}) \)

---

**Database Message**

\[
\begin{array}{cccc}
\text{Constellation} & c_1 & p_{11} & \cdots & p_{1r} \\
& \vdots & \vdots & \ddots & \vdots \\
& c_i & p_{i1} & \cdots & p_{ir} \\
& \vdots & \vdots & \ddots & \vdots \\
& c_n & p_{n1} & \cdots & p_{nr} \\
\end{array}
\]

**Figure 9b** Information structure matrix \( (p_{ik}) \)
7.3.2.2 Non-harmfulness

The database should not be harmful; this goal has implications in two directions:

A The database should not too often and/or with too serious effects cause worse decisions to be made than would have been made without the assistance of the database. Note that non-harmfulness in this sense is not a logical consequence of usefulness as defined in the previous section; a database may very well on the whole lead to improved decision-making, although a few decisions made over a period of time are of unacceptable quality; the average usefulness of the database is then above zero, but negative deviations from the average, which will almost certainly occur with any database, are too significant or too frequent.

B Negative side-effects of a database should be under control. Any database will inevitably have effects outside the decision area for which it has been primarily intended. These effects may be called side-effects or external economies and diseconomies. Positive side-effects will in principle add to the usefulness of the database. Failure to recognize negative side-effects of a proposed database may not only cause sudden death to the particular project, but may more importantly have severe back-lash effects on other, non database efforts made by the database projecting organization and on database ventures undertaken by other organizations. This has been clearly demonstrated by the public database debate which has been quite vivid in Sweden and other countries during the last couple of years\(^1\), and which in addition to a lot of excellent consequences has undoubtedly caused a few fatal back-lashes of the kind mentioned above.

The discussion under A and B has hopefully convinced the reader of the necessity to consider "non-harmfulness" an autonomous, high-level goal component, and that it is not sufficient to regard it as a correction item to the usefulness component.

\(^1\) This is written in 1972.
7.3.2.3 Economy

The data base should represent a reasonable utilization of scarce resources, i.e., it should not be possible to make the organization better off, whatever that may mean, by transferring resources between the data base project and alternative ventures in one direction or the other.

Beside the benefits and negative effects produced by a data base we naturally have to consider its resource consumption. The resources to be or not to be allocated to the data base are more or less tangible (theoretically and practically expressible in monetary terms), and they may be classified in several dimensions. According to one classification scheme one could talk about persons, know-how, software, hardware and money as different kinds of resources; of these money is of course the most flexible resource, whereas the purchase of hardware, the fabrication of software, and above all the recruitment of systemers, programmers, operators, and other employees imply long-range commitments for the organization.

In another dimension one could distinguish between design, construction, and operation resources to be consumed by a considered data base project.

It is important that the goal-setters and decision-makers do not exclusively look at the more or less artificial price-tickets that may have been applied at some time for some reason to the different resources. It is very useful in connection with data base projects to recollect the opportunity cost concept. The opportunity cost for using a resource for which the organization already has a long-range commitment may be far below what the price ticket says, just to mention one example. It is also important to remember that it may be an equally serious mistake to be too conservative in one's judgment of the future benefits of a project as to overestimate them.

---

1 A resource consumption is theoretically but not practically expressible in monetary terms when there is a correct method of measuring the dollar cost, but when the implementation of this measuring method would itself cause costs of at least the same magnitude as the cost to be measured. Cf. section 7.2.
2 Cf section 7.1.4 where we had a general discussion of the flexibility of resources.
7.3.2 Viability

The database should be viable, which means that it should be so designed as to be useful, not harmful, and represent a reasonable utilization of scarce resources not only immediately upon its implementation but for some consciously chosen duration of time.

It could be argued that the viability requirement should not be an autonomous goal, because the usefulness and economy goals could be so interpreted as to cover the viability aspects of a database as well. If this argument is carried out just a little bit further, it could also be maintained that usefulness or economy is the one and only database goal. The obvious counterargument is that when it is only possible to sort in one important goal under another with the aid of rather far-fetched interpretations of the latter, there is a significant risk that the former will be more or less neglected in practical situations, when one has not got time for too deep theoretical analysis. As it is no doubt extremely important that the goal setters and decision makers make it clear to themselves, at an early stage of the goal setting and design process, for how long the database is expected to live, and what changes in its environment it should resist, I find it justified to consider viability a database goal of equal rank as usefulness, non-harmfulness and economy.
7.3.3 Reconciliation of goals

There is an absolute and a relative aspect of a goal structure and of each component goal of the structure as well. A relative formulation of a database goal makes it possible to compare different design proposals with respect to this goal. On the other hand an absolute formulation of a goal defines an ideal level to be aimed at in the design work.

It is natural to start a goal setting process by stating a few general goals from the relative point of view. In connection with data bases that would imply, according to our previous analysis, that we stated conditions under which one data base would be more useful, or less harmful, or more economizing with scarce resources, or more viable than another data base. When a goal is at least in principle quantifiable, it is customary to replace the relative conditions with a single maximization phrase: "we want as much as possible of property x ...".

To give a set of relatively formulated goals amounts to the same thing as stating desirable goals, or desirables\(^1\).

A set of desirables is of limited value to the designers of a data base. One design may be doubtlessly preferred to another, only when the former is better than the latter according to all relative conditions that have been stated. Thus the desirables may be used to rule out so-called Pareto-inferior solutions.\(^2\)

The goal setting process cannot be allowed to terminate with only a set of desirables at hand; the desirable goals sooner or later have to be reconciliated into a goal structure consisting of a set of absolute goals, i.e. ideal levels, and a set of comparison criteria, which enable overall, inter-goal comparisons to be made between different design proposals. Even

\(^1\) Cf section 7.2 and Langefors [26].

\(^2\) If solution (strategy, course of action) A is Pareto-inferior to solution B, it is also customary to say that A is dominated by B.
if such a goal structure is never made perfectly explicit during the goal stating and design process, it will be implied by the successive, intuitive decisions made by the decision makers, provided that they are consistent in their judgments. ¹

After the goals have been reconciliated at certain levels, and the design loop has been repeated a few times ², it may be questioned whether the ideal level of one or more goals should be shifted upwards or downwards. Then it is important that one has some idea of the logical dependencies ³ that inevitably exist between different goal structure components; if there were no such interdependencies, there would not be any reconciliation problems.

Goal discussions often become confused because different people make different, tacit assumptions about goal interdependencies. Even when the number of goals under consideration is low, confusion may easily occur, and in such situations it may be convenient to make different opinions explicit, e.g. by means of the illustration technique used in figure 10.

In figure 10 the data base goal structure \(<G_1, G_2, G_3, G_4>\), defined in section 7.3.2, is represented by a directed graph. Every goal in the goal structure is represented by a node in the graph. The arrows originating in a particular node, A, are intended to illustrate the "most likely" effects upon the other goals by an upwards shift in A, i.e., the shifting of the corresponding goal to a more "ambitious" level.

¹ Cf section 7.2.

² Cf figure 4.

³ As was demonstrated in section 7.1.4 these dependencies also manifest themselves in the resource/process structure of the (data base) system. If the same resource is a direct or indirect precedent of two different processes, contributing to different goals, then the goals of the two processes are potentially conflicting goals. Conversely, the goals of two processes may reinforce each other if a succedent resource of one process is a precedent resource of the other. Cf also figures 1 and 2.
Remark. Note that we are here concerned only with the interdependencies which exist between different goals on the same level in the goal/subgoal hierarchy (of figure 1). These interdependencies could be called horizontal interdependencies. Naturally the goal breakdown process itself, together with the horizontal interdependencies, will imply vertical interdependencies of different kinds as well. For instance, in figure 11, $G_{11}$ and $G_{12}$ contribute to $G_1$, to which they are therefore vertically related. Similarly, $G_{21}$ and $G_{22}$ are vertically related to $G_2$, of which they are subgoals. Now, suppose that $G_{12}$ and $G_{21}$ are horizontally related, e.g. because processes contributing to them directly or indirectly use the same resource. Then there is an implied vertical relation between $G_{12}$ and $G_2$, as well as between $G_{21}$ and $G_1$.

Remark. Though related to the goal structure $\langle G_1, G_2, G_3, G_4 \rangle$, developed for data bases in general in section 7.3.2, the discussion here in connection with figure 10 does not aim at deriving unquestionable truths about this particular goal structure, which is merely used for the purpose of illustrating a graphical technique. Possibly the statements about specific interdependencies between specific data base goals could be regarded as preliminary hypotheses for further research.

Now let us return to figure 10. If two goals are conflicting, an upwards shift of the level of ambition for one of them will necessitate a downwards shift for the other. unless we are able to increase the total amount of resources available, or unless it is possible to "buy" more of the resources competed for by the conflicting goals with other, flexible\(^1\) resources; in the latter case other conflicting goals will be affected instead.

Figure 10 does not show all potential goal conflicts and goal reinforcements. Instead the idea is to show for each goal, by means of arrows leaving the goal symbol, the "most likely" net effect upon each of the other goals, incurred by an upwards shift in the particular goal under consideration. By\(^1\) Cf 7.1.4.
Figure 11.
"net effect" we mean the resulting effect after the system has been redesigned to keep up with the particular goal shift considered. If the expected net effect is an upwards shift of the other goal, too, this is represented by a "+" arrow. In the opposite case, there is a "−" arrow. If the other goal will probably not be very much affected, there is no arrow at all. For instance, as is shown by figure 10, we think that an upwards shift of the viability goal will cause an upwards shift in the usefulness goal. This means that if we settle upon a more ambitious viability level for the data base to be designed, the data bases satisfying the new viability level will "automatically" satisfy a higher usefulness level. A reason behind this could be that a viable data base will probably provide information compatibility for a longer period of time for a certain decision area than will a series of short-lived constructions, and thus the information in the more viable data base will probably be more useful.

Let us take another example from figure 10. If one wants to make a total saving of resources, this means that the economy level is shifted upwards. According to the figure this implies lower viability and lower usefulness, whereas the non-harmfulness goal is left unaffected. This expresses the insight that the latter is probably much more rigid (less elastic) than the former two: if we have to save resources, we will probably cut off data base features, the usefulness of which has already been questioned, or features which can only be defended on the grounds that they make the system more flexible, and thus more viable; the non-harmfulness goal is often much more "discontinuous": (it is felt that) one has to maintain a certain (minimum) level of quality and protection and there is no margin to bargain with.

Two arrows in figure 10 have a "¿" sign on them. This means that it is hard to say without further analysis whether an upwards shift in one goal will result, as a net effect, in an upwards or downwards shift in the other. For instance, higher viability may cause more resources to be consumed in the near future, whereas maintenance costs may grow slower
with time, and the development costs for an heir system will occur later; we have thus here a typical investment situation.

Finally, note the difference between situations where there is a "?" arrow from one goal, A, to another, B, and situations where there is no arrow at all. In the latter case we are convinced that an upwards shift of A will not have any important net effects upon B, whereas in the former situation we think that there are effects but we are uncertain whether they force the level of B upwards or downwards.
7.3.4 Data base subsystems

Of the four data base systemeering tasks mentioned in the introduction to 7.3 we have discussed in the previous sections (7.3.1 - 7.3.3) the task of defining and analyzing the goal structure of a data base. In this section we attack the task of finding a feasible subsystem structure, or functional structure, of the data base. As a matter of fact we have already made a similar kind of breakdown, and that was when we in chapter 4 defined a subsystem structure for the data base environment. If we recall from section 7.1.4 the definition of the environment of a system as "those parts of the supersystem which are external to the considered system", we realize that a breakdown of the environment of the data base amounts to much the same thing as a breakdown of the supersystem of the data base, and that a breakdown of the data base itself is a repetition of the same task on the next lower level in the system/subsystem hierarchy. Cf also section 7.1.3.

The environment analysis carried out in chapter 4 implies a certain demarcation between the data base and the other subsystems of the supersystem of the database. However, as was stressed in 4.3 and 7.2, this does not mean that in every practical data base design situation a clear-cut borderline between controllable conditions and conditions which are out of the designer's control could or should be defined. Nor does it mean that the borderline is drawn identically in different data base projects. For instance, in one project the computer configuration may be a more or less uncontrollable condition, whereas another project may be so timed as to coincide with the purchase of new hardware equipment, a coincidence to be taken advantage of, naturally. These qualifications should be kept in mind, because in the analysis to follow we concentrate upon the most "narrow" design situation, i.e., a design situation where there is not very much we can do about the environment subsystems as defined in 4.3.1. One reason for a narrow demarcation is that this makes the analysis more general. Another reason is that there is very little theory as yet even for the "narrow" data base

---

1 Recall in particular figure 2 of chapter 4.
design problem, and there is a lot of empirical evidence
in support of the hypothesis that we shall need such theory
very badly, if we are at all to become competent designers
of data base systems.\(^1\)

One subsystem structure of the data base was established in
section 4.4. However, as was stressed there, the aim of that
breakdown was primarily to provide a deeper understanding,
e.g., among consumers and decision-makers, of the data base
concept as such. In this section we aim at a breakdown which
is feasible from design point of view, which implies for
instance that the breakdown has to be meaningful and useful
to the designers and constructors of the actual software
products which are to control the operative data base system\(^2\).
This means that we have to leave, gradually, the strictly
infological conception of the data base. Instead it becomes
practical to start thinking of the data base system as a
set of data processes, i.e., processes which use and produce
data resources\(^3\). That it is actually possible to carry out
the transition from infological to datalogical thinking
gradually, in very small steps, will be demonstrated in some
detail for one particular data base subsystem in chapter 8,
"File design".\(^4\)

Figure 12 shows a tentative, crude functional structuring
of the data base, which we believe to be compatible with
the requirements

(a) that the structuring should be meaningful and
useful to designers and programmers of data base
software, and

\(^1\) As was pointed out in 5.4.2, even imposing joint efforts
like the one undertaken by the experienced professionals of
the CODASYL Data Base Task Group are sort of "in the air"
without a sound theoretical foundation.

\(^2\) "Data base management system" has become the standard label
for such software.

\(^3\) Cf. the general discussion in 7.2 of a system as a set of
interrelated resources and processes.

\(^4\) Cf. also chapter 1, figure 5.
Figure 12 A structuring of the set of all data base processes into eight data base subsystems
(b) that the structuring should permit gradual transition from infological to datalogical concepts and thinking, which in particular means that it should be possible to transform, in a systematic way, the desired infological properties of the data base\(^1\) into datalogically oriented software specifications.

Requirement (a) implies that the structuring should not be just any imaginable subdivision of the set of all data base processes in the central box of figure 12, but a structuring which imposes a subdivision of the data (and program\(^2\)) resources that will make it possible to partition the total task of designing the data base system into more tractable data structuring and program design subtasks.

Let us now very briefly discuss the meaning and scope of each of the eight data base subsystems depicted in figure 12.

The subsystem labeled "files" is the datalogical counterpart of the infological "nucleus" subsystem (cf section 4.4). Beside the actual data files it contains the file access mechanisms, i.e., the programs for accessing the files as well as may auxiliary data, index tables, control blocks, etc., required by the access programs.

The "catalogues" subsystem contains data representing "information on information"\(^3\) together with programs for processing this information. Thus it contains the formal and informal definitions of attributes, object relations and other infological entities, and it also contains programs corresponding to the more or less intricate deduction and induction rules of the infological model underlying the data base\(^4\).

The "interaction" subsystem is the communication interface between the data base and its environment. Among other things

---

\(^1\) As stated in the goal structure of the data base.

\(^2\) A program may be considered as a particular kind of data resource. Cf chapter 9.

\(^3\) Cf 3.2.5.

\(^4\) Cf 4.4.1.
it contains the compilers and interpreters of the data base interaction languages. It will translate the input transactions into the "internal dialect"\(^1\) of the data base, and conversely it will edit the output transactions appropriately.

The "quality" and "protection" subsystems are the datalogical counterparts of the infological "filter" function (cf. section 4.4). The importance of these subsystems will grow rapidly as the complexity of data bases increases. The GPI-system, [151], designed, constructed, and implemented at the National Central Bureau of Statistics in Sweden, provides a good example of what can be done if due, explicit attention is paid to the problems of quality control; at the same time this project has also revealed that the complexity and costs of such efforts may easily be underestimated.

The quality control problems seem to have been more or less neglected by the manufacturers of the data base software which is available on the market today. In the protection area these systems are usually equipped with a few features like data protection through password techniques, and certain logging and recovery functions. However, many problems remain to be solved, even on the theoretical level.\(^2\)

The "accounting" subsystem should collect statistics about different aspect of data base utilization and performance, not only to make it possible to distribute costs among different users of the data base, but also in order to supply the "maintenance" function and/or the data base administrator with adequate information for reorganization decisions etc.

The scheduling subsystem, finally, is responsible for the planning and coordination of the other processes. For instance,

---

\(^1\) Cf Nordbotten [55].

\(^2\) A lot of research and development work is going on at present in the security, protection, and privacy areas. See for example references [154], [155], [158], [159], [160], [161], [162], [163].
it should dynamically find efficient strategies for processing particular retrieval requests, and it should administrate the queues in front of hardware resources like disk drives and channels. More about these problems will be found in chapter 9.

We shall not over-emphasize the importance of exactly that subsystem structure of a data base, which is given in figure 12. "Quality", "scheduling", "protection", "interaction", "accounting", "catalogues", "files", and "maintenance" are believed to form a feasible subsystem structure for many data base design projects on a particular stage of analysis. There are two qualifications to be made, however. Firstly, the particular character of an individual data base may distort the proportions between the subsystems, so as to make it more feasible even at an early stage of analysis to divide one of the subsystems into several functions. Secondly, the further breakdown of the structure of figure 12 almost certainly will not result in a perfect tree structure. Once in a while it will be found purposeful to isolate a function on one level which may serve several functions on the next higher level.  

\[1\] Cf figure 1 and the foot-note in the beginning of section 7.1.3.
7.3.5 Matching the goals and the functions

As we have pointed out earlier, the processes of breaking down goals and functions are by and large creative tasks. What theory could do in such areas is to supply formal tools stimulating the creative processes and preventing the systemeer from forgetting important aspects of the problem. One such technique will be presented in this section.

Let us assume that a database systemeer has

(1) defined, at least roughly, a particular infological model for the data base

(2) defined a high-level goal structure like \( \langle G_1, G_2, G_3, G_4 \rangle \) described in 7.3.2

(3) defined a high-level data base subsystem structure like the one discussed in 7.3.4

How should he continue? Naturally, he could try to refine each of the breakdowns separately. However, he will probably get more fruitful ideas by asking himself questions like

(a) what subsystems may contribute to a particular goal, and how?

(b) to what goals may a particular subsystem contribute, and how?

In order to answer these questions in a systematic way he could draw a matrix (figure 13) with the goals on one axis, and with the database subsystems on the other. Then he should consider every cell of the matrix carefully. He may find some of the cells irrelevant, whereas others may raise a lot of interesting problems, which he has to investigate further. We shall give a few examples of what questions may be raised for particular cells in the goal/function matrix.
<table>
<thead>
<tr>
<th></th>
<th>USEFULNESS</th>
<th>NONHARMFULNESS</th>
<th>ECONOMY</th>
<th>VIABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CATALOGUES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERACTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUALITY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROTECTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCOUNTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAINTENANCE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCHEDULING</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 13** Goal/function matrix
Example 1. <FILES, USEFULNESS>

(a) What message types (e-concepts) should the database contain?

During early stages of the design process such questions may only be answered in terms of rather vague categories of message types, so-called pro-concepts.¹

(b) How many time versions of each message type should be saved in the database?

Example 2. <FILES, NON-HARMFULNESS>

(a) What message types are confidential?

(b) Who should be authorized to do what with confidential message types?

Example 3. <FILES, ECONOMY>

(a) How many instances will there be of each message type?

(b) How much direct access storage can we afford to spend upon the files?

(c) What priorities are there, if we cannot store all the desired information?

Example 4. <FILES, VIABILITY>

(a) With what rate will the number of message types and the number of instances per message type increase?

¹ Cf [52].
Can we expect new storage technology and changed cost relationships during the life time of the data base system?

**Example 5. <INTERACTION, USEFULNESS>**

(a) What categories of interactors should the data base be capable of interacting with?\(^1\)

(b) What interaction types will occur?\(^2\)

(c) What kind of interaction languages should be utilized/design for different kinds of interactors and interactions?

(d) How fast does the data base have to respond to different kinds of requests?

This question needs a well differentiated answer. Some requests may require response within 3 seconds, others might not suffer very much from having to wait for hours or days. In connection with large statistical data bases, for instance, an information consumer will typically want to know very fast whether he will at all be able to get the desired information, and how much it will cost, whereas he may very well be prepared to wait over night for the actual set of tables to appear.

**Example 6. <INTERACTION, ECONOMY>**

(a) How large will the total interaction work-load be?

(b) How will the total interaction work-load be distributed over time (of day), interactor categories, interaction types, etc?

\(^1\) Cf chapters 4 and 5.

\(^2\) Cf chapter 5.
Example 7. <QUALITY, USEFULNESS>

(a) What quality declarations of the messages are needed in order to make the information meaningful to different categories of interactors?

(b) Can the information be allowed to contain inconsistencies? If not, how can they be avoided?

(c) To what extent will "missing data" be acceptable? Should the data base try to prevent such situations from occurring by automatically reminding late information suppliers? What techniques should be used when missing data are needed for the processing of different kinds of requests?

Example 8. <QUALITY, ECONOMY>

(a) How much redundancy can we afford in the data base?

Example 9. <PROTECTION, NON-HARMFULNESS>

(a) Could over-protection of data base information hurt vital interests of the organization?

Unnecessary drain of information could be very harmful because of lost opportunities. Moreover, over-protection may be indirectly harmful, if the protection measures strike most people as "ridiculous" and/or if they are so hampering that they cannot be maintained "in practice".

Example 10. <PROTECTION, VIABILITY>

(a) Will protection procedures have to be changed during the lifetime of the data base, e.g., because of expected government bills?
Example 11. \(<\text{MAINTENANCE, USEFULNESS}>\)

(a) What kind of reorganizations should the data base system be able to carry out automatically on its own initiative?

(b) What kind of reorganizations should the data base system be able to carry out automatically on the data base administrator’s initiative?

(c) To what extent is it possible to state formal reorganization criteria?

If it is possible to state complete, formal reorganization criteria, this implies (i) that all reorganization could be automatically initiated, and (ii) that the designers need not care so much about the original file organization, because the reorganization procedures will anyhow automatically see to it that the organization will approach optimum; thus if we can solve the problem of automatical redesign, we shall not need to think any more about automatical design!\(^1\)

Naturally, the examples above only give a very small sample of questions that the matching of data base goals against data base functions may raise. Also there may very well be much more adequate goal and subsystem structures than those used here. Our main purpose has been to show how one could proceed to solve the creative data base systemeering task of finding successively more detailed and operational goal formulations and subsystem descriptions. It is certainly a task well worth further study to extend the list of examples above to a complete check list for data base systemeers, as well as to test alternative goal and subsystem structures.

\(^1\) However fascinating this may sound, it probably only means that the problem of automatical redesign is still more difficult to solve than the problem of automatical design. Or does it not? At least it is often easier to find successively better solutions of a problem than to find the best one at once.
8.1 Introduction

Any system engineering problem raises two kinds of questions: "what?" questions, and "how?" questions. "What do we want the planned system to achieve?" and "How should the system be constructed?" It seems to be a human peculiarity to escape the "what?" questions and to be attracted by the "how?" questions, even though we all have our enlightened moments when we admit that the former are to a great extent logical precedents of the latter. An explanation to this phenomenon might be that it is often easier to state the "how?" problems, and solutions to them, in precise, operational terms, than it is to do the same for the "what?" problems.

The human tendencies and the relative difficulties of the two classes of system design problems seem to conform to the general pattern described above even within the data processing field. Accordingly, much of the data processing literature deals with narrowly, though precisely defined sub-problems. Few attempts have been made to relate the sub-problems to each other in a systematical way, and to evaluate the relative significance of the problems under varying environmental conditions. On the other hand, when gross perspectives are really adopted, like in a certain kind of "management-oriented" literature, the reported results often suffer from a lack of precision which makes them hard to translate into concrete guidelines for practical project work.

The conceptual frame-work and the methodology developed in this report could possibly help to bridge over the gap between the "what?" questions and the "how?" questions in the problem area of data base design. In previous chapters we have tried to develop concepts and models which among other things should enable data base designers to state and tackle the "what?" problems with just the necessary degree of precision. On the one hand this degree should be so great as to make the communication within a project team consisting of both "what?" -oriented people and "how?" -specialists meaningful and constructive. On the other hand the "what?" problem formulations should not lay unnecessary restrictions
on the "how?" problem solutions.

The remainder of this report will be devoted to a survey of the "how?" problems within two major data base design areas: "file design" and "process design". In this survey we deliberately try to avoid repeating what has been written on sub-problems within these areas in numerous books and articles, some of which may be found in the bibliography at the end of the report. Instead our main objective is to show that the infological frame-work, developed in previous chapters, may be extended in a very natural way to cover the datalogical aspect of the data base design problem. We want to show that the extended infological frame-work will enable representation and computer oriented data base designers to tackle datalogical "how?" problems with at least the same precision and systematics as offered by the more or less general and more or less well-defined datalogical frame-works which are in use today in the computing world.

The datalogical extension to the infological frame-work, to be presented in chapters 8 and 9, contains definitions and systematizations of many well-known datalogical concepts and problems. It is hoped that these definitions and systematizations will be recognized as "straightened up" versions of the old conceptions rather than as completely new and strange coinages. The effort has been to produce a set of datalogical conceptions which are natural and formally well-defined complements to the infological conceptions introduced in earlier chapters, but which are still by and large compatible with the conceptions of experienced computer professionals.

In section 8.2 we shall define and discuss a number of concepts which are fundamental to the understanding of the file design problem, e.g. "memory", "file", "entry", and "cluster". The definition of "file" will be coordinated with the earlier stated definition of "message type" (section 3.2.3) so as to facilitate the transition in the design process from "the

1 Recall from section 2.1 the identification of (a) an object system aspect, (b) an information aspect, and (c) a datalogical aspect of the data base design problem.
infological sphere" to "the datalogical sphere". In section 8.3 then, we shall define a set of file structuring operators by means of which the set of e-concepts of a specified infological model may be gradually transformed into a feasible and efficient file structure. The ability to describe the transformation in terms of small, formally well-defined steps is believed to be a necessary pre-requisite for automatic or semi-automatic file design. In section 8.4, finally, we discuss and exemplify how the conceptual tools introduced in 8.2 and 8.3 may be used in practical file design situations. We assume then that the file structuring problem has arisen within a data base project where the infological frame-work has been used during earlier design stages.

Chapter 9 will be devoted to process design problems. In "traditional" data processing systems the problems regarding the planning, scheduling, and coordination of processes are by and large solved at design time, before the system is put into operation. In data base systems the situation is typically quite different. Many data base process design problems will have to be solved automatically and dynamically at operation time by the data base system itself. Why this is so, and what are the implications thereof, will be explained in section 9.1. At the same time a number of basic concepts, like "program", "process", and "routine", will be defined and discussed. In the latter part of chapter 9 we shall use the extended infological frame-work in order to show in some detail what process design problems a data base system has to solve dynamically, and how it might be designed so as to be able to perform this task.

"File and process design" could be regarded as a common label for the design tasks pertaining to the "files" and "scheduling" subsystems of a data base system. However, we recall from section 3.2.5 that there are several data base subsystems which may themselves be regarded as data bases, so-called meta-data-bases. Thus file and process design problems will

\(^1\) Cf. chapter 7, figure 12
occur recursively in other data base subsystems than "files" and "scheduling", and the applicability of the material in chapters 8 and 9 will consequent ly be wider than it might appear at first sight. Admittedly though, there are several datalogical "how?" problems pertaining to the data base subsystems of figure 12, chapter 7, which will not be covered in this report. Hopefully, however, it will turn out to be possible to fit even those data base design problems which are not explicitly treated in this text into the general conceptual frame-work developed here.
8.2 Basic concepts

8.2.1 Accessible memories

In chapter 4 we defined the computer as a subsystem of the database environment, and thus not a part of the data base itself. We shall not review the reasons for this demarcation. However, the data base designer naturally has to consider carefully how to utilize in the best way the resources of the environment subsystems, which are at his disposal. Among other things he has to consider how to utilize computer hardware resources: primary and secondary storage, channel and processor time, etc.

In order for the data base designer to be able to pay adequate attention to computer hardware, neither too much, nor too little, he has to be equipped with adequate concepts for describing different hardware facilities in a way which is relevant to his central problems. One of these problems is that which is the theme of this chapter: to design the files of the data base system in such a way that they may in the end be allocated to available secondary storage and then efficiently operated upon until they are automatically or manually reorganized and/or reallocated. In this connection the designer needs a couple of concepts by means of which he may roughly categorize available memories. We shall try in this section to develop a basic set of such concepts, and we start by rehearsing the definition of "data", stated in chapter 1:

**Definition.** If a person intentionally arranges one piece of reality to represent another, we shall call the former arrangement data, and we shall say that the arranged piece of reality is a medium, which is used for storing the data.

In connection with EDP the medium may be a magnetizable surface, for instance. A memory, then, is defined in the following way:
Definition. A memory is a structured medium, an ordered set of storage positions, \(\langle p_1, \ldots, p_n \rangle\), where each \(p_i\) can be in at least two different states. The individual storage positions of a memory may be temporarily or permanently grouped into larger storage structures, called (storage) blocks, tracks, cylinders, etc.\(^1\)

The blocks of a magnetic tape are a good example of a temporary storage structure. On the other hand the tracks and cylinders of a disk memory are obviously permanent.

From the definitions follows that a memory or any substructure of it, down to an individual storage position, may be used for storing data, i.e., for representing (a part of) a piece of reality. The data which are actually stored in the memory (storage position, ...) at a particular point of time, \(t\), will be called the data contents of the memory (storage position, ...) at \(t\). If a memory (storage position, ...) is not used for storing data at a particular point of time, \(u\), we shall say that the memory (storage position, ...) has the null data contents, \(\emptyset\), at \(u\).

Stored data are not of much use if they cannot be accessed and processed. That is why we define the concept of an accessible memory.

Definition. An accessible memory is a quadruple \(\langle x, y, z, u \rangle\), where \(x\) is a memory, and \(y\) is an access mechanism, capable of presenting the data contents of any named storage position\(^2\) of the memory to a processor, \(z\), to which the memory is tied by means of a channel, \(u\). The access time needed to present a particular storage position, \(p\), is a function of

\[(a) \quad \text{the distance, } \delta(p, p'), \text{ between the demanded storage position, } p, \text{ and the most recently demanded storage position, } p', \text{ within the same memory, and} \]

\(^1\) For a thorough discussion of storage structures see Bachman [202]

\(^2\) A storage position may always be named by its ordinal number in the memory structure. Cf the definition of "memory" above.
the number of storage positions belonging to the same access block as \( p \), i.e., the number of storage positions which has to be transmitted from the memory via the channel to the processor in the same access operation.

If the size of the access block were always equal to 1, \((b)\) would not have to be considered. However, for the memories available on the market today the size of the access blocks are usually\(^1\) larger than one. Today’s hardware and operating systems are usually designed so as to make access blocks and storage blocks coincide. However, it is often possible by means of low-level programming\(^2\) to demonstrate and make use of the fact that access blocks and storage blocks are different concepts, and that an access block may be both smaller and larger than a storage block.

The distance function, \( \delta \), defined above, is an important concept for classifying memories into different categories, such as "serial" and "direct" and subcategories of these.\(^3\) Rough classifications of this kind should be essential to human, possibly computer-aided, designers as well as to automatic maintenance and redesign functions of advanced data base systems.

It is customary\(^4\) to break down the total access time into components like

- positioning time
- rotational delay
- transmission time

\(^1\) If there are any exceptions at all depends upon how we interpret the "storage position" concept for the particular memories.

\(^2\) For instance, IBM uses the terms "channel programming" and "chained scheduling" in this context.

\(^3\) For a more detailed discussion of this topic, see Sundgren [199].

\(^4\) See for instance Martin [32].
Such a breakdown is meaningful if and only if the resource utilization pattern is different during different phases of an access operation. This is often the case. For instance the channel will certainly be occupied during transmission time, but during rotation and/or positioning time it may be free for users of another memory, which is tied to the same channel. Beside being meaningful, the breakdown is practical at a certain design stage if and only if it will substantially affect the design decisions at that particular stage. This will certainly not be the case, for instance, if there is only a low probability that the environment will "simultaneously" produce two transactions, the processing of which will engage the same channel but different memories. In general, of course, it is very difficult to tell how much attention the data base designer should pay to positive and negative overlapping, or multiprogramming, effects. Naturally he has to consider the resource drain due to other applications than the data base system, if there are any. As to the data base application itself, it may be perfectly legitimate at early design stages to regard the computer essentially as a single-programming system, or, if there are relevant statistics available, to multiply throughput and response time figures by certain rough percentages. The important thing is not to assume multiprogramming benefits without allowing for its costs. If the dynamics of a data base environment is very little under the control of data base designers and administrators, and if the environment produces a lot of more or less simultaneous transactions requiring response within few seconds, simulation techniques may be of great help in the design work.¹

¹ See Dubenko & Berild [32].
8.2.2 Files, entries, and terms

We have defined the file design task as the task of mapping an infological structure on a storage structure. We have discussed infological structures rather thoroughly in earlier chapters of this report, and in the previous section, we defined briefly a basic set of storage structure concepts. What we need now is a set of concepts which allow us to proceed smoothly, by small, perceivable design steps, from the strictly infological level, where the storage structure is not at all considered, to the strictly computer-oriented level, where bytes, blocks, accesses and microseconds are of primary concern. Thus we are about to enter a sphere of intermediary concepts, which in current computing world terminologies often appear with the prefix "logical". In this section we shall try to give precise definitions of "file" and a few other related concepts, which seem to be useful in file design work.

Before we start to state and analyze the definitions, we shall point to a difficulty. We shall often say that a file, an entry, or some other kind of entity, represents a particular infological entity. As was stated in chapter 1 and rehearsed in the previous section, a data representation is an arrangement of a particular piece of reality, e.g. a particular memory. However, files, entries, etc., are not data representations in this sense until they have been allocated to physical memories. Until then they may be regarded as abstract data representations, or data representations which are temporarily allocated to an imaginary, virtual memory, without any particular storage structure and without any limitations. Upon these abstract representations we may perform the structuring operations to be defined and discussed later in this chapter. Thus the initial, "normal" file structure, to be defined below, will be successively transformed until it "fits" the available storage structure and may be mapped, or allocated, to that structure. The allocation to a physical memory marks the end of the design process and the birth of the files, entries, etc as non-abstract entities.
Now we turn our attention to the following definition of the file concept.

**Definition.** A file is a triple \( <x, y, z> \) where

(a) \( x \) is a set of **entries**, each of which is an ordered set of (data) **terms** representing a message belonging to a particular message type\(^1\), and each of which is uniquely identified within the file by the data contents of a subset of its terms, called the **entry point** (term group), or the (primary) **key**, of the entry.

(b) \( y \) is an entry **description** describing how the terms of the entries of the file should be rearranged in order to conform to the **normal format** for representing messages with file entries; the normal format is assumed to be common to all files of the data base.

(c) \( z \) is an **access algorithm\(^2\)**, which, given the data contents of a particular entry point, delivers for further processing the complete entry, identified by the particular entry point value, and edited in accordance with the **normal format** mentioned in (b); the access algorithm should also be able to retrieve, for successive processing, all the entries of a file, as the result of one request, in which the entry point values of the file do not have to be explicitly given.

There is a representative relationship between the datalogical concepts "file", "entry", and "term", and the infological counterparts, which are "message type", or "e-concept", "message", and "reference", respectively. The data representation of an infological entity is not at all uniquely determined, of course. Many a time different data representations of the same infological entity will occur even within one and the

---

\(^1\) Cf section 3.2.3.

\(^2\) A formal definition of "algorithm" may be found in Knuth [61], for instance.
same base. For instance a particular attribute may be represented by the text string "INCOME BEFORE TAX" in data messages presented to a database interactor, and by "V 123", or rather the EBCDIC representation of "V 123", in database internal file entries. Thus, just as the same object system phenomenon may be referred to by infologically different messages\(^1\), the same infological entity may be represented by different datalogical entities.

Cf figure 1. We shall say that a datalogical entity \( x \) refers to an object system entity \( z \), if and only if \( x \) represents an infological entity \( y \), which in turn refers to \( z \).

The access algorithm of a file may need to be commented upon. Naturally this algorithm may be very sophisticated with different subalgorithms corresponding to different search strategies to be used in different situations. In the definition we stated as a minimum requirement that the algorithm should be somehow capable of retrieving any identified entry of the file as well as all the entries of the file. The former requirement may be thought to imply that the file has to be allocated to a direct access memory. This is not true, however. An algorithm may consist of a lot of steps, and these steps may involve, for instance, a partial serial scan of a file. Naturally, much more efficient algorithms for retrieving single entries may be designed, if it has been decided that the file will reside on a direct access memory. Other operations, which the steps of the access algorithm of a file may involve, are table look-ups and hashing, or scrambling, of identifiers. Tables and other auxiliary data used by the access algorithms may sometimes be considered as files themselves. However, it should be noted that they are meta-files in the sense that they (usually) do not represent information about the object system proper\(^2\), but rather information about the datalogical data base itself.

Before we leave this section, we shall say a couple of words about "terms", and we shall also introduce the concepts of "term groups" and "segments".

---

\(^1\) Cf section 3.1.1

\(^2\) Cf sections 3.2.5 and 4.3.1.
The same object system entity ...

... may be referred to by ...

... different infological entities, each of which ...

... may be represented by ...

... different datalogical entities.

Figure 1
Very often each term of an entry will be the data representation of an explicit reference, i.e., a reference referring directly to a particular object system entity, like an attribute, a value, or a time.\footnote{Cf section 3.1.2.} Other possibilities are not excluded, however. For instance, for some design reason it may be practical to split the representation of an explicit reference into several terms. Some terms of an entry may contain auxiliary data of the kind mentioned above, i.e., data which do not represent information about the object system proper, but which are required by the access algorithm of the file; "entry continuation pointers" are good examples of such terms.

A **term group** is a subtuple of (not necessarily adjacent) terms of a file entry which together represent a reference. The individual terms of a term group will usually represent sub-references of the reference represented by the term group as a whole.

**Example.** Consider an entry

\[(i) \quad <t_1, t_2, t_3, t_4, t_5, t_6>\]

where

\[t_1 = "\text{CAR REG NO}"\]
\[t_2 = "\text{ABC 123}"\]
\[t_3 = "\text{COLOUR}"\]
\[t_4 = "\text{YELLOW}"\]
\[t_5 = "\text{TIME}"\]
\[t_6 = "\text{JAN 1970}"\]

The entry \((i)\) is supposed to represent the information (a-message) that a certain car has a certain colour at a certain point of time. It is obviously practical to be able to talk about the term groups
(ii) \( \langle t_1, t_2 \rangle \) = "the object term group"

(iii) \( \langle t_3, t_4 \rangle \) = "the property term group"

(iv) \( \langle t_5, t_6 \rangle \) = "the time term group"

as entities in their own right.

**Example.** The data representation of the Swedish civic registration number could be regarded as a term group

(v) \( \langle t_y, t_m, t_d, \langle t_x, t_s \rangle, t_c \rangle \)

where

\( t_y = "year of birth" \)

\( t_m = "month of birth" \)

\( t_d = "day of birth" \)

\( \langle t_x, t_s \rangle = "birth number" \)

\( t_s = "sex" \)

\( t_c = "check digit" \)

An entry segment is a subtuple of adjacent terms of a file entry. For each entry one or more segments should be defined in such a way that each term of the entry belongs to exactly one segment.

The segment concept will be needed in the file structuring discussions to be carried out in section 8.3. The idea is that entry segmentation should be a prerequisite for mapping (allocating) different parts of one and the same entry into different storage blocks of a memory. Thus, whereas the terms of a particular segment should be mapped into contiguous storage positions\(^1\), different segments of an entry may be spread out all over a memory, or even over different memories. Naturally, however, the access algorithm of the file has to be able to retrieve all segments of all entries.

\(^1\) Recall the definition of a memory as an ordered set of storage positions; section 8.2.1.
Term groups and entry segments are two different kinds of term structures. The term group structure of an entry is a reflection of the infological micro-structure of a message. By grouping the terms of an entry, the designer can make it easier for himself and others to survey the infological contents of the entry. When a designer proposes a split of an entry into several segments, he should give datalogical, efficiency-oriented reasons for this.

In figure 2 the diagram technique introduced by Bachman has been used to illustrate the relationship between the concepts defined in this section.

\[\text{\footnotesize See Bachman [201].}\]
Figure 2  "Bachman relationships" between the concepts introduced in section 8.2.2. Each arrow represents a relationship between two kinds of entities. The direction of an arrow represents the fact that an occurrence of one kind of entity consists of zero, one, or more occurrences of another kind of entity.
8.2.3 Different kinds of files

We shall define three basic kinds of files, object files, property files, and relational files, each of them characterized by the kind of entity referred to by the file entry points.¹

**Definition.** An **object file** is a file whose entry points refer to objects or \( \langle \text{object}, \text{time} \rangle \) pairs.

**Definition.** A **property file** is a file whose entry points refer to properties or \( \langle \text{property}, \text{time} \rangle \) pairs.

**Definition.** A **relational file** is a file whose entry points refer to tuples \( \langle R, o_1, \ldots, o_n \rangle \), or \( \langle R, o_1, \ldots, o_n, t \rangle \), where \( R \) is an \( n \)-ary object relation, \( o_1, \ldots, o_n \) are objects, and \( t \) is a time.

Naturally, all three kinds of files are subject to the general file definition in section 8.2.2. Thus any object, property, or relational file should be equipped with an entry description and an access algorithm fulfilling the requirements (b) and (c) stated in the general file definition. For access algorithms of relational files we state here the additional requirement that they should accept as input not only complete entry points

\[ \langle R, o_1, \ldots, o_n \rangle \text{ or } \langle R, o_1, \ldots, o_n, t \rangle \]

but also **partial entry points** which result from suppressing at least one and at most \( n \) object references in a complete entry point. As output the access algorithm should deliver all entries in the relational file whose entry points coincide with the input entry point as to the components which are not suppressed in the latter.

As can be seen from the definitions we have left the door open for a time term to be part of the entry point term group

¹ By "the entry points of a file" we mean the entry points, or primary keys, of the entries of the file. The entry points, like any data terms, are said to refer to the object system entities referred to by the references represented by the data. Cf 8.2.2, in particular figure 1.
of the entries of all kinds of files. We have chosen to do so, because it may sometimes be desirable to be able to establish files where different time versions of the same message type occur in different entries of the same file.

Example. Suppose that a data base should contain a complete history\(^1\) with respect to the message type \(<\text{PERSON}, \text{INCOME}>\). This means that the data base should contain an ever-growing number of time versions of the particular message type. Then it is a design decision to choose among the following file structuring alternatives (and others):

(a) One object file per time version of the message type. This choice implies an ever-growing number of files.

(b) One object file for all time versions of the message type, and one file entry for all time versions of the message type for each relevant object, i.e., one file entry per person. This choice implies a constant number of ever-growing entries, as long as the number of persons remains constant.

(c) One object file for all time versions of the message type, and one file entry per time version and person. This choice implies an ever-growing number of constant-length entries.

Alternative (c) requires the entry points of the file to contain a time component. Otherwise the entry points would not be unique entry identifications as required by the definition in section 8.2.2. Thus the entry points of an object file must be allowed to refer to \(<\text{object}, \text{time}>\) pairs, and not only to objects Q.E.D. Similar arguments may be carried out for property files and relational files.

In chapter 3 we defined elementary messages (e-messages) as the "atoms" of information. We also defined basic classes of

\(^1\) Cf section 5.3.1, "information substitution".
information, called elementary message type (e-message types, or e-concepts). The datalogical counterparts of e-messages and e-concepts will be called e-entries and e-files, respectively, and they are defined in the following way.

**Definition.** An **elementary file entry**, or *e-entry*, is a file entry representing an e-message.

**Definition.** An **elementary file**, or *e-file*, is a file, each of which represents e-messages belonging to one and the same e-concept.

In view of our earlier discussion of three basic kinds of files, we also realize that there are three basic kinds of e-files, namely

1. elementary object files, or **object e-files**
2. elementary property files, or **property e-files**
3. elementary relational files, or **relational e-files**

In section 3.2.3 we defined two kinds of e-concepts, namely

(a) **attribute e-concepts**, and
(b) **relational e-concepts**

The question now arises which kind of e-concepts may be represented by which kinds of e-files. The six different possibilities are displayed in figure 3. Which of them are feasible?

Each entry of a property e-file will tell what objects have or had a particular property belonging to a particular attribute. The entry-identifying property may be fundamental
<table>
<thead>
<tr>
<th></th>
<th>attribute e-concept</th>
<th>relational e-concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>object e-files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>property e-files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relational e-files</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3** Different kinds of e-concepts may be represented by different kinds of e-files
or derived according to the definitions in section 2.2.2. If the property is fundamental, the entry will represent an e-message of property type, and the property e-file as a whole will represent an attribute e-concept and nothing else.

If the entry points of a property e-file represent derived properties, the property file may sometimes represent a relational e-concept. This situation will occur when the entry-identifying properties are relation-dependent\(^1\).

Consider, for example, a property file whose entries have the structure

\[
\langle \textsc{father,} o_i, o_j, t_k \rangle.
\]

The underlining marks the entry point of the entry, and in this case the entry point refers to the relational property "to be father of \(o_i\)." According to the message represented by the entry, object \(o_j\) has this relational property at time \(t_k\). Property e-files containing entries of this kind could alternatively be thought of as representing a relational e-concept or a (derived) attribute e-concept.

The e-concept in the previous example could also be represented by a relational e-file, whose entries would then have the structure

\[
\langle \textsc{father,} o_i, o_j, t_k \rangle
\]

If we compare this structure to the structure of the property file entries above, the only difference is seen to be the scope of the entry point. However, because of earlier stated definitions, this small difference has important implications as to the capabilities of the access algorithms. Both the access algorithm of the property e-file and the access algorithm of the relational e-file would be capable of

(i) retrieving all \(<\text{father, child}>\) pairs, and

\(^1\) Cf section 2.2.2, example 5.
(ii) retrieving the father of any child.

Because of the additional requirement for access algorithm of relational files, stated earlier in this section, the latter would also have the capability of retrieving

(iii) all the children of any father

Object e-files will typically represent attribute e-concepts. For instance the entries of an object e-file could represent e-messages telling the age, income, name or address of identified persons. Conventional tape files are usually good examples of consolidated object e-files. However, like property e-files, object e-files may sometimes represent relational e-concepts as well. As a matter of fact, the "father" e-concept used in the examples above could be represented by an object e-file, too.

Thus, we have seen that at least five of the six combinations are feasible, although three of them are more "typical" than the others. Relational files will typically represent relational concepts, whereas both object files and property files will typically represent attribute concepts. If we choose to represent an attribute concept by a property file, we facilitate the answering of queries like "for what objects does attribute A take value v?" Choosing an object file representation of the property file will instead make the answering of queries like "what value does attribute A take for object o_i?" easier. In order to be able to make the correct design decision we must have some idea of the frequencies and response time requirements of different kinds of queries. Cf chapter 5 and chapter 7.

The reader may have observed that our earlier stated definitions of "e-file" and "e-entry" do not logically imply that all entries of e-files are e-entries. As a matter of fact, the entries of property e-files will mostly be non-

---

1 Cf 8.3.4.1.
elementary\(^{1}\). Consider, for example, the property e-file representing the e-concept

\[ \langle \text{PERSON, AGE} \rangle \]

The entries of this property e-file will have the structure

\[ \langle \text{AGE, } a, \ o_1, \ldots, \ o_n, \ t \rangle \]

where \( \langle \text{AGE, } a \rangle \) is the entry point, and where \( o_1, \ldots, o_n \) refer to objects having the property \( \text{AGE} = a \) at time \( t \). Obviously this entry represents a message which is a consolidation of \( n \) elementary messages

\[ \langle o_i, \text{AGE} = a, \ t \rangle, \ i = 1, \ldots, n \]

In an object e-file each of these e-messages would have been represented by a separate e-entry, each of them identified by a unique object reference, but in a property e-file this cannot be done, because of the requirement stated in the general file definition in section 8.2.2 that each entry point should uniquely identify one entry.

Sometimes even object e-files may contain e-entries. Suppose, for example, that we choose to represent the relational e-concept

\[ \langle \text{BROTHER, PERSON, MALE PERSON} \rangle \]

by an object e-file. This file would then have the following structure, entry points underlined:

\[ \langle \text{PERSON, JOHN} , \ \text{MALE PERSON, SAM} , \ t ; \text{MALE PERSON, AL} , \ t ; \langle \text{PERSON, ED} , \ \text{MALE PERSON, DAN} , \ t ; \langle \text{PERSON, EVE} , \ \text{MALE PERSON, DAVE} , \ t ; \text{MALE PERSON, TOM} , \ t ; \ldots \]\

\(^{1}\) Non-elementary entries will be called consolidated entries, or e-entries.
Note that the corresponding set of e-entries

\[
\langle \text{PERSON}, \text{JOHN} \rangle, \text{MALE PERSON, SAM} \rangle, t ;
\langle \text{PERSON}, \text{JOHN} \rangle, \text{MALE PERSON, AL} \rangle, t ;
\langle \text{PERSON}, \text{ED} \rangle, \text{MALE PERSON, DAN} \rangle, t ;
\langle \text{PERSON}, \text{EVE} \rangle, \text{MALE PERSON, DAVE} \rangle, t ;
\langle \text{PERSON}, \text{EVE} \rangle, \text{MALE PERSON, TOM} \rangle, t ;
\]

is not a feasible file, because the entry points do not uniquely identify their entries. We shall sometimes use the term "quasi-file" for a set of entries with non-unique entry points.

Before we leave this section, we shall call attention to the hypothesis stated in the very first section of this report. According to that hypothesis it should be possible to develop an integrated body of concepts, intelligible and useful both to users, who prefer the black-box view of data bases, and to designers, who have to be equipped with subtle conceptual tools for efficient creation and manipulation of complex data structures. The datalogical concepts introduced here and in the previous section are obviously greatly inspired by the infological concepts introduced in earlier chapters, and showed there to be useful in user-oriented data base discussions, in which the data base is thought of as a black box reservoir of information. Thus the desired integration has been achieved. What remains to demonstrate then, is that the datalogical concepts are sufficiently subtle to enable efficient design and construction of complex data bases.

We shall devote the rest of this report to supplying a convincing amount of this still missing evidence in support of the apostrophized hypothesis.
8.2.4 Clusters and file allocation

In section 8.3 we shall see how elementary object, property and relational files may be combined into more complex file structures. These file structures should then be mapped, or allocated, into accessible memories\(^1\), which may be fairly complex structures of storage positions, blocks, tracks, cylinders, and so on. The allocation of a file structure to a memory may thus be a very complex mapping. Suppose now that we want to change the mapping. The reason for this may be that the file structure has to be changed because of the interaction between the data base and its environment\(^2\). Alternatively the reason may be that the storage structure is changed as the result of a modification of the hardware configuration, or because of new demands from other applications competing for the computer resources used by the data base system. Anyhow, changing the file \(\rightarrow\) storage mapping may be very inefficient and may even generate serious errors, unless it has been carefully planned for at data base design time. There is a remedy which is used, to some extent, in today's operating systems, and which seems to be of such general importance as to motivate a position among the basic concepts discussed in this report. This remedy will here be labeled "cluster" and is related to such concepts as "area" used by the CODASYL Data Base Task Group\(^3\) and "data set" used by IBM.

**Definition.** A **cluster** is a triple \(\langle x, y, z \rangle\) where

\[(a)\]

\(x\) is an ordered set of **cluster blocks**, each of which is a set of **cluster positions**, and is identified by its ordinal number within the cluster

\[(b)\]

\(y\) is a **cluster description** describing how to rearrange the cluster positions of the cluster blocks into file entries and file entry segments\(^4\)

---

\(^1\) Cf section 8.2.1.

\(^2\) Cf chapter 5.

\(^3\) Cf reference [75] in the bibliography.

\(^4\) As is further discussed below, each cluster block may contain entries and entry segments belonging to different files.
(c) z is an access algorithm, which, given a natural number i, delivers the cluster block identified by i for further processing

When a cluster block has been retrieved, the file entry retrieval process will

(i) use the cluster description in order to select the cluster positions of the block which are contained in the requested entry (segment)

(ii) repeat (i) until all requested segments have been retrieved

(iii) use the file entry description in order to rearrange the retrieved entry or entry segments into "data base normal format"

The main idea behind the cluster concept is to split the complex file + storage mapping into simpler file + cluster and cluster + storage sub-mappings, thereby making it possible to separate administration of storage from administration of files. Even in the future, storage administration will probably to a large degree be the responsibility of the operating system of the computer, particularly if the computer system has to be shared by several applications of which the data base system is only one. File administration, on the other hand, will be the responsibility of the data base system alone.

Figure 4 visualizes that both sub-mappings, file + cluster and cluster + storage, may in general be "many-to-many". Thus, for example, different entries of the same file and even different segments of the same entry may in principle be allocated not only to different blocks within the same cluster, but even to

---

1 This further processing will be controlled by a file access algorithm. Thus a cluster block retrieval process will be a sub-process of a file entry retrieval process.

2 Of the general file definition in section 8.2.2.
Figure 4  The file → storage mapping and its sub-mappings
file → cluster and cluster → storage.
different clusters. On the other hand, entries and entry segments from different files may very well be allocated to the same cluster block. Naturally there always has to be very good reasons for making design decisions resulting in complex mappings of the kind mentioned. In connection with the discussion of "file overlapping" in section 8.3, we shall see on what grounds relatively complex file + cluster mappings may be defended.

The cluster + storage mapping may in principle be equally complex as the file + cluster mapping. However, today’s operating systems do not allow particularly complex mappings between the set of cluster identifications and the set of physical memory addresses, and it is hardly to be expected, or even to be desired, that this situation should be substantially changed in the future.

---

1 Cluster identification is "relative block number within data set" in the IBM terminology.
8.3 File structuring operators

File structuring is a part of the design process which starts with a set of e-concepts, specified in a particular infological model, and ends up with a more or less complex structure of files to be allocated to available memories. One requirement upon the resulting file structure is that it should "fit" the available storage structure, i.e., it should at all be possible to map the file structure into the memories at hand or to be acquired. Another requirement is that the allocated file structure should be as good as possible with regard to the goals of the data base system and the methodological, personal, and economical resources available for the structuring work.

In this section we propose a set of file structuring operators. It is our hypothesis that it should be possible to describe the transformation of any set of e-concepts into any feasible file structure of practical importance by means of a sequence of these operators. There are several reasons why it is highly desirable that we be able to do this. One is that it enables us to solve any file structuring problem in small, perceivable steps. With a clearly defined "language" for file structuring, it will also be much easier for the designers to communicate ideas among themselves, and to document their analyses and decisions in a detailed and precise way. It might also be possible to build a comprehensive, normative file structuring theory in terms of the elementary operators to be presented here, a theory which would be less partial than most theoretical analyses are today because of the lack of a general, precisely defined framework. Moreover, such a general, formal framework will be absolutely necessary if computers are going to be able to solve total file structuring problems, either at pre-operation time, or dynamically as a regular data base maintenance task. In summary, a file structuring language and theory based upon the operators proposed below should be of potential value.

\[1\] By a partial analysis we mean an analysis which concentrates upon one aspect of the (file structuring) problem and more or less forgets about other aspects and possibilities.
(a) for individual file designers in problem-solving situations  

(b) as a communication tool for designers  

(c) for documentation purposes  

(d) for the development of a general, normative file structuring theory  

(e) in computer-aided design and redesign of file structures  

(f) in file maintenance efforts automatically controlled and performed by a data base management system  

We have grouped the proposed set of file structuring operators into five subsets, namely  

(1) file establishment operators  

(2) intra-entry structuring operators  

(3) inter-entry, intra-file structuring operators  

(4) inter-file structuring operators  

(5) file allocation operators  

Each of these subsets of operators will be discussed in separate sections below.
8.3.1 File establishment

The first file structuring step consists in establishing a feasible initial file structure. This file structure need not fulfill all design goals, but it should be feasible in the sense that it seems reasonable to believe that a few applications of the structuring operators will lead to a satisfactory solution of the structuring problem. Nor is it necessary or even desirable to define in detail the entry descriptions and access algorithms of the initial file structure, as these entities will anyhow change during the continued structuring work. As will be exemplified later there is a kind of complementary relationship between the entry description and the access mechanism on the one hand, and the contents and structure within and between the file entries on the other. When one is changed, the other will have to be changed as well.

The file establishment design step involves the decision, mentioned in section 4.4, concerning the demarcation between the data base nucleus and the rest of the data base information contents. This decision affects all e-concepts which are derivable from other e-concepts by means of the derivation rules in the data base schema.\(^1\) We shall say that an e-concept, which is made part of the data base nucleus, is established as a real file, and that an e-concept, which is made derivable by means of schema algorithms, is established as a virtual file. For infological reasons discussed in 4.4 it could sometimes be a wise decision to establish an e-concept both as a real file and as a virtual file.

What are then the datalogical advantages and disadvantages of a virtual file compared to a real one? The main datalogical advantage achieved by algorithmizing a file is of course the decrease of space needed to represent the messages

\(^1\) The information analysis during the specification process resulting in the particular infological model underlying the data base should show which of all the e-concepts are derivable from others.
of the database. This decrease may be so substantial that it implies secondary effects, like a decrease in average access time within the database as a whole.

On the other hand, the retrieval of a virtual entry will ordinarily be more resource-consuming than the retrieval of a real file entry. There are two reasons for this. Firstly, the retrieval of a virtual entry will usually imply the retrieval of several real entries, because the logical precedents of the algorithmized message may very well be spread over several files. Secondly the breaking down of transactions concerning virtual entries into transactions concerning real entries, and conversely the assembly of retrieved real entries into requested virtual entries will imply a lot of processing and, very likely, accesses to catalogues, tables and other auxiliary data on secondary storage.

If we decide to establish a particular e-concept as a real file, the next design decision will be to choose one of the three basic file types defined in section 8.2.3. We may establish the e-concept as an object e-file, a property e-file, or a relational e-file. As was discussed in section 8.2.3, this choice is not at all uniquely determined for all kinds of e-concepts but has to be carefully considered in view of such parameters as expected frequencies and response time requirements for different kinds of database queries.

We said above that it could sometimes be well motivated to establish an e-concept redundantly both as a virtual file and as a real one. There are also good infological and datalogical reasons for duplicating real files in certain situations. One typical example is that it may lower the average response time for a certain mix of database queries, if we make the design decision that a particular e-concept should be established both as an object e-file and as a property e-file. One should always remember, however, that each duplication will not only increase space demand, but it
will above all increase the problems of database maintenance, including addition, deletion, and substitution transactions.¹

If there are to be several time versions of a particular e-concept in the data base, we have to decide whether to establish one e-file per time version, or one e-file with separate e-entries for different time versions, or one e-file with one e-entry per object. This design problem was discussed in connection with an example in section 8.2.3. An important circumstance here is whether the data base is supposed to contain a complete history with respect to the e-concept, or if, say, the ten most recent time versions will do.

In summary then, there are the following file establishment operators to be applied and combined by the file structurer:

- establish e-concept x as a virtual file

- establish (time version y) of e-concept x as an object e-file

- establish (time version y) of e-concept x as a property e-file

- establish (time version y) of e-concept x as a relational e-file

From time to time, as the file structuring process goes on, the designer will probably also use the inverses of the establishment operators. Moreover, this possibility will certainly be exploited by manual, computer-assisted, or automatic maintenance processes which dynamically restructure the data base when it is in operation.

¹ Cf chapter 5.
8.3.2 Intra-entry structuring

In this section we shall study how it is possible by means of four different operation types, namely

- entry compression

- reordering of terms

- entry chopping, or segmentation

- ordering of segments

to transform the internal structure of the entries of a file into a format which may be more efficient than the structure given by the "data base normal format". The operators discussed here may not only be applied to entries which conform to the normal format but also to entries which have already been subject to other structuring operations\(^1\) during earlier design stages. This also implies that the entries operated upon need not necessarily be elementary entries of elementary files.

\(^1\) These operations may be of any kind described in this or later sections.
8.3.2.1 Entry compression

There are several variations of the entry compression operation. The purpose is always the same, however, to reduce the volumes of data to be stored and transmitted. There is another characteristic, which is typical of most forms of information and data compression. This characteristic may be expressed as a general rule: compression of one entity usually has to be balanced by increased sophistication with another. For instance, communication between people may be much more laconic if they have a common, extensive frame of reference. The more elaborate the frame, the less elaborate the framed has to be, one might also say. In the case of entry compression the framed is of course the entry itself, whereas the frame is the entry description and, to some extent, the algorithm which uses the entry description in order to interpret the contents of the entry.

There are two variations of entry compression to be treated in this text. One, compression by suppression, attracts interest in situations where "data missing", "attribute not relevant" and similar conditions are frequently appearing within the entries of a file. For any term which should occur with some value in all entries of a file, we can define one of the possible values as a "default value", which may be suppressed instead of being explicitly represented in the entries to which it pertains. If this form of entry compression is at all worth-while, it will probably be most advantageous if the most frequent term value is chosen to be suppressed.

After compression by suppression the entries of the file will inevitably be of variable length. As is well-known to all practitioners and as also follows from the general rule stated above, this leads to a more complicated entry administration. The entry description has to be expanded, and the access mechanism may be less efficient in finding and interpreting entries.

The other variation of the compression operation to be mentioned is called compression by extraction. It is applicable
when the entries of a file contains a term, which is
predetermined to have the same value throughout the file.

*Example.* Consider the file

```
<PERSON, JOHN>, <WEIGHT, 70>, 1971
<PERSON, CAROL>, <WEIGHT, 50>, 1971
<PERSON, JIM>, <WEIGHT, 88>, 1971
<PERSON, AL>, <WEIGHT, 62>, 1971
<PERSON, EVE>, <WEIGHT, 65>, 1971
```

From the entries of this file we may extract the first term,
which has the constant value "PERSON", the third term ("WEIGHT"),
and the fifth term ("1971").

The extraction operation implies obvious storage and trans-
mission gains. The disadvantages of the operation are often
unimportant. Anyhow they are less striking than in the case
of suppression, because extraction conserves the fixed-format
of the original entries, if they are fixed-format in the first
place, that is.

However, by extracting terms from the entries of a file we
may destroy *self-describing* properties of the entry. An entry
is self-describing to the extent that the entry description
need not be consulted in order to interpret the entry. Self-
describing entries may be very useful under specific circum-
stances. Consider, for instance, a file system consisting
of numerous small files, serving an environment that submits
queries, each of which requires fast response and access
to many files. Having to access a lot of entry descriptions
may then be time as well as space consuming; there may not
even be enough room in main memory for all relevant entry
descriptions, with swapping as an efficiency-degrading result.

Even if the original entries are not self-describing, extraction
will, according to the general rule stated in the beginning of
this section, cause the entry descriptions to grow. If the
entries are stored on a cheap storage medium, whereas the
cost of storing the entry descriptions is relatively high, compression by extraction may not be profitable even from sheer space point of view, although the total number of stored bits is reduced.

To summarize: compression by extraction will normally improve the structure of the database, but the exceptions from this rule must be kept in mind.
Reordering of terms

According to the definition in section 8.2.2 entries are tuples of terms, i.e. the terms of an entry are considered to be ordered. In some situations, e.g. if we are about to split the entries into segments, the initial ordering of terms may be inadequate. Then the reordering-of-terms operation has to be carried out. Beside the internal restructuring of entries it has the effect of changing the entry description of the file appropriately.

Reordering the terms within one and the same entry segment, will rarely have any important performance consequences. Two potential exceptions should be mentioned, however. Firstly, if there is a term group structure and the group is sometimes processed as a unit, processing may be faster if the terms of the group are contiguously arranged. Secondly, if a few terms of a segment are often processed together, e.g. because of the particular data requirements of the most frequent transaction types, it may be advantageous to allocate these terms to a contiguous secondary storage area, provided that parts of storage blocks, and not only full storage blocks, may be read and transmitted by the access mechanism of the memory.¹

¹ Cf section 8.2.1.
8.3.2.3 Entry chopping

The file structuring operation which splits an entry into several segments will be called chopping, or segmentation. By chopping the entries of a file into segments we prepare the file for two things, multi-block allocation and file overlapping. These operations are described in other sections, but a brief review may not be out of place here. Multi-block allocation of an entry means the mapping of the segments of the entry into different cluster blocks. Two files are overlapping, or have been overlapped with each other, if the entries of one file share data with the entries of the other. This in turn implies that the overlapping entries have segments, which are mapped into the same cluster block. See also figure 5.

An unchopped entry contains but one segment, i.e. one contiguous tuple of data terms. A chopped entry consists of several segments. One of the segments, the initial segment contains the entry point of the entry. All other segments should be retrievable by an segment collecting algorithm, starting from the initial segment. The segment collecting algorithm is a sub-algorithm of the access algorithm of the file. Very often the collecting algorithm makes use of data pointers, i.e. auxiliary data terms, which are added to the segments when the entries are chopped, and which are intended to contain the address of the next segment. Other arrangements are also possible though. The segment collecting algorithm may use the value of the entry point or the address of the initial segment to calculate the address of the next segment.¹ We may talk about virtual pointers in such cases.

¹ Cf. section 8.3.3.1 and Thorburn [191].
**Figure 5a** Five cluster blocks, two segments, one entry, one file.

**Figure 5b** Seven cluster blocks, two files, one, with two single-segment entries, the other, with one entry consisting of four segments, two of which overlap with the entries of the other file.
8.3.2.4 Ordering of segments

After an entry has been chopped into segments, there must exist a kind of minimum ordering between the segments, which ensures the retrievability of the collection of segments belonging to the entry. This ordering is implied by the segment collecting algorithm, and it conserves the entry as a logical entity.

In many cases, however, it may be efficient with regard to the goals of the design work to add more structure to the entries than is logically necessary. For instance, in order to ensure entry retrievability, chaining the segments will always be logically sufficient, but bundling the segments instead might considerably reduce the average number of accesses necessary to retrieve requested terms. These two basic methods of ordering the segments of an entry are visualized in figure 6. Hybrid structures are easily conceivable and may be advantageous under special circumstances. For example, if the number of segments varies much between different entries of the same file it may be inefficient to reserve pointer space for "the worst case" in all initial segments. A "variable-length chain of fixed-span bundles"¹ may be more efficient and easier to administrate.

The advantages and disadvantages of different intra-entry orderings are similar to those of the corresponding inter-entry ordering. The latter will be treated in the next section. It is difficult, however, to discover any gains obtainable from entering inverse pointers between entry segments, except when the segments may be entered without passing the initial segment, which may be possible if several files have been overlapped.

¹ Cf figure 7.
Figure 6a  Chain ordering of entry segments

Figure 6b  Bundle ordering of entry segments

Figure 6  Two basic methods of ordering the segments of an entry
8.3.3 Inter-entry structuring

8.3.3.1 Ordering of entries

According to the definition in section 8.2.2, the access algorithm of a file should be able to retrieve all entries of the file as the result of one request, which does not have to submit any entry point values. Thus, assuming that the access algorithm works deterministically, there has to be at least an implicit ordering of the entries of a file.

For example, the file may be explicitly or implicitly ordered as a chain or a bundle of entries. Many other structurings, built up from these elementary structures are also possible. See figure 7.

The requirement that all entries of a file should be retrievable as the result of one request is not the only reason for ordering the entries. For instance, the search for a particular entry proceeds much faster, if the entries are ordered into a tree according to the values of the entry point, than if the whole file has to be scanned "linearly" until a match is found (which will happen after half the file has been searched on the average).

Ordering an object file might also be a way of representing object relations. Such ordering may often exist within, and also between, object files without explicit decisions by the designer. Suppose, for instance that an object file entry contains the representation of a relational-type message. Then this entry points, by means of a name pointer to another entry of the same or another file.

---

1 By an implicit ordering of the entries of a file, we mean an ordering of which there are no visible signs in the very entries. Implicit orderings are revealed by the way the access algorithm scans the file, or by the way it retrieves single entries. Explicit orderings are materialized in the file entries by name or address pointers.
Figure 7a  Chain of bundles

Figure 7b  Bundle of chains

Figure 7c  Bundle of bundles

Figure 7  Different structures formed by recursive use of the two basic structures: chains and bundles
Example. Consider the object e-file

<table>
<thead>
<tr>
<th>entry point</th>
<th>other terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨PERSON, JIM⟩</td>
<td>⟨PERSON, JOHN⟩, FATHER</td>
</tr>
<tr>
<td>⟨PERSON, JOHN⟩</td>
<td>⟨PERSON, SAM⟩, FATHER</td>
</tr>
</tbody>
</table>

The entry ⟨PERSON, JIM⟩ points to the entry ⟨PERSON, JOHN⟩ in the same object e-file by name. The name pointer has a double function. Firstly, it tells the name of the father of the person identified by the entry point. As a matter of fact, the entry thus represents a property-type "name-of-father" message, too. Secondly, the name pointer points, through the access algorithm of the file, to the entry which contains more information about the person's father than his name.

The ordering of an object file according to the values of an attribute (other than the entry point) could facilitate the search for objects having certain properties. A similar advantages could be attained by establishing a property file. The choice between the two alternatives may deserve careful consideration.

If we establish an explicit inter-entry ordering we have to maintain that structure. Beside space consumption, this is the main potential problem connected with file ordering. Strangely though, it may be advantageous to use more pointers to help administrate the existing ones. For example, inverse pointers may be very helpful when an entry is to be deleted from an explicitly ordered file.

Discussing file ordering we cannot just pass over a problem, which often causes vivid debates, but which is not seldom solved on more or less irrational grounds. We think of the problem whether a direct memory file should be allocated and accessed via a hashing algorithm or via an algorithm
based upon (i) an explicit ordering of the file entries in accordance with ascending or descending entry identifiers, and (ii) auxiliary index tables, the entries of which are ordered in the same way as the main file and point to every n:th entry in the latter.

The two methods mentioned for allocating and accessing a direct memory file will be called pseudo-direct organization, PDO, and indexed-sequential organization, ISO. If we choose the PDO method, we shall have to solve the following problems:

- A good hashing algorithm must be found. We usually want an algorithm which is not too slow in terms of processing time, and which generates all available cluster addresses with equal probabilities. There are plenty of algorithms described in the literature which fulfill these requirements. A modified random number generator will usually be all right.

- Hashing algorithms will generate synonyms, i.e. different entries will be mapped to the same primary address. A complementary algorithm must be found, which resolves synonym conflicts and generates secondary addresses both in allocation and retrieval situations. There are a lot of methods for this purpose, too. They may be grouped into four classes:

1 Linear search. As secondary addresses are generated successively the addresses following immediately upon the primary address. This method is simple but inefficient because of systematic overlapping between different suites of secondary addresses.

2 Random search based upon the primary address. Secondary addresses are generated by feeding another hashing algorithm

---

1 Earlier in this section we defined an explicit file ordering as an ordering which is materialized in the file entries by data pointers. However, such data pointers will often be suppressed (cf 8.3.2.1) when they take their default value, which is usually interpreted as the address of the entry which is the successor of the present entry according to the ordering of the cluster blocks (cf 8.2.4). Even orderings based upon such tacitly understood "next pointers" will be regarded as explicit orderings.
with the primary address as input. This method is better than linear search but still has the disadvantage of generating identical suites of secondary addresses for entries with the same primary address.

3 Random search based upon the entry identifiers. Secondary addresses are generated by feeding another hashing algorithm with the original entry identifiers. This method is better than the two previous ones, because only random (non-systematic) interference occurs between different suites of secondary addresses.

4 Pointer-linked chains of synonyms. The synonyms are linked to each other by means of data pointers. The method consumes some space for the pointers, but it systematically avoids interference between different suites of synonyms. The retrieval efficiency of the method could be further improved by dynamic rearrangement of the synonym chains, so that the most frequently requested entries are placed in the beginning of the chains. Bundle orderings and hybrid structures could also be utilized instead of pure chains.

- With PDO the unutilized space will be spread randomly all over the cluster to which the file is allocated. Cluster blocks which do not contain file entries must be kept track of, e.g. by means of "flags" or by means of a bit map of the cluster. This is necessary both for the functioning of the hashing algorithms for allocation and retrieval, and for the functioning of the "retrieve-all-entries-of-the-file" algorithm.

There is an extensive literature about problems which are related to the problems of PDO-files. An early paper which is still well worth reading was written by Buchholz [73]. Summarizing discussions, analytical formulas, sample tabulations of relevant efficiency figures, and further references may be found in Sundgren [182], and Thorburn [190]. A brief review in English is given in Arvas [42]. As a general comment one may

1 The titles of the articles do not always reveal that they are relevant to file organization problems.
say that any competent designer who is prepared to think about
the PDO problems for a couple of hours will be able to develop
satisfactory solutions to them. This investment in knowledge
may seem small compared to other costs in projects where file
organization is at all a problem worth considering. Yet the
knowledge threshold seems to be prohibitive in many practical
design situations.

The problems of ISO are not less important than those of PDO.
However, the manufacturers, led by IBM, have devoted more
energy to the former and developed standard software in order
to "conceal" them from their customers. In summary the problems
of ISO are:

- The ordering of entries has to be maintained. This implies
  more complex algorithms for addition and deletion of entries
  than if the file had not been ordered.

- If the file is dynamic, i.e., if it is frequently hit by
  addition and deletion transactions, there is a great risk of
  quickly degrading performance. This threat has to be met by
  an appropriate combination of

  (i)  "space slacks" which are introduced all over the
       cluster structure to which the file is allocated

  (ii) regular reorganizations of the whole file

- The auxiliary index table has to be properly maintained.
  Moreover, this table requires extra space and implies at least
  one extra access per retrieved entry.

More detailed comparative discussions of PDO and ISO may be
found in Nijssen [183], [184], and Sundgren [190].

An interesting variation of the ISO method has been suggested
by Ghosh & Senko [77]. They propose the drastic remedy of
simply removing the auxiliary index table. Instead they use
interpolation in order to retrieve the entries of the file. The
results they have achieved by this method are surprisingly good. For example, the expected number of accesses is as low as 1.13 when

- the number of entries = 500

- the entry identifiers are randomly distributed over an interval of length 10 000

- the file is allocated to a cluster consisting of 10 "tracks", each of which contains 50 entries
8.3.3.2 Intra-file consolidation

There are two cases of intra-file entry consolidation worth mentioning.

Firstly, suppose we have established a quasi-file, i.e., a file where several entries have the same entry point value. Then, as we have pointed out earlier, it is commendable to transform the quasi-file into a proper file by entry consolidation.

Example. Consider the e-concept \(<\text{ENTERPRISE, INDUSTRY}\>$, where the attribute INDUSTRY is multiple-valued as one enterprise may belong to several industries. Before consolidation we may have the quasi-file

<table>
<thead>
<tr>
<th>ENTERPRISE</th>
<th>INDUSTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>STEEL</td>
</tr>
<tr>
<td>B</td>
<td>TEXTILE</td>
</tr>
<tr>
<td>C</td>
<td>FOOD</td>
</tr>
<tr>
<td>D</td>
<td>PULP</td>
</tr>
<tr>
<td>A</td>
<td>MANUF</td>
</tr>
<tr>
<td>C</td>
<td>DISTRIBUT</td>
</tr>
<tr>
<td>D</td>
<td>PAPER</td>
</tr>
<tr>
<td>D</td>
<td>TIMBER</td>
</tr>
</tbody>
</table>

After entry consolidation within the file we obtain a proper file with variable-length c-entries:

<table>
<thead>
<tr>
<th>A</th>
<th>STEEL</th>
<th>MANUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>TEXTILE</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FOOD</td>
<td>DISTRIBUT</td>
</tr>
<tr>
<td>D</td>
<td>PULP</td>
<td>PAPER TIMBER</td>
</tr>
</tbody>
</table>

Secondly, suppose we have an \(<\text{object, time}\>$-pair entry point file, i.e., a file where there may be several entries concerning the same object and the same attribute (or object relation), separated only by the time component of their entry point. If we consolidate the entries of such a file per object over time, we obtain c-entries, which may be identified by single object identifications.
8.3.4  Inter-file structuring

8.3.4.1  Consolidation

The inter-file consolidation operator is defined differently for the three basic kinds of files.

Two object files, $F_1$ and $F_2$, are consolidated into a new file $F_3$ in the following way.

(a) Matching entries, $e_1$ and $e_2$, of the two files give birth to a $c$-entry $e_3$ of the consolidated file $F_3$. The term tuple of $e_3$ is built up by the term tuple of $e_1$ followed by the term tuple of $e_2$, from which terms identical with $e_1$ terms have been eliminated, though. ($e_1$ and $e_2$ have at least the entry point term or term group in common)

(b) Non-matching entries are transferred to the new file after "entry expansion" (see example below)

(c) $F_1$ and $F_2$ are cancelled from the file system and $F_3$ is established. The entry descriptions and access algorithms of $F_1$ and $F_2$ are replaced by a new entry description and access algorithm belonging to $F_3$.

Example. Consider the object files

$F_1 = \langle \text{PERSON, INCOME, AGE, 1970} \rangle$, and
$F_2 = \langle \text{PERSON, AGE, IQ, 1970} \rangle$

$F_1 = \begin{array}{|c|c|c|}
\hline
\text{PERSON} & \text{INCOME} & \text{AGE} \\
\text{JIM} & 50 & 30 \\
\text{JOHN} & 62 & 57 \\
\text{BETTY} & 35 & 40 \\
\text{DAN} & 20 & 19 \\
\text{MARY} & 45 & 55 \\
\hline
\end{array}$

\footnote{Two entries are matching if and only if their entry points are identical.}
\[ F_2 = \begin{array}{|c|c|c|} \hline \text{PERSON} & \text{AGE} & \text{IQ} \\ \hline \text{DAN} & 19 & 115 \\ \text{BETTY} & 40 & 95 \\ \text{SARAH} & 5 & 105 \\ \text{JIM} & 30 & 100 \\ \text{MARY} & 55 & 145 \\ \hline \end{array} \]

After consolidation we obtain

\[ F_3 = \text{cons} \left( F_1, F_2 \right) = \langle \text{PERSON}, \text{INCOME}, \text{AGE}, \text{IQ}, 1970 \rangle \]

\[ F_3 = \begin{array}{|c|c|c|c|} \hline \text{PERSON} & \text{INCOME} & \text{AGE} & \text{IQ} \\ \hline \text{JIM} & 50 & 30 & 100 \\ \text{JOHN} & 62 & 57 & ? \\ \text{BETTY} & 35 & 40 & 95 \\ \text{DAN} & 20 & 19 & 115 \\ \text{MARY} & 45 & 55 & 145 \\ \text{SARAH} & ? & 5 & 105 \\ \hline \end{array} \]

Remark. The "JOHN" and "SARAH" entries are non-matching entries of \( F_1 \) and \( F_2 \) respectively. They have been expanded in \( F_3 \). The question marks resulting from the expansions call for further investigation. Do they represent missing but relevant values, or are the attributes not relevant to all objects in the new object domain (= the union of the old ones)?

Pseudo-consolidation (by entry bundling) is implied as soon as two object files have identical entry points. Example: two person files, both with civic registration numbers as entry points.

Two relational files may be consolidated by application of the natural join relation operator\(^1\). The consolidated file will then contain the original files as projections\(^1\).

---

\(^1\) Codd [76], [77], [78] has developed an extensive set of operators for manipulating general relations. Some of them, like "the natural join", seem to be very useful even with our slightly different approach to data base design and file structuring.
Example. Consider the relational files

\[ R_1 = \text{PRODUCE} = \]

<table>
<thead>
<tr>
<th>ENTERPRISE</th>
<th>ARTICLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ R_2 = \text{CONSUME} = \]

<table>
<thead>
<tr>
<th>ENTERPRISE</th>
<th>ARTICLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ R_3 = \text{cons}(R_1, R_2) = \]

<table>
<thead>
<tr>
<th>ENTERPRISE</th>
<th>ARTICLE</th>
<th>ENTERPRISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>E</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>E</td>
</tr>
</tbody>
</table>

Two property files \( P_i \) and \( P_j \) may be consolidated into a new property file \( P_k \) in the following way. Suppose \( P_i \) and \( P_j \) contain entries corresponding to the properties \( p_{i1}, \ldots, p_{im} \) and \( p_{j1}, \ldots, p_{jn} \) respectively. We may form \( r = \max \) conjunctive properties \( p_{k1}, \ldots, p_{kr} \) corresponding to the elements of the Cartesian product \( \{ p_{i1}, \ldots, p_{im} \} \times \{ p_{j1}, \ldots, p_{jn} \} \). \( p_k = \text{cons} \{ P_i, P_j \} \) should contain one entry for each of the \( r \) conjunctive properties, which is had by any object referred to in \( P_i \) och \( P_j \).
Example. Consider the property files, $P_i$ and $P_j$, corresponding to the e-concepts $\langle \text{PERSON, SEX} \rangle$ and $\langle \text{PERSON, MARITAL STATUS} \rangle$

$$
\begin{array}{|c|c|}
\hline
\text{SEX} & \text{PERSONS} \\
\hline
\text{MALE} & X, Y, Z, U, W \\
\text{FEMALE} & A, B, C, D, E, F \\
\hline
\end{array}
$$

$$
\begin{array}{|c|c|}
\hline
\text{MARITAL STATUS} & \text{PERSONS} \\
\hline
\text{SINGLE} & X, U, C \\
\text{MARRIED} & Y, Z, W, B, E, F \\
\text{DIV/WID} & A, D \\
\hline
\end{array}
$$

After consolidation we obtain

$$
\begin{array}{|c|c|}
\hline
\text{PROPERTY} & \text{PERSONS} \\
\hline
\text{MALE SINGLE} & X, U \\
\text{MALE MARRIED} & Y, Z, W \\
\text{FEMALE SINGLE} & C \\
\text{FEMALE MARRIED} & B, E, F \\
\text{FEMALE DIV/WID} & A, D \\
\hline
\end{array}
$$

Remark. The consolidation operators we have defined in this section for different kinds of files are rather unsimilar to each other. Perhaps one should keep them apart by naming them differently, e.g. joining of relational files and (conjunctive) combination of property files instead. "Consolidation" would then be reserved for object files.

There are several reasons why it may often be advantageous to consolidate (join, combine) several files into one:

- The consolidated file will occupy less secondary storage space than the original files did as a whole.\(^1\)

- Consolidation reduces the number of files, which may be essential if the file structure is to be allocated to conventional tape memories.

\(^1\) This is a rule with exceptions. It is generally invalid for relational files.
- If the consolidated file occupies less secondary storage space, a serial scan of the consolidated file will cause less data transmission between secondary and primary storage than a serial matching of the original files. Less total buffer space will be needed, too.

- Consolidation will reduce the number of accesses needed to process data base transactions which concern more than one of the original files. On the other hand, if such transactions are rare, consolidation will be detrimental, because the transmitted data volumes as well as buffer space demand will increase without any gain as to the number of accesses.
8.3.4.2 File overlapping

We start with a linguistic remark. The verb "overlap" is both transitive and intransitive. The phrase "overlapping files" is thus ambiguous, as it may denote both the action (structuring operation) of overlapping files with each other and the state which prevails among a set of files after they have been operated upon by the overlap structuring operator. We intend to live with this ambiguity.

We overlap two files by overlapping entries of one file with entries of the other. Two entries, \( \varepsilon_1 \) and \( \varepsilon_2 \), are overlapping if they have some common data, i.e., if one segment \( \sigma_1 \) of \( \varepsilon_1 \) and another segment \( \sigma_2 \) of \( \varepsilon_2 \) are to be mapped by their respective file + cluster allocation functions into overlapping positions of the same cluster allocation block. The situation is schematically illustrated in figure 8.

There are two reasons for overlapping files. The first reason is that of space economy. By overlapping one entry with another we reduce space consumption by the size of the common cluster block positions. On the other hand, if chopping is a necessary condition for overlapping (which it often is, if an object file is overlapped with a property file) the pointers between segments consume space (or other resources, if virtual pointers are used). We must also remember that a chopped entry requires more accesses, if it is to be retrieved, than a contiguous entry does.

The other reason for overlapping files is that of shorter response times for certain transaction types. This is possible because overlapping files implies the creation of new access paths to the overlapping entries. Consider again the overlapping entries \( \varepsilon_1 \) and \( \varepsilon_2 \) of fig 1. Suppose that we have just retrieved segment \( \sigma_2 \) of entry \( \varepsilon_2 \) and that we then, e.g., because of the inherent structure of the transaction being processed, wish to retrieve segment \( \sigma_1 \) of entry \( \varepsilon_1 \). If the two files had not been overlapped, we would then have been forced to proceed along
entry $e_2$ of file $F_2$

cluster block $i$

entry $e_1$
of file $F_1$

$e_1$ and $e_2$ are overlapping since $e_1 \cap e_2 = \sigma_1 \cap \sigma_2 = \text{not empty}$

Figure 8 Overlapping files
the "official" access path and use the access mechanism of file $F_1$. Now, as the files are overlapping, we know that segment $a_1$ is in the same access block as $a_2$, which is probably still in primary storage. We may thus save an access each time this particular transaction type occurs.

**Example.** Suppose $F_1$ is an object file containing names and addresses of people, and suppose that $F_2$ is a property file, the entries of which contain the identifications (civic registration numbers) of people having a particular occupation. The overlapping field XXXX (cf fig 1) might then contain the identification of, say, a particular baker. It is then obvious that the overlapping file structure facilitates the processing of query transactions like: "list the names and addresses of all bakers". More examples will be analyzed in chapter 9.

What about the similarities and differences between file consolidation and file overlapping? The two structuring operations are similar in the sense that they both bring together, integrate physically and logically, two or more files. But whereas consolidation means the death of the old files and the birth of a new, consolidated file, overlapping preserves the logical independence between the overlapping files: they keep their respective entry descriptions and access algorithms, although, as we have seen, they may also take advantage of each other's facilities, e.g. each other's access algorithms.

There are other differences between the two operators as well. The consolidation operator may only operate upon files of the same kind, e.g. two object files, two property files, or two relational files. As to the overlap operator, the following two argument combinations seem to be of greatest practical impact:

(a) \(<\text{object file, property file}\> \\
(b) \(<\text{relational file, object file}\> \)
Example: Consider the relational file $F_1 = \text{"TRADE 1970"}$:

<table>
<thead>
<tr>
<th>EXP COUNTRY</th>
<th>IMP COUNTRY</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWEDEN</td>
<td>BRITAIN</td>
<td>PULP</td>
</tr>
<tr>
<td>JAPAN</td>
<td>GERMANY</td>
<td>WATCHES</td>
</tr>
<tr>
<td>JAPAN</td>
<td>USA</td>
<td>TV-SETS</td>
</tr>
<tr>
<td>YEMEN</td>
<td>SWEDEN</td>
<td>OIL</td>
</tr>
<tr>
<td>SWEDEN</td>
<td>USA</td>
<td>CARS</td>
</tr>
<tr>
<td>USA</td>
<td>SWEDEN</td>
<td>CARS</td>
</tr>
<tr>
<td>FINLAND</td>
<td>BRITAIN</td>
<td>PULP</td>
</tr>
</tbody>
</table>

Consider also an object file $F_2$, whose objects are defined as object tuples of the "TRADE" object relation, and whose attributes are "VALUE" and "VOLUME". Thus, the objects of this file are so-called compound objects of transaction type$^1$, and we give them names according to the following pattern:

\[
\text{SBP} = \langle \text{SWEDEN}, \text{BRITAIN}, \text{PULP} \rangle \\
\text{JGW} = \langle \text{JAPAN}, \text{GERMANY}, \text{WATCHES} \rangle \\
\]

etc. The file "VALUE AND VOLUME OF TRADE 1970" may then look as follows.

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>VALUE</th>
<th>VOLUME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBP</td>
<td>$g \ a$</td>
<td>t tons</td>
</tr>
<tr>
<td>JGW</td>
<td>$g \ b$</td>
<td>u units</td>
</tr>
<tr>
<td>JUT</td>
<td>$g \ c$</td>
<td>v units</td>
</tr>
<tr>
<td>YSO</td>
<td>$g \ d$</td>
<td>w m$^3$</td>
</tr>
<tr>
<td>SUC</td>
<td>$g \ e$</td>
<td>x units</td>
</tr>
<tr>
<td>USC</td>
<td>$g \ f$</td>
<td>y units</td>
</tr>
<tr>
<td>FBP</td>
<td>$g \ g$</td>
<td>z tons</td>
</tr>
</tbody>
</table>

$F_1$ and $F_2$ may now be overlapped in an obvious way. As a matter of fact each entry of $F_1$ will be completely overlapped by an entry of $F_2$. Note that no chopping has to be done before overlapping in this case.

---

$^1$ Cf section 2.2.1.
Example. Consider the object file $F_3 = \text{MONETARY UNITS 1970}:

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>MONETARY UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWEDEN</td>
<td>KRONA</td>
</tr>
<tr>
<td>JAPAN</td>
<td>YEN</td>
</tr>
<tr>
<td>YEMEN</td>
<td>RIAL</td>
</tr>
<tr>
<td>USA</td>
<td>DOLLAR</td>
</tr>
<tr>
<td>FINLAND</td>
<td>MARK</td>
</tr>
</tbody>
</table>

If we want to overlap $F_3$ with $F_1$ of the previous example we get into trouble. The desired result might be:

<table>
<thead>
<tr>
<th>EXPORT COUNTRY</th>
<th>IMPORT COUNTRY</th>
<th>PRODUCT</th>
<th>MONETARY UNIT OF EXP COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWEDEN</td>
<td>BRITAIN</td>
<td>PULP</td>
<td>KRONA</td>
</tr>
<tr>
<td>JAPAN</td>
<td>GERMANY</td>
<td>WATCHES</td>
<td>YEN</td>
</tr>
<tr>
<td>JAPAN</td>
<td>USA</td>
<td>TV-SETS</td>
<td>YEN</td>
</tr>
<tr>
<td>YEMEN</td>
<td>SWEDEN</td>
<td>OIL</td>
<td>RIAL</td>
</tr>
<tr>
<td>SWEDEN</td>
<td>USA</td>
<td>CARS</td>
<td>KRONA</td>
</tr>
<tr>
<td>USA</td>
<td>SWEDEN</td>
<td>CARS</td>
<td>DOLLAR</td>
</tr>
<tr>
<td>FINLAND</td>
<td>BRITAIN</td>
<td>PULP</td>
<td>MARK</td>
</tr>
</tbody>
</table>

$F_1$ is still intact, but $F_3$ has degenerated into a quasi-file\(^1\), as the entry point uniqueness condition no longer holds for that file. The entries $\langle \text{JAPAN, YEN} \rangle$ and $\langle \text{SWEDEN, KRONA} \rangle$ have been duplicated. As has been pointed out earlier there are severe disadvantages connected with quasi-files. Any decision to establish such a file should be carefully considered.

\(^1\) Cf section 8.2.3.
8.3.5 File allocation

After the file establishment, intra-entry, inter-entry, and inter-file structuring operations have been applied and re-applied a number of times, we should have transformed the initial set of e-concepts into a file structure which may easily and efficiently be allocated to a cluster structure which is compatible with the available storage structure. Although the earlier structuring operations will have put many constraints upon the file → cluster mapping, there will usually be a few "degrees of freedom" left for this final file design step. These design decisions will be the topic of this section.

In order to be able to define a file → cluster allocation mapping we must have defined the file and the cluster. We assume here that we have defined the file in earlier structuring steps, so that we know approximately, for example, the number and size of the entries in the file. The cluster, on the other hand, will usually not be equally well-defined when we start the first iteration of the file allocation design step. As was pointed out in section 8.2.1 memory types which are common today possess both permanent and temporary structure. For example, the tracks and cylinders of conventional disk memories form a permanent structure, but each track may often be temporarily formatted into one or more storage blocks, at the designer's free choice.

Thus the formatting of a cluster into a set of cluster blocks is usually an important design decision. If we know approximately the size of the file entries, this design problem could be phrased as

"How many file entries should be mapped to the same cluster block by the file allocation function?"

This problem is also known as the bucket size problem. ¹ It has to be solved under the constraints given by the permanent

¹ Cf Buchholz [173]. The bucket size is defined here as the number of entries per block.
storage structure of the available memories. We may or may not assume that an access block is equal to a storage block.¹

There is no general solution to the bucket size problem. We shall just point to the factors which should be considered when this design decision is made.

The principal advantages of a large bucket size are:

- The available memory space is well utilized, because only a low percentage of the storage positions will be needed for block gaps.

- If a hashing algorithm² will be used for allocating and retrieving the entries of the file, the expected number of memory accesses needed for the allocation or retrieval of an individual entry will be lower at larger bucket sizes, if we assume a constant memory utilization rate (loading density or packing factor).

- Relatively fast retrieval of all entries of the file, because

  (a) the file will occupy relatively few storage positions, block gaps included

  (b) the number of interrupts because of block gaps will be relatively low

  (c) in the case of a rotating memory, the number of "lost rotations"³ will be relatively low, because of the relatively low number of blocks occupied by the file

---

¹ As was mentioned in section 8.2.1 an access block may in principle be both smaller and larger than a storage block. However, it will usually require programming on "the operating system level" to make use of this theoretical possibility, if it is at all possible.

² Cf section 8.3.3.1.

³ It is a common experience that the time it takes for the access mechanism to pass the block gap is shorter than the time it takes for the processor to administrate the interrupt caused by the block gap. Thus one revolution will "be lost".
The principal disadvantages of a large bucket size are:

- The transmitted blocks will occupy a large amount of space in main memory.

- The channel between secondary and primary storage will be busy for long periods at a time, which may cause serious delays in a multiprogramming environment.

- Of the data which are transmitted between secondary and primary storage, only a relatively low percentage will actually be needed by the requesting process. Thus the effective transmission speed will be low.

If we could quantify the factors mentioned above, it would be possible to transform the problem of choosing the best bucket size(s) for a (structure of) file(s) into a numerical optimization problem. For a discussion of this problem and for further references, see Roupé [186].

When we have settled upon both the file and the cluster characteristics, the remaining file design problem is the selection of a particular file + cluster mapping function. This problem will usually have to be solved under several constraints. For example, if we have ordered the entries of the file in one or more ways, it is usually assumed that the mapping function should as far as possible conserve one particular of these orderings. This means that entries which are consecutive according to the file ordering should be mapped to consecutive cluster locations, and finally (by the operating system) to physically consecutive storage locations. If the file should be organized according to the PDO principle, the problem of finding a good allocation mapping coincides with the problem of finding efficient hashing and synonym managing algorithms.

---

1 Cf section 8.3.3.1.

2 Cf section 8.2.4.
8.4 File structuring techniques

In sections 8.2 and 8.3 we have defined a set of basic concepts and structuring operators, which we believe to be useful in data base design work. Thus we have proposed a set of tools. We have also stated advantages and disadvantages with each of the operators seen as separate tools (section 8.3). What remains to be done then, is to develop techniques for using the tools in combination in practical design situations. This is a task which goes beyong this report. However, we shall try to give some indications of what kinds of problems the data base designer might be faced with, and what kinds of techniques might be developed for handling them. The examplifications will be continued in chapter 9.
8.4.1 Demarcation of e8-complexes

Like all systemeers the file designer will often face situations where he feels

- that, on the one hand, he cannot design one file at a time, because most of the files are somehow related to most of the other files, but

- that, on the other hand, the whole system of files is imperceivable, it is a too complex entity to be considered as one undivided design object.

This is a situation which we have dealt with theoretically in section 7.1. According to what was said there, the designer should break down the imperceivable system (of files) into perceivable subsystems. "Well", the designer could reply, "that's what I've done. Thanks to specification of a particular infological model with its detailed and precise e-concept structure, I could at once give a detailed and precise subsystem structure of the data base file system." The designer is right, or almost right. The e-files certainly are basic subsystems of the total data base file system, and it is a good thing that the infological model helps us to establish this basic structure. However, the "span of control" of the file system as a whole seems to be too wide in terms of e-files. An intermediary level needs to be introduced into the hierarchy. The entities on this level should be sets of e-files, called file complexes.

How should file complexes be demarcated within the set of all e-files? Here again we are guided by general systemeering principles: there should be relatively many relationships within each complex, or subsystem, and relatively few relationships between different complexes. The next question is what kind of relationship it is, the number of which should be minimized between complexes. It seems that two e-files are related in a relevant way from the designer's point of view, as soon as it is logically necessary that they are both involved.
in the processing of the same database transaction. Thus it becomes an important design task to find and document these relationships in a way which enables the designer in the next step to demarcate perceivable complexes of e-files for further design activities. One possible technique will be described here.

In order to be able to carry out the analysis described above we must have some idea of the pattern of transactions that will hit the database. We must be able to state formally to what extent different e-files are related to each other in the sense that they are involved in the processing of the same transaction. In this task we may utilize our findings in chapter 5. In section 5.3.2 we established a retrieval query pattern:

\[
(1) \quad "For \ all \ objects \ having \ the \ property \ P^\alpha, \ retrieve \ the \ values \ of \ the \ attributes \ \{A_1^\beta, \ldots, A_m^\beta\} \ at \ the \ times \ \{t_1^\beta, \ldots, t_m^\beta\}, \ respectively." \]

We recall that the property \( P^\alpha \) may be generated from a set of so-called \( \alpha \)-attributes, \( \alpha \)-times, and \( \alpha \)-relations. We also recall that we claimed \( (1) \) to be a fairly general pattern, not only for retrieval queries. Other transaction types will often result in retrieval requests, too.

According to pattern \((1)\) a retrieval query consists of two parts, an \( \alpha \)-part and a \( \beta \)-part, involving \( \alpha \)-e-concepts and \( \beta \)-e-concepts, respectively. Given a query and the specification of an infological model it should not be difficult for the designer to identify these \( e \)-concepts and the corresponding \( e \)-files. It is his task to investigate systematically all imaginable transactions and to document his findings in a form which is useful to the continued analysis. One possible technique is indicated by figure 9.

Figure 9 shows a matrix, spanned by an \( \alpha \)-axis and a \( \beta \)-axis and having one row and one column for each \( e \)-file in the file system. A figure in a particular matrix entry, say \( \langle F_i, F_j \rangle \)
indicates that there will occur transactions which involve $F_i$ as an $\alpha$-file and $F_j$ as a $\beta$-file. The value of the figure is a weight, indicating how frequent and/or important (from response time point of view, for example) the transactions are, in which the particular file combination occurs. Naturally, in practical design situations the figures in the matrix will usually be highly subjective estimates, but formalized "informed judgement" is often the best the designer can hope for.

After rearrangement of rows and columns an $\alpha\beta$-matrix could look like the one in figure 9, where it is not too difficult to find a feasible subsystem structure of the kind we are looking for, i.e., a structure of file complexes with many intra-complex relationships and few inter-complex relationships.\footnote{There are statistical methods for solving this problem, given the matrix.}

In figure 9 there seem to be three complexes of files, comprising $F_1-F_7$, $F_8-F_{10}$, and $F_{11}-F_{15}$, respectively. For each of these file complexes the designer could then tentatively apply the file structuring operators defined in section 8.3. Thanks to the structuring of the original e-files into complexes, the number of alternatives he has to consider has been considerably reduced.

Aside the effect of grouping the e-files into complexes, which are manageable from design point of view, the $\alpha\beta$-analysis also suggests to some extent how the e-files should be established, i.e., if they should be established as object files, property files, or relational files. If we look a little closer at figure 9, we see that for $F_2$, $F_4$, $F_5$, $F_9$, $F_{10}$, $F_{13}$, and $F_{15}$, the columns are crowded with figures, whereas the rows are more empty. For the other files it is rather the other way round. This means that the former files are relatively often concerned by $\alpha$-parts of retrieval queries, whereas the latter are more often involved in $\beta$-parts. This in turn may indicate that the former should be established as property files and the latter as object files. At least that could serve as the designer's first hypothesis. If the object files
Figure 9  αβ-matrix
of a complex are consolidated, the resulting file will be called the *main file* of the complex, whereas the property files, consolidated or not, will be called *directory files*, forming together the *directory* of the *directory/file-complex*. In the next section we shall give an example of what kind of continued design analysis that may be fruitful for directory/file complexes.
8.4.2 An exemplifying analysis of a directory/file complex

After manageable file complexes have been demarcated, the designer has reduced the total file design problem into a number of smaller and hopefully more tractable design problems. However, the designer will usually have a very large number of structuring alternatives to choose among. This will be clearly seen from the example to be analyzed in this section.

As our object of design analysis we have chosen a directory/file complex with the following characteristic.

(1) The main file of the directory/file complex is an object file into which a number of object \( \beta \)-e-files have been consolidated. It is not necessary for our analysis to know the exact number of consolidated e-files. We assume that the file contains information about \( 10^6 \) objects.

(2) The directory of the directory/file complex contains six property e-files representing the e-concepts \( \langle 0, A_1^a \rangle, \ldots, \langle 0, A_6^a \rangle \)

where 0 is the same object group as the main file object group. We assume that each of the attributes \( A_1^a, \ldots, A_6^a \) can take 10 different mutually exclusive values. We also assume that each of the \( \alpha \)-attributes occurs redundantly among the \( \beta \)-attributes, i.e., that the value of an \( \alpha \)-attribute for a particular object may be found both in the directory and in the main file.

(3) In order to be able to evaluate different structurings of the directory, we also need to assume something about the factual distribution of the values of the attributes \( A_1^a, \ldots, A_6^a \) among the objects in 0. We simply assume that the distribution is perfectly uniform for each of the attributes, and that the
attributes are perfectly uncorrelated. Nothing prevents a designer from making more realistic assumptions, but in an illustration the resulting, more complicated calculations would obscure principally interesting conclusions.

Note that so far we have only made "infological assumptions", i.e., assumptions which are parameters in the particular infological model underlying the data base, and which are thus as intelligible to infologically oriented users and decision-makers as to datalogically oriented file designers. This will certainly make it easier for the designer when he needs "intelligent guesses", or still better estimates of parameters which turn out to be crucial for the design decisions.

In addition to the assumptions about certain file parameters, the designer will also need to make assumptions about certain characteristics of the transactions, which will concern the particular file complex to be designed. As far as possible these assumptions, too, should be made in terms of characteristics which are infologically meaningful. For the examplifying analysis in this section, we make the following, infologically oriented assumption.

(4) The bulk of transactions which will hit the file complex will conform to the general retrieval query pattern defined in section 5.3.2:

"For all objects having the property \( P \), retrieve the values of the attributes \( A_1^o, \ldots, A_m^o \) at the times \( t_1^o, \ldots, t_m^o \), respectively."

For this example we assume that the requested values of the \( o \)-attributes are all contained in the object main file of the directory/file complex\(^1\). We further assume that \( F^o \) is the conjunction of from 1 to 6

\(^1\) Cf assumption (1) above.
different complex properties⁴ belonging to the
α-attributes:

\[ p^α = \bigwedge_{i \in \{1, \ldots, 6\}} \langle A_i, v_{ij} \rangle, \quad v_{ij} \in A_i \langle 2 \rangle \]

The different property conjunctions are assumed to occur with equal probabilities.

Even with the rather strict assumptions stated in (1) - (4) there are numerous design alternatives for the particular directory/file complex under consideration. We shall analyze and compare nine different structurings of the directory, which are all feasible, but which show highly varying performance characteristics. For each of the nine structures we have computed the following performance figures which are clearly related to space and time efficiency goals:

(5) \[
\text{Storage overhead} = \frac{\text{Directory size}}{\text{Number of objects}} \approx (\text{Number of directory files}) \times 6 \text{ decimal positions}³
\]

(6) \[
\text{Total expected number of directory and main file accesses per object requested by a retrieval query containing 1, 2, 3, 4, 5, or 6 (selection) attributes}
\]

**Remark regarding (5)**

It follows from assumption (2) above, the definition of "property file" in section 8.2.3, and the definition of the consolidation structuring operator for property files, stated in section 8.3.4.1, that each directory file, elementary or consolidated, will contain one and only one reference to each of the \(10^6\) objects in \(0\). The space needed to represent each of these object references is equivalent to \(10 \log 10^6 \text{ pos} = 6 \text{ pos}\). Thus formula (5) actually gives a measure of the space needed.

⁴ A complex property is a property which is referred to by a complex property reference. Cf section 3.1.4.
² Cf the definition of "attribute" in section 2.3.3.
³ We shall write "pos" instead of "decimal position(s)" during the rest of this section. For a proof of the approximate formula, see the remark below.
to represent the references to each of the objects in \( O \).

Because of the assumptions (1) - (3) above, it is justified to assume that the space needed for entry points, pointers, unused parts of cluster blocks, etc will be small in comparison with this figure. With other assumptions these needs may have to be more carefully estimated.

By calling the quantity (5) "storage overhead" we indicate that the whole directory part of the directory/file complex is redundant because of assumption (2) according to which each \( A^a_i \) occurs redundantly among the \( A^b_j \). For example, if quantity (5) is 12 pos, and each entry of the main file has a size of 100 pos, the relative storage overhead will be \( 12/100 = 12 \% \). This figure represents the space cost paid for the achievement of better retrieval performance.

**Remark regarding (6)**

In the calculations below we count one access per entry segment which has to be retrieved during the processing of a query. We do not allow for the extra accesses which the access algorithm will often need in order to retrieve a particular entry. Nor does our crude measure reflect for example the fact that the accessing of an initial segment of an entry will probably take longer time than the accessing of subsequent segments. In order for our calculations to be at all realistic we must assume that all the files, including the main file, will be allocated to direct-access storage. If, in a realistic situation, we must also consider the alternative of allocating the main file to serial storage, the method of calculation presented here will have to be refined.

Let us now turn our attention to the quantitative and comparative evaluation of nine different directory/file structurings which are all feasible with respect to the database environment as far as the latter was defined in (4).
Structure I: One consolidated object type main file and six elementary property type directory files.

The main file has one-segment entries. There is one directory file for each of the six $a$-attributes$^1$. Each entry of the directory files contains references to the objects for which a particular $a$-attribute takes a particular value, $A_i^a = v_{ij}$. According to assumption (2) and (3) each directory file entry will contain $10^5/10 = 10^4$ object references. With 100 references per segment$^2$, each entry will consist of $10^5/100 = 1000$ segments.

Performance calculations for structure I

In the table below we show the partial results leading in the last two columns to the values of the performance characteristics defined in (5) and (6) above. The calculations are commented upon in the remarks following the table.

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>No of directory accesses</th>
<th>No of main file accesses</th>
<th>Total no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>36</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>2 000</td>
<td>10 000</td>
<td>1.2</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>3 000</td>
<td>1 000</td>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>3 000</td>
<td>1 000</td>
<td>40</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>3 000</td>
<td>1 000</td>
<td>400</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3 000</td>
<td>1 000</td>
<td>4 000</td>
<td>36</td>
</tr>
</tbody>
</table>

Remarks

For 2 and 3 selection attributes 2 and 3 property file entries respectively have to be matched against each other. It is assumed that this matching process may be carried out by the

$^1$ In the existing literature dealing with file organization issues such files are often called "inverted files" or "inverted list files". See for example Lefkowitz [5].

$^2$ We shall use the same segment size, 100 object references/segment $\approx 6000$ pos/segment, for eight of the nine structurings to be investigated in this example. Naturally, this is a design decision, which requires argumentation in a real case.
central processor in primary storage. In order for this to be possible, it is probably necessary with this structure, where there are one thousand 100-object segments (and cluster blocks) per entry, to keep the object references sorted in the directory file entries. Thus the "reordering of terms" structuring operator, defined in section 3.3.2.2, has to be applied.

For 4, 5 and 6 selection attributes 4, 5 and 6 property file entries respectively may analogously be matched, but it is more efficient in terms of accesses to match only 3 property file entries and then access the main file for the remaining 1000 objects.

In the calculations, the results of which are shown in the table, we have assumed that the scheduling subsystem of the database system will be capable of choosing dynamically one processing strategy when a query against the complex contains 1, 2, or 3 selection attributes, and another strategy when the number of selection attributes in the query is 4, 5, or 6.\(^1\)

\(^1\) Cf chapters 7 and 9.
Structure 2: One consolidated object type main file and \( \binom{6}{2} = 15 \) two-dimensional combined property type directory files.

The main file has one-segment entries. Each entry of the directory files contains references to objects for which a particular pair of \( \alpha \)-attributes takes a particular pair of values, \( (A_i^\alpha = v_{ij}) \land (A_k^\alpha = v_{kr}) \). There is one directory file for each of the possible \( \binom{6}{2} = 15 \) combinations of two \( \alpha \)-attributes. With 100 object references per segment, each directory file entry will consist of \( 10^6/10^2/10^2 = 100 \) segments.

### Performance calculations for structure 2:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>No of directory accesses</th>
<th>No of main file accesses</th>
<th>Total no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>100</td>
<td>10 000</td>
<td>1.01</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>200</td>
<td>1 000</td>
<td>1.2</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>200</td>
<td>100</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>200</td>
<td>100</td>
<td>30</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>200</td>
<td>100</td>
<td>300</td>
<td>90</td>
</tr>
</tbody>
</table>

**Remark**

One processing strategy has been assumed for queries containing 1-4 attributes, and another for queries containing 5 or 6 attributes. Cf structure 1.

---

1 A "\( k \)-dimensional combined property file" is a property file resulting from the consolidation, or combination, of \( k \) elementary property files. Cf section 8.3.4.1.
Structure 3: One consolidated object type main file and $6/2 = 3$ two-dimensional combined property type directory files.

The main file has one-segment entries. Each entry of the directory files contains references to objects for which a particular pair of α-attributes takes a particular pair of values, $(A_{i}^α = v_{ij}) \land (A_{k}^α = v_{kr})$. There is one directory file for each of the three attribute pairs $<A_1, A_2>, <A_3, A_4>$, and $<A_5, A_6>$. With 100 object references per segment, each directory file entry will consist of $10^6/10^2/10^2 = 100$ segments.

Performance calculations for structure 3:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>Expected no of directory accesses</th>
<th>Expected no of main file accesses</th>
<th>Total expected no of accesses overhead (pos/object)</th>
<th>Storage (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>1 620$^1$</td>
<td>10 000</td>
<td>1.162</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>1 860$^2$</td>
<td>1 000</td>
<td>2.86</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>920$^3$</td>
<td>820$^4$</td>
<td>17.4</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>200</td>
<td>100</td>
<td>30</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>200</td>
<td>100</td>
<td>300</td>
<td>18</td>
</tr>
</tbody>
</table>

Remark

The weighting procedure indicated in the foot-notes might need some explaining. Thus let us have a closer look at queries involving 2 selection attributes (row 2 in the table). There are $\binom{6}{2} = 15$ different combinations of 2 attributes out of six. Only 3 of these 15 combinations, namely $\{A_1, A_2\}, \{A_3, A_4\}$ and $\{A_5, A_6\}$, have both their member attributes within one and the same combined property file. Queries corresponding to these attribute combinations constitute $\frac{3}{15} = 20\%$ of all queries involving 2 selection attributes, and they require one access to each of the 100 segments of the uniquely determined property.

$^1 0.2 \times 100 + 0.8 \times 2 000$

$^2 0.4 \times 3000 + 0.6 \times 1 100$

$^3 0.2 \times 200 + 0.8 \times 1 100$

$^4 0.2 \times 100 + 0.8 \times 1 000$
file entry. The other 80% of this kind of queries necessitate the matching of ten 100-segment entries from 2 of the 3 property files against one another, which amounts to \(2 \times 10 \times 100 = 2000\) accesses. This explains footnote 1; the other foot-notes are similarly understood. As for the change of processing strategy when the query contains 4-6 attributes, see structures 1 and 2.
Structure $4$: One consolidated object type main file and $\binom{6}{3} = 20$ three-dimensional combined property type directory files.

The main file has one-segment entries. Each entry of the directory files contains references to objects for which a particular triple of $\alpha$-attributes takes a particular triple of values. There is one directory file for each of the possible $\binom{6}{3} = 20$ combinations of three $\alpha$-attributes. With 100 object references per segment, each directory file entry will consist of $10^6/10^3/10^2 = 10$ segments.

Performance calculations for structure $4$:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>No of directory accesses</th>
<th>No of main file accesses</th>
<th>Total no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>100</td>
<td>10 000</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>10</td>
<td>1 000</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>20</td>
<td>100</td>
<td>1.2</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>3</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>20</td>
<td>1</td>
<td>21</td>
<td>120</td>
</tr>
</tbody>
</table>
Structure 5: One consolidated object type main file and $6/3 = 2$ three-dimensional combined property type directory files.

The main file has one-segment entries. Each entry of the directory files contains references to objects for which a particular triple of α-attributes takes a particular triple of values. There is one directory file for each of the two attribute triples $<A_1, A_2, A_3>$, and $<A_4, A_5, A_6>$. With 100 object references per segment, each directory file entry will consist of $10^6/10^3/10^2 = 10$ segments.

**Performance calculations for structure 5:**

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>Expected no of directory accesses</th>
<th>Expected no of main file accesses</th>
<th>Total expected no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>1 240$^1$</td>
<td>10 000</td>
<td>1.12$^4$</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>991$^2$</td>
<td>1 000</td>
<td>1.991</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>124$^3$</td>
<td>460$^4$</td>
<td>5.8$^4$</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>110</td>
<td>10</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>20</td>
<td>1</td>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>

**Remark**

The foot-notes are analogously explained as in connection with structure 3.

---

1 $0.4 \times 100 + 0.6 \times 2000$
2 $0.1 \times 10 + 0.9 \times 100$
3 $0.4 \times 10 + 0.6 \times 200$
4 $0.4 \times 1000 + 0.6 \times 100$
Structure 6: One consolidated object type main file and \( \binom{6}{4} = 15 \) four-dimensional combined property type directory files.

The main file has one-segment entries. Each entry of the directory files contains references to objects for which a particular quadruple of \( a \)-attributes takes a particular quadruple of values. There is one directory file for each of the possible \( \binom{6}{4} = 15 \) combinations of four \( a \)-attributes. With 100 object references per segment, each directory file entry will consist of \( 10^6/10^4/10^2 = 1 \) segment.

Performance calculations for structure 6:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>No of accesses to main file</th>
<th>No of accesses to directory file</th>
<th>Total no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>100</td>
<td>10 000</td>
<td>1.01</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>10</td>
<td>1 000</td>
<td>1.01</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>100</td>
<td>1.01</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>1.2</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>90</td>
</tr>
</tbody>
</table>

Remark

With 100 object references per segment (cluster block) no more than 4 dimensions (attributes) per combined property file will pay with the assumptions of the example. For instance, 5 dimensions would leave 90% directory space unutilized, as only 10 (= \( 10^6/10^5 \)) objects would have each property combination.
Structure 7: One consolidated object type main file and 3 four-dimensional combined property type directory files.

The main file has one-segment entries. Each entry of the directory files contains references to objects for which a particular quadruple of α-attributes takes a particular quadruple of values. There is one directory file for each of the three attribute quadruples \( \langle A_1, A_2, A_3, A_4 \rangle \), \( \langle A_5, A_6, A_1, A_2 \rangle \) and \( \langle A_3, A_4, A_5, A_6 \rangle \). With 100 object references per segment, each directory file entry will consist of \( 10^6 / 10^4 / 10^2 = 1 \) segment.

Performance calculations for structure 7:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>Expected no of directory accesses</th>
<th>Expected no of main file accesses</th>
<th>Total expected no of accesses overhead per requested object</th>
<th>Storage (pos/ object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>100</td>
<td>10 000</td>
<td>1.01</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>86(^1)</td>
<td>1 000</td>
<td>1.086</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>16.2(^2)</td>
<td>100</td>
<td>1.162</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>2.1</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>

Remark

For explanation of the footnote calculations, compare structure 3.

\(^1\) 0.4 \times 200 + 0.6 \times 10

\(^2\) 0.2 \times 1 + 0.8 \times 20
Structure 8: One consolidated object type main file and \( {6 \choose 4} = 15 \) directory files of four-dimensional combined property type, each of which is overlapped\(^1\) with an object file containing the remaining \( 6 - 4 = 2 \) attributes.

The main file has one-segment entries. Each entry of the property type directory files contains references to objects for which a particular quadruple of \( \alpha \)-attributes takes a particular quadruple of values. At the same time each property file entry is overlapped with a number of object file entries, each of which contains the values of the remaining two \( \alpha \)-attributes for one of the objects in the property file entry. The situation is visualized in figure 10. There is one property file for each of the possible \( {6 \choose 4} = 15 \) combinations of four \( \alpha \)-attributes and one object file for each of the possible \( {6 \choose 2} = 15 \) combinations of two \( \alpha \)-attributes. With 100 object references per property file entry segment, each directory file entry will consist of \( 10^6/10^4/10^2 = 1 \) segment.

\(^1\) Cf section 8.3.4.2.
**Figure 10** One entry in a combined property file, containing the attributes $A_1^a$, $A_2^a$, $A_3^a$, and $A_6^a$, overlapped with 100 entries in a consolidated object file containing the attributes $A_3^a$ and $A_4^a$.
Performance calculations for structure 8:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>No of directory accesses</th>
<th>No of main file accesses</th>
<th>Total no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>1 000</td>
<td>100 000</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>100</td>
<td>10 000</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>10</td>
<td>1 000</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>100</td>
<td>1.01</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>1.1</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>120</td>
</tr>
</tbody>
</table>

Remark

For this kind of structure storage overhead per object is calculated as

\[
[\text{(Number of property type directory files)} \times 10 \log (\text{Number of objects}) + \\
+ \text{(Number of object type directory files)} \times \\
\times \text{(Number of attributes per object type directory file)} \times 10 \log (\text{Number of values per attribute})] \text{pos} = \\
= \left( \binom{6}{4} \times 10 \log 10^6 + \binom{6}{2} \times 2 \times 10 \log 10 \right) \text{pos} = \\
= 120 \text{ pos}
\]
Structure 9: One consolidated object type main file and \( \binom{6}{3} = 20 \) directory files of three-dimensional combined property type, each of which is overlapped with an object file containing the remaining \( 6 - 3 = 3 \) \( \alpha \)-attributes.

The main file has one-segment entries. Each entry of the property type directory files contains references to objects for which a particular triple of \( \alpha \)-attributes takes a particular triple of values. At the same time each property file entry is overlapped with a number of object file entries each of which contains the values of the remaining three \( \alpha \)-attributes for one of the objects in the property file entry. There is one property file and one object file for each of the possible \( \binom{6}{3} \) combinations of three \( \alpha \)-attributes. With 1 000 object references per property file entry segment, each directory file entry will consist of \( 10^6/10^3/10^3 = 1 \) segment.\(^1\)

Performance calculations for structure 9:

<table>
<thead>
<tr>
<th>No of attributes in query</th>
<th>No of objects satisfying query</th>
<th>No of directory accesses</th>
<th>No of main file accesses</th>
<th>Total no of accesses per requested object</th>
<th>Storage overhead (pos/object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 000</td>
<td>100</td>
<td>100 000</td>
<td>1.001</td>
<td>180</td>
</tr>
<tr>
<td>2</td>
<td>10 000</td>
<td>10</td>
<td>10 000</td>
<td>1.001</td>
<td>180</td>
</tr>
<tr>
<td>3</td>
<td>1 000</td>
<td>1</td>
<td>1 000</td>
<td>1.001</td>
<td>180</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>100</td>
<td>1.01</td>
<td>180</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>1.1</td>
<td>180</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>180</td>
</tr>
</tbody>
</table>

\(^1\) Note that we have used a different segment size for this structure.

\(^2\) \( \binom{6}{3} \times 10^6 \log 10^6 + \binom{6}{3} \times 3 \times 10^3 \log 10 = 180. \) Cf structure 8, remark.
Comparison of the nine structures

The performance characteristics of the nine investigated structures may be condensed into the following table.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Storage Overhead</th>
<th>Total No of Accesses per Requested Object when No of Attributes in Query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Pos/Obj)</td>
<td>= 1</td>
</tr>
<tr>
<td>Structure 1</td>
<td>36</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 2</td>
<td>90</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 3</td>
<td>18</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 4</td>
<td>120</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 5</td>
<td>12</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 6</td>
<td>90</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 7</td>
<td>18</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 8</td>
<td>120</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 9</td>
<td>180</td>
<td>1.001</td>
</tr>
</tbody>
</table>

The evaluation criteria we use in this example are "storage overhead" and "response time". Even with as few as two decision criteria it is not possible to select one of the nine suggested structures as definitely superior to the others. For this to be possible a formalized or intuitive trade-off rule has to be introduced. However, according to the well-known Pareto principle, we may at once rule out a few structures, which are definitely inferior, or dominated, regardless of the trade-off decision.

For example, structure 1 is dominated by structure 3, structure 5, and structure 7. This is remarkable, because structure 1 is no less than the frequently used inverted list approach.\(^1\) It is also remarkable that structure 1 requires as much as 2, 3, and 2 times the amount of storage overhead required by the respective Pareto superior structures.

Structure 2 is dominated by structure 6 and structure 7. The latter requires, by the way, only one fifth of the overhead storage required by structure 2.

\(^1\) Cf. Lefkowitz [51].
Structure 3 is dominated by structure 5 and structure 7.

Structure 4 is dominated by structure 6 and structure 8.

By assuming very little about the time/storage preferences not yet decided upon, we may also rule out structure 6, structure 8, and structure 9, which are only slightly "faster" than structure 7, but which require considerably more space than the latter. Note, however, that the situation would be completely different if only frequencies were asked for by the queries, so that the main file never needed to be accessed; 1 should then be deducted from every access cell of the comparison table, and structure 9 is then seen to be up to 10 times as "fast" as, for instance, structure 8.

We are thus left with a reduced comparison table containing only structure 5 and structure 7:

<table>
<thead>
<tr>
<th>Structure</th>
<th>Total no of accesses per requested object</th>
<th>Total no of accesses per requested object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(pos/object) = 1</td>
<td>(pos/object) = 2</td>
</tr>
<tr>
<td>Structure 5</td>
<td>12</td>
<td>1.01</td>
</tr>
<tr>
<td>Structure 7</td>
<td>18</td>
<td>1.01</td>
</tr>
</tbody>
</table>

Structure 7 is seen to require 50% more overhead space than structure 5, whereas the latter is significantly "sloven" than the former for more complex queries.
Conclusions

"The infological approach to data bases is an approach which yields an adequate understanding of the data base idea and the basic data base problems. Moreover it is an approach which provides conceptual tools for problem-solving and communication in connection with data bases, tools which may be shared by all parties who are somehow involved in the design and use of a data base." This is how we may summarize the main hypothesis of this report as stated in chapter 1. The objective of this chapter and in particular of the final example has been to show that the infological approach leads us in a very natural way to the traditional hunting-grounds of data logically oriented filorganization professionals. By showing this we also show that we really do not have to sacrifice anything from a datalogical point of view in order to achieve the improved communication possibilities with object-system-oriented data base goal-setters, decision-makers, and users. We can all use the same basic body of infological concepts, and yet the "computer man" can state and solve his problems with the same precision as before. Using the common frame of reference he can also be reasonably confident that he is solving the right problems.

I think that it is fair to say that we have reached our objective in this chapter, i.e. that we actually have reached the traditional hunting-grounds of file organization professionals. For example, the problem which we have illustrated with the very simple example in this section has been treated by several authors.¹

A confusing experience when one studies the literature is that so many authors, with different proposals, claim to have found the optimal file organization scheme. Naturally they are all right in a sense, but it must not be forgotten that any proposal will only be optimal under very specific conditions. Therefore we should once again strongly emphasize

¹ See for example Abraham & Ghosh & Ray-Chandhuri [111], Lum [114], Mullin [119], and Severance & Merten [119].
the importance of the assumptions (1) - (6) underlying the example above. As a matter of fact none of the nine investigated structures is optimal even under these assumptions. The nine structures only constitute a sample of all imaginable solutions to the stated design problem, and there are solutions which are better according to the two stated criteria than any of the proposed structures.

One conclusion from the calculations in this section is that it should not be too difficult to construct an algorithm which (i) generates feasible alternatives to specified directory/file complex design problems and (ii) orders the feasible alternatives according to different design criteria. An effort in this direction has been undertaken by Severance [11]. However, as he has stated his assumptions in terms of datalogical parameters it is not easy to see how his method should be applied in a real design situation, where these parameters are both theoretically and practically unknown. In our example above we have tried to avoid non-infological parameters in the assumptions.

The assumptions made in the example are unrealistic in detail but not in principle. Of course we have chosen "nice figures" like integer powers of 10 for most of the parameters, but that is only a matter of calculation simplicity. Besides, we may not have so good estimates of the entities involved that it is worthwhile using more precise figures than integer powers of some integer like 10 or 2.

In a realistic design situation we should hardly make the assumption stated in (3) that the factual distributions of the values of the attributes $A_1^a, \ldots, A_6^a$ among the objects in $O$ are perfectly uniform and uncorrelated. With more realistic, random distributions it might be impossible to calculate the expected performance characteristics analytically, but then we could resort to simulation. Under such assumptions we shall certainly have to allow for things like highly variable directory entry sizes in the file structures we suggest.
In assumption (4) we stated among other things that the bulk of transactions against the directory/file complex would be retrieval transactions. If instead update would be the most important transaction type this would naturally change our evaluation of different structuring alternatives. For a general discussion of the retrieval/update trade-offs, see Mullin [118].
9. Process design

9.1 Basic concepts and philosophy

9.1.1 Data base processes and resources

In section 7.1.4 we stated that the parts of a system may be classified into two general categories: resources and processes. The two kinds of entities and the relations between them were visualized in figure 2 of chapter 7.

In section 7.3.4 we grouped the processes and resources of a data base system into eight subsystems, visualized in figure 12 of chapter 7. The internal resources of a data base system are data forming either

(a) programs, defining patterns of behaviour for sets of data base processes, or

(b) other data structures, like files, index tables, and so-called control blocks

Beside the data resources, which are internal to the data base system, data base processes interact with computer resources which belong to the computer subsystem of the data base environment and which are thus external to the data base system. Primary and secondary storage, processors, data transmission channels, and input/output devices are typical examples of computer resources. However, recalling section 4.3.4, we should observe that manual resources may also be part of the computer subsystem of the data base environment.

Figure 1 shows a typical pattern of relationships between a data base process and a set of data base internal and data base external resources. A closer analysis would make it possible to classify the relationships into different categories. For example, some resources will either be completely blocked or not at all used by a particular process at a particular point of time. Other resources may be shared by several

1 Cf section 4.3, in particular figure 2.

2 Cf Bubenko [83].
Figure 1 A data base process and the resources to which it is related, directly and indirectly, at a particular point of time
processes, each of which uses the resource to a certain degree, less than 100 %. There are also resources which may in principle be used by any number of data base processes at a time. So-called re-entrant programs belong to the latter category of resources.

When a data base process has used a certain resource it may return the resource in its original state or modified in some way. Data base processes may also create and destroy resources.

In section 7.1.4 we defined a resource in general as more or less flexible depending upon (i) the range of other resources into which it may be transformed, and (ii) the "easiness" with which the transformation takes place. Reversely, we shall say that a resource is more or less elastic\(^1\) depending upon how easy it is to replace the resource with some other resource. For example, main memory may be an elastic resource provided that the operating system of the computer contains efficient algorithms for paging and swapping.\(^2\)

A data base process interacts not only with internal and external resources but also with other processes which are internal or external to the data base system. As was visualized by figure 2 in chapter 7, the interaction between processes consists in the exchange of stimuli: initiation, suspension, activation, and termination stimuli. A data base process is initiated by a stimulus from the environment, i.e. from an external process, or by a stimulus produced by another data base process. After its initiation, the data base process exists as an alternately active and passive (suspended) process until it is terminated by itself or by another internal or external process. After that the process does not exist any longer; it can never be activated again.

In earlier chapters we have discussed the interaction between the environment and the data base in terms of a black box S-O-R model. We said that an input transaction stimulus, S,

\(^1\) Cf Colin [134], chapter 14, page 99.
\(^2\) Cf Bubenko & Ohlin [133], and Watson [140].
somehow caused or resulted in an output transaction response, R, and a transformation of the data base "organism", 0 → 0'. We need the process concept to explain how the result comes about. The input transaction, seen as a stimulus\(^1\), initiates or activates a data base process, which in turn initiates or activates other processes etc. The processes use, release, create, and destroy resources. Among the created resources may be an output message, and among the destroyed resources may be a piece of paper upon which the message is written. The total transformation of the organism consists of all changes of states, creations, and destructions of internal resources, which have taken place during the interaction. Naturally, for different purposes one may use different degrees of resolution in order to explain the behaviour of a data base system. Both the processes and the resources may be described with different degrees of detail. On the one hand a certain interaction may be analyzed in terms of a process with just a few sub-processes, or steps, using a few main resources. On the other hand the execution of each instruction of the programs governing the processing of the interaction may be regarded as a separate sub-process, each bit of storage may be regarded as a separate resource etc.

\(^1\) The input transaction may often be seen both as a stimulus and as an external resource.
9.1.2 Design of programs, processes, and routines in "conventional systems" and in data base systems

In the previous section we discussed among other things the "program" and "process" concepts. We shall start this section by introducing a third related concept, "routine", which we shall need to make a clear distinction between what we shall call "conventional systems" on the one hand and "data base systems" on the other. This distinction in turn is fundamental to the understanding of the partially new meaning of the process design task in connection with data base systems.

Let us first review the "process" and "program" concepts. A process is an abstract entity, a theoretical construct, which we often find it practical to introduce as the bearer or the cause of an observed or a desired change, produced or to be produced by a system. In so doing the process uses, produces, and transforms system internal and system external resources. A process is born, initiated, at a particular point of time, and it dies, terminates, at another point of time.

During its life time, the behaviour of a process is governed by a program, which is one of the resources used by the process. A program is a data structure representing a set of statements or instructions which are executable by a processor, e.g. the central processor of a computer. A program is created by a process, and it may be destroyed by another process.

During its existence, the same program may very well govern the behaviour of several processes. Programs governing processes which do not modify the governing program may even govern several processes more or less simultaneously. A routine is an abstract entity which we define here as the set of all processes which have existed, are existing, or will exist, and which are governed by one and the same program. A routine is said to be run as often as one of its member processes is being executed. If an individual process, p, belonging to the routine R, is initiated at t and terminated at \( t + \Delta t \), we shall say that R is run at t, or more precisely, during the time interval \( \langle t, t + \Delta t \rangle \).
Note that

(a) a routine is an abstract entity without extension in the time dimension

(b) a process is an abstract entity with extension in the time dimension

(c) a program is a concrete entity with extension in the time dimension

Routines, processes, and programs may be hierarchically built up by entities which are themselves routines, processes, and programs, respectively.

Remark. The terms "routine", "process", and "program" are used in many different ways in the literature, in manuals, and in data processing practice. Any attempt to introduce theoretically well-founded definitions will therefore get into conflict with some usage of the terms. For example, whereas "routine" usually refers to a concept which is at least approximately identical with the routine concept defined above, "sub-routine" is very often used to refer to an entity which would be called "sub-program" in our terminology. The distinction between "process" and "routine" is not always made explicitly. Very often the term "process" is used to refer both to individual processes and to classes of processes "of the same kind". Admittedly, there are many practical situations, especially in connection with what we shall call "conventional systems" (of below), where the distinction between processes and routines is so obvious that there is very little risk of ambiguity even if the same term is used for both concepts.

Let us now turn to the announced analysis, of the differences between "conventional systems" and "data base systems" and the implications of these differences for the process design task.

We shall use the term "conventional system" for a computerized information processing system with the following characteristics.
The system consists of a number of routines which become well-defined at design time. Each routine is typically run at regular intervals, for example once a month, once a week, or once a day. The length, $\Delta t$, of the time intervals $\langle t_1, t_1 + \Delta t \rangle$, $\langle t_2, t_2 + \Delta t \rangle$, etc., during which a routine is run, is short in comparison with the time distance, $t_2 - t_1$, between two successive runs of the same routine. Every time a particular routine is run, different time versions of the same standing files and temporary files (transaction files) are processed, and new time versions of the same standing files and output transaction files (report files) are produced.

Naturally we can build a data base system around the standing files of a "conventional system" as defined above. However, if one devotes a lot of scarce resources to the design and construction of a data base system, one will probably not be satisfied with a system which is only capable of interacting with its environment in such a rigid way as a "conventional system". In particular one should probably not be willing to determine at design time a well-defined set of routines to be run at regular intervals. Instead one should like different categories of interactors to be able to influence at operation time what kind of processing the data base system should be occupied with during a particular time interval. An information consumer, for example, should be able to initiate a particular kind of data base processing by means of a retrieval query transaction formulated in a result-oriented (non-procedural) interaction language. Thus the temporary files of a data base system will often contain single transactions rather than batches of transactions of the same kind. Many transactions will appear at the convenience of different uncoordinated data base interactors in a way which is more or less out of the designer's control. The best the designer can do in such situations is to estimate the distributions of different transaction types as discussed in chapter 7.

The above-mentioned requirements upon data base systems have important implications both for the routine structure of a data base system, and for the very process design task. The routine structure of a data base system is typically more
complex, more dynamical, and, above all, less explicit at design time than the routine structure of a "conventional system". The data base system designer may only be able to establish certain basic routines explicitly at design time. The set of basic routines would include for example

(a) "routine skeletons", corresponding to the main transaction types that the data base system should be able to process

(b) "dynamic design routines", which are responsible for the dynamic transformation of the "routine skeletons" into complete routines by the time a particular data base transaction is actually being processed.

Thus the data base designer will not be able to design and schedule all data base routines (and indirectly all data base processes) at data base design time. Nor will it be possible "for the operation staff to make the missing planning and scheduling decisions at run time. Instead the data base designer has to design the data base system so as to make the system itself capable of planning and coordinating its own activities to a much greater extent than has been usual in "conventional systems".

Example. With the "conventional" philosophy the system designers do not only look for and document logical relations between e-concepts. They also determine, permanently, what e-concepts are to be contained in what files, and what files are to be used by what information processes. In a data base system the mapping of e-concepts into files may have been changed between two executions of similar user requests, as the result of automatically or manually initiated reorganization processes. This in combination with the vast number of user transaction types that a data base system should be ready to process makes it impossible to state an explicit file processing strategy in beforehand for each imaginable situation. Planning processes, which are integrated parts of the data base system, have to make these decisions dynamically in accordance with certain
basic rules, which have been formally stated and programmed at data base design time. What kinds of strategy choices concerning the processing of the standing files of the data base system that have to be made dynamically will be discussed to some detail in subsequent sections.

Example. In a "conventional system" development situation the programmer may have the possibility and the duty to consider different sub-process structures for a certain information process. It may be his task to choose access methods, to determine ordering and possible overlaps of sub-processes, etc. Similarly the computer operations staff may have responsibility for the coordination of different batch executions so as to achieve reasonable through-put and turn around times. The necessary conditions for such detailed pre-planning are not present when data base requests are to be processed. Even if response time requirements would admit a professional programmer to be engaged, it would hardly be possible for him to keep in his mind the ever changing situation of the interior and the exterior of the data base system and to make feasible, not to speak of optimal, process structuring decisions. Thus a data base system has to include processes for this type of process planning and coordination activities as well.

Example. Langefors and others have developed a theory of process grouping. In the development of "conventional systems" it is quite feasible to make process grouping decisions "manually", even though it has naturally been tempting and challenging to automatize this and other system design tasks. In a data base system many process grouping problems have to be solved dynamically and thus automatically.

The examples have shown that many different process design decisions, which have traditionally been the responsibility of systemeers, programmers, and operating staff, have to be made automatically, at processing time, in a data base system. Note that this does not imply for example that the importance

1 See Langefors [48], Arvas [133], Sundgren [138], and Nunamaker [136], [137].
of programmers will decline in data base oriented information systems. It is obvious that the software for the dynamic planning processes themselves will require extremely advanced programming, as will all modules of a data base management system. Further, the data base administration function will continuously need skilled programming assistance. Finally, result-oriented languages will not eliminate the need for clever, problem-oriented algorithms; the problem programmer will not have to worry about data management, however; he will have more powerful built-in functions at his disposal and might find it feasible to attack more complex problems than before.
9.1.3 Resource-oriented and transaction-oriented process design

There are two basic categories of process design activities to be performed dynamically and automatically when a data base system is run upon a computer. We shall call the two categories

(a) resource-oriented process design and

(b) transaction-oriented process design

With some simplification we could state the general objectives of the two kinds of activities in the following way. The resource-oriented process design activities should see to it that the resources of the system are as efficiently utilized as possible under certain restrictions given by the transaction-oriented process design activities. Conversely the transaction-oriented process design activities should see to it that each interaction\(^1\) between the data base system and its environment is as efficiently handled as possible under certain restrictions given by the resource-oriented process design activities. Still less precisely, we could state the two general objectives as (i) maximize through-put under certain response time restrictions, and (ii) minimize response time under certain through-put restrictions.

Even the vague goal formulations above suggest that there are many important trade-offs which the designer of data base process design processes has to consider. There is the basic trade-off between a high utilization rate and a high service rate, which is well-known from other kinds of systems. On a more detailed level trade-offs have to be made between the utilization rates of different resources as well as between the individual service rates with which different transactions from different interactors are processed. In this connection it is important that the designer or, still better, the system itself is able to identify the most scarce resources of the

---

\(^1\) Cf section 4.2 and chapter 5.
system as well as the variability in service demands which
often exists. For example, all kinds of transactions from the
data base environment do not require immediate response. It
may be essential for good overall functioning of the data base
system that the planning processes of the system are capable
of identifying and postponing the processing of transactions
which do not require response within the next couple of seconds,
the next couple of hours, the next couple of days, etc.
Consider, for instance, a long-range planner. He may need
a piece of information from the data base which requires
extensive processing of many large files. It is quite possible
that he will not be significantly delayed in his planning work,
if he has to wait a day or a week for the information he wants,
provided only that he is certain that he will get response
within a day or a week, respectively, and that he will then
get exactly the information that he expects that he will get.
Such a situation has the following implications for the data
base system. Immediately when the planner has submitted his
request to the data base, there has to be a very intensive
series of interactions between the data base and the interactor
with questions and answers within seconds or possibly minutes.
During this communication the information consumer should
learn if the data base is at all capable of replying to the
query, how long it will take, how much it will cost, etc.
He should also get some sample output showing how the ultimate
answer will be presented\(^1\), and he should be enabled to modify
the layout. The data base system should also see to it that
the information consumer has a compatible frame of reference\(^2\),
so that he is aware of the meaning and the quality of the
information output that he will ultimately get. Sometimes the
data base may itself take the initiative to modifications of
the initial query\(^3\).

\(^1\) If the data base is equipped with a "mini-base" (cf 9.1.4 below)
the information consumer could even get a rough idea of what
the information contents of the ultimate answer will be. On the
basis of the sample information he may e.g. decide that continued
processing is not worth-while.

\(^2\) Cf section 1.5 and section 4.4.1.

\(^3\) An example of this kind of "Interactive Modification of User
Requests" (IMUR) will be found in the end of section 9.2.5.
The example with the long-range planner above should have illustrated some aspects of transaction-oriented process design in data base systems. Other aspects will be treated in subsequent sections of this chapter. Resource-oriented process design in data base systems has very much in common with resource-oriented process design in "conventional systems". Typically the dynamical resource-oriented process design activities which take place in "conventional systems" are performed by the operating system of the computer, or by user-manufactured extensions to the operating system. This is natural, because the resources, the requests for which are administered, are often demanded by several applications at the same time. As to the computer hardware resources this may very well be true for data base systems, too. As has been pointed out in earlier chapters, the computer is formally a part of the data base environment, and may be shared by the data base system and other applications, which may themselves be other data base systems.

In a "conventional system" the set of computer resources is more or less identical with the set of all resources that require dynamic, automatic administration. As was discussed in section 9.1.2 the situation is quite different in a data base system. For example, the standing files of a data base are examples of system-internal resources which are very central to the functioning of the data base system itself. They are also examples of resources, which are often demanded during the same time interval by several different transactions. All together, there are in data base systems internal resources which are natural objects of dynamic, automatic resource-oriented planning and coordination activities, performed by the data base system itself and not by the operating system.

1 Example: the scheduling of the access requests to a particular disk storage device. These kinds of process design problems have for a long time attracted the attention of manufacturers and users of operating systems allowing multiprogramming. See for example references [92], [90].

2 See chapter 4, figure 2, and section 4.3.1.
Remark. The argument above points again to the need for a
generalized theory of process grouping, a theory of dynamically
and automatically performed process groupings and other process
scheduling operations. Cf section 9.1.2. It is a difficult
problem area, even when restricted to the statical, non-
automatical or semi-automatical special cases. See references
[48], [131], [136], [137].

Having pointed to the resource-oriented data base process
design problems to be handled at operation time by (i) the
operating system of the computer, and (ii) the data base system
itself, we now turn our attention to the transaction-oriented
process design problems to be solved dynamically by the data
base system alone.
9.1.4 Transaction-oriented process design functions to be performed dynamically by a data base system

It was explained in section 9.1.2 why we cannot expect the process design for all imaginable data base transactions to be completed at data base design time. In order to get an overview of the transaction-oriented process design functions which have to be performed dynamically, at data base operation time, we shall follow the trace of a particular data base transaction on its way through the data base system. We have chosen to follow the processing of a retrieval query, but this does not significantly lessen the generality of the analysis. The basic processing principles and process design problems of other transaction types\(^1\) are in effect very similar to those of retrieval queries.

What happens first when the retrieval query hits the data base system is that it is recognized as a retrieval query. The interaction module\(^2\) of the data base system will also select the appropriate compiler/interpreter, the compiler/interpreter of the interaction language in which the retrieval query has been formulated by the interactor. The external representation of the query will then be translated into some kind of normalized representation. Following Nordbotten [55], we shall call this data base internal language the internal dialect of the data base system. Thus the external references to attributes, values, object groups, object relations, generation rules, etc, will be translated into the vocabulary of the internal dialect, which will probably be much more laconic than the interaction language. Moreover, the grammatical structure of the transaction statements has to be analyzed and transformed into the grammatical structure of the internal dialect. This parsing process will be facilitated if the underlying structure of the interaction language is similar to that of some well-defined formal language like first-order predicate calculus.

---

1. Different kinds of data base transactions were systematically identified in chapter 5.
2. We shall refer in this section to the data base modules, or subsystems, suggested in chapter 7. Recall figure 12 of that chapter.
The normalized statements corresponding to the retrieval query have to be analyzed further in order to eliminate references to entities, which are not explicitly stored in the data base, but which are formally defined directly or indirectly in terms of stored entities. During this analysis the data base system has to consult the data base schema dynamically, at operation time, in much the same way as the systemeer of a "conventional system" has to consult the precedence matrices\(^1\) at design time. We shall therefore call this dynamic data base process design task "dynamic precedence analysis".\(^2\)

After the dynamic precedence analysis the data base system knows what stored e-concepts the retrieval query explicitly or implicitly refers to. The data base system also knows how these e-concepts should be infologically processed in order to produce the desired result. However, each initial e-concept may be stored in several files, and there may be several datalogical strategies corresponding to the same infological processing. The scheduling module of the data base system will have to consult the data base catalogues as well as the accumulated statistics in the accounting module in order to be able to make an executively optimal decision about which of the files to access and which of the strategies to choose.

In order to support the modal\(^3\) optimality of the dynamic choice of a file processing strategy, the data base system should collect and maintain as much information as possible about its own infological and datalogical characteristics. Obviously it would not be executively\(^3\) optimal to design the data base system to do so. Even if "perfect" information about the data base were to be available at the time when a file processing strategy has to be chosen for our retrieval query, it might not be executively optimal for the scheduling process to make full use of it, because of the implied resource consumption by the scheduling process itself.

---

1 The use of different kinds of precedence analyses and precedence matrices in the design of "conventional systems" is treated in detail in Langefors [48].

2 The dynamic precedence analysis process includes such things as the break-down of a complex αβ-query into αβ-queries. Cf section 6.3.1.

3 For a discussion of the distinction between modal and executive optimality see Langefors [26].
An interesting alternative to the maintenance and utilization of elaborate data base statistics is the maintenance and utilization of "mini-bases". A mini-base is a miniature data base, containing for example all available information about a random \(^1\) sample of the objects about which information is contained in the data base. In strategy choice processes the data base system often needs to know approximately how many objects have a particular combination of \(n\) properties \((n \geq 1)\).\(^2\) If the number of fundamental properties is large it will be feasible to store exact statistics about property combinations for low \(n\)'s only, often only for \(n = 1\). With a mini-base instead no exact figures would be available, but on the other hand estimates could be calculated for arbitrary property combinations and arbitrary \(n\)'s, and this may be more valuable, particularly in cases where not much is known at design time about the frequencies and response time requirements of different query types.

Mini-bases have other advantages as well. Sometimes the information available from a mini-base could satisfy not only the needs of the process design processes but also the needs of the data base interacting information consumer. Thus under favourable circumstances an information consumer could get useful response to his retrieval request much faster and at a much lower cost from a data base with a mini-base than from a data base which does not have this feature. Mini-bases could also be very helpful to the maintenance and reorganization functions of the data base system.

Let us assume now that the data base system has performed dynamic precedence analysis and chosen a file processing strategy\(^3\) for the retrieval query which is under our consideration.

\(^1\) In order to be random the sample should be selected by a criterion which is not related to any information contained in the data base.

\(^2\) This information need may be almost as complex as the information need expressed by the original retrieval query, the efficient processing of which is the concern of the strategy choice process. Then it is important to note that very crude estimates may be quite satisfactory to the latter process, whereas the data base external information consumer is likely to need more precise information.

\(^3\) Examples of retrieval situations and file processing strategies will be given in section 9.2.
in this section. This implies that the scheduling module of
the database system has now developed a plan, which tells
what access requests to issue to what files, and in what order
to initiate the respective file accessing processes. The
schedule may be quite complex. Some file access requests may
be independent of each other and may be processed in an
arbitrary order or in parallel. Other file processes may be
serially dependent, so that one has to be completed before
the other may be initiated. Still other processes on the
schedule may be "parallelly dependent", i.e., they have a common
precedent without being serially dependent. In figure
2, for example processes A and C are independent of each other,
D and K are serially dependent (with a precedence relationship
of order 2), and E and J are parallelly dependent with the
common precedent D.

The scheduling module must not only develop the file processing
plan. It should also supervise the realization of the plan
and be prepared to revise it dynamically.

According to our definition of the file concept in section
6.2.2 it should be the task of the files subsystem of the
data base system to actually access the files. The access
algorithms should take normalized entry requests as input and
should produce the requested entries edited in accordance
with the data base normal format. As was discussed in chapter
8 the file access algorithms may in turn employ sub-algorithms
like segment collecting algorithms, hashing algorithms for
the retrieval of synonyms, and cluster access algorithms
(which may belong to the operating system of the computer
rather than to the data base system). When a file access process
is executed it will initiate sub-processes corresponding to
the sub-algorithms.

The output from the individual file processes could be said
to constitute sub-replies corresponding to sub-queries of the
retrieval query originally received by the data base system.
Infologically the synthetizing of the sub-replies into a
"grand reply" is the reverse process of the dynamic precedence
analysis discussed earlier. Matching processes and processes corresponding to "standard operators"\(^1\) are examples of probable sub-processes of the synthesis.

Finally, before the "grand reply is presented to the data base interactor, it has to be translated from the internal dialect to the appropriate external format in accordance with the interactor's specifications.

This completes our analysis of the processing of a retrieval query transaction. It should be emphasized that we have only followed "the main stream" of the processing, For example, we have not treated the important sub-processes which are handled by the quality and protection filter functions of the data base systems. However, the processing of a request from one data base subsystem to another will often in principle be very similar to the processing of an external transaction from the data base environment to the data base. Many of the data base subsystems will themselves have subsystem structures similar to the subsystem structure of the whole data base, i.e., they will have their own internal file systems, containing meta-information\(^2\), their own interaction and scheduling modules their own filter functions, etc. Thus if we reinterpret the analysis in this section and apply it iteratively and recursively, it should be relevant to most transaction-oriented process design activities undertaken by the data base system or any of its subsystems.

---

\(^1\) "Standard operators" are operators, or generation rules, which are once and for all defined to the data base system. Common examples: mathematical operators and statistical operators computing means, deviations, and correlations, performing regression analyses, etc.

\(^2\) Cf section 3.2.5.
9.2 The processing of an αβ-query

Like before\(^1\) an αβ-query is defined as a retrieval query conforming to the general pattern

\[(q)\] "For all objects having the property \(p^\alpha\), retrieve the values of the attributes \(A^\beta_1\), ..., \(A^\beta_m\) at the times \(t^\beta_1\), ..., \(t^\beta_m\), respectively."

In the previous section we gave a general overview of the transaction-oriented process design tasks that a database system has to solve dynamically. Among other things we mentioned that the database system must be able to choose dynamically a feasible and efficient file processing strategy for any particular transaction which is being processed. In this section we shall try to concretize this task, and the problems connected with it, by means of a couple of examples. We adopt the following general assumptions as to the structure of the query to be processed and the file structure of the database at processing time:

(a) The query is an αβ-query \((q)\) where

\[p^\alpha = p^\alpha_o \land (A^\alpha_1 = v^\alpha) \land ... \land (A^\alpha_n = v^\alpha)\]

and \(p^\alpha_o\) is a type property\(^2\) like "person", "car", "enterprise", "customer", or "pupil". All times involved in the α- and β-parts of the query are identical (= t).

(b) The files under consideration in the subsequent analysis are object files and property files corresponding to time versions t of attribute i-concepts\(^3\). The attributes involved in the files are classified as α-attributes, β-attributes, or x-attributes, depending upon whether they belong to the α-part, the β-part, or no part at all of

---

\(^1\) Cf sections 5.3.2, 6.3.1, and 8.4.1.

\(^2\) Cf section 2.3.2.

\(^3\) Cf section 3.2.3.
the \(\alpha\beta\)-query which is being processed.\(^1\)

(c) For the files under consideration in the subsequent analysis we also assume that the objects referred to in the entry points of the object file entries and in the "exit points" of the property file entries have the property \(p^\alpha\) in common.

With the assumptions adopted as to the type of transaction and the general structure and contents of the files, there are still a lot of different process design situations which may occur, depending upon the particular combination of query and detailed file structure and file characteristics. We shall partition these situations into a few typical classes, and for each class we shall devote a section below to an examplifying discussion of the design considerations which have to be made dynamically by the planning and scheduling module of the data base system.

In order to show the typical features of each retrieval situation to be analyzed, we shall use a tool called the relevance matrix. It is defined for each combination of \(\alpha\beta\)-query and file system in the following way (cf figure 3):

There is one row in the matrix for each \(\alpha\)-attribute and one for each \(\beta\)-attribute in the \(\alpha\beta\)-query. Then there is one column for each of the files in the file system which contains at least one of the \(\alpha\)-concepts corresponding to the \(\alpha\)- and \(\beta\)-attributes. Finally there is one row in the matrix for each \(x\)-attribute, i.e. each attribute which is contained in at least one of the column files, but which is neither an \(\alpha\)-attribute, nor a \(\beta\)-attribute.

The cells of the relevance matrix may contain different symbols. Thus if cell \((j, k)\)

\[= a,\]

then file \((k)\) contains attribute \((j)\)

\(^1\) Note that the "\(\alpha, \beta, x\)"-classification of attributes in only defined with respect to a particular query. The same attribute of the same file may be an \(\alpha\)-attribute with respect to one query and a \(\beta\)- or an \(x\)-attribute with respect to another.
<table>
<thead>
<tr>
<th>object files</th>
<th>property files</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>S</td>
<td>D</td>
</tr>
</tbody>
</table>

$A_1^a$  
$A_2^a$  
$A_3^a$  
$A_4^a$  
$A_5^a$  
$A_6^a$  

$A_1^a$  
$A_2^a$  

$A_1^x$  
$A_2^x$  
$A_3^x$  
$A_4^x$  
$A_5^x$  
$A_6^x$  

<table>
<thead>
<tr>
<th>$i^1$</th>
<th>$i^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

$A_1^i$  
$A_2^i$  
$A_3^i$  
$A_4^i$  
$A_5^i$  
$A_6^i$  

$A_1^i$  
$A_2^i$  

$A_1^c$  
$A_2^c$  

Figure 3  Relevance matrix for an αβ-query

S = serial access only; D = direct access

a = accessible attribute

i = accessible and object identifying attribute

c = component attribute of compound key

The meaning of the superscripts is explained in the text.
= i, then attribute \((j)\) is an object identifying attribute contained in file \((k)\); if the file is an object file, the attribute is the (primary) key of the file

= c, then attribute \((j)\) is a component attribute of the compound key of file \((k)\).

Some of the symbols in the cells of the relevance matrix have integer superscripts. For example, the \(P^O\) column has superscript "1" for the "i" symbol in the \(A^X\) row. This implies that the entries of the object file \(P^O\) are ordered (sorted) by the values of the object identifying attribute \(A^X\). The \(P^P\) column has superscript "1" for the "c" in the \(A^a\) row, "2" for the "c" in the \(A^a\) row, "3" for the "c" in the \(A^a\) row, and finally "4" for the "i" in the \(A^X\) row. This implies (i) that the entries of the property file \(P^P\) are ordered by the values of the key component attributes \(A^a\), \(A^b\), and \(A^c\) (in that very order), and (ii) that finally the object references ("the exits") are sorted (by the identifying attribute \(A^X\)) within each entry.

We may add extra rows and columns to the relevance matrix in order to indicate further characteristics of the files or the attributes, respectively. For example, in figure 3 there is one extra row where an "S" indicates that the entries of the file are only serially accessible and where a "D" indicates that the entries of the file are also directly accessible via the key attribute.

---

1 Consolidation (combination) of property files was defined in section 8.3.4.1. See also section 8.4.2. If \(n\) elementary property files, representing \(n\) e-concepts of attribute type, \(\langle 0, A^1\rangle, ..., \langle 0, A^n\rangle\), are combined into one compound property file with the (primary) key attribute \(A^X\) formed by Cartesian multiplication, then \(A^1 \times ... \times A^n\) is said to be a compound key with the component attributes \(A^1, ..., A^n\).

2 According to our definition of a file (see section 8.2.2) each entry of a file is always serially accessible. If the file is coupled with an access algorithm, which makes each entry accessible through a number of (auxiliary) accesses, which is small, both absolutely and in comparison with the number of accesses necessary for serial access, the file entries are said to be directly accessible as well. Then the file is called a direct access file, whereas other files are serial.
The relevance matrix could be used not only as an illustration tool. When the database system is about to process a particular query, it will have to assemble data corresponding to the contents of the relevance matrix for the particular \( \text{query, file system} \) combination. It will have to do this in order to be able to choose dynamically a feasible and efficient file processing strategy.

**Remark.** The following exemplifying analyses should not be mixed up with the analyses in section 8.4.2, even though they represent in a sense two sides of the same coin. The decisions considered in chapter 8 were design time decisions, i.e., decisions to be made by the system developer when the database is being designed. The designer may make the decisions "manually" or assisted by a computer. The decisions considered in this chapter are operation time decisions which have to be made automatically by the database system. In the "chapter 8 situation" the designer is free to choose any structuring of the files which he thinks will fit the (more or less unknown) transaction pattern that will hit the database system at operation time. In the "chapter 9 situation" the database system has to accept the existing file structure and make the best of it when it processes a particular (well-defined) database transaction. Naturally a series of bad experiences from such situations could and should lead the database system to recommend the designer to restructure the files, if it does not possess the capability and authority of reorganizing its files automatically, on its own initiative.
9.2.1 A basic retrieval situation

We start our process design analysis of query (q1) from a "basic" retrieval situation (figure 4) with the following characteristics:

(a) The compound attribute \( A_1^\alpha \times \cdots \times A_n^\alpha \) formed by all the \( \alpha \)-attributes of query (q1) is the key of a compound property file.

(b) All the \( \beta \)-attributes of query (q1) are contained in one object file.

(c) The "exit points" (object identifications) of the combined property file are entry points of the object file.

(d) The entry points of both files are directly accessible.

The obvious way of processing query (q1) under conditions (a) - (d) is first to access the appropriate entry of the compound property file, which will render the identifications of all objects having the \( \alpha \)-property, and then to access in turn the object file entries corresponding to these objects. The latter accesses will render the requested values of the \( \beta \)-attributes.

If there are \( m \) objects having the \( \alpha \)-property the processing of query (q1) according to this strategy will probably require at least \( (m+1) \) secondary storage accesses, one for the compound property file entry and one for each of the \( m \) object file entries. The number could be less only if at any stage of the processing a demanded access block is already in primary storage; one may increase the probability for this to occur by sorting the object identifications resulting from the property file access(es) according to the object file sequence, if there is such a sequence.
For several reasons the actual number of secondary storage accesses will probably be greater than \((m+1)\). For instance,

- auxiliary accesses may be needed to determine, during query analysis, what files are to be accessed and then to "open" the files if they have not been opened already

- if the entry point values of a file are more or less randomly scattered over a range of possible values, the accessing of an entry of the file will probably, on the average, require more than one access

- the entries may have been chopped, and all requested data may not be in the entry point segment; continuation segments may be in other access blocks than that of the initial segment.

Sometimes it may be advantageous to deviate from the "obvious" strategy of processing query \((q1)\), even when all the "basic" conditions \((a)\) - \((d)\) are fulfilled. If the appropriate entry of the property file has been chopped and mapped into several access blocks, it may be worthwhile or even necessary to group the property file access process with the object file access process. One would then access one segment of the property file entry and then, possibly in parallel, access the corresponding object file entries and access the next property file entry segment.

A more dramatic change of strategy would be to replace the direct accessing of the directory/file-pair\(^1\) with a "traditional" serial search of the object file. This is logically possible provided that the object file contains not only the \(\beta\)-attributes, but also the \(\alpha\)-attributes. If a sufficiently large fraction of the object file entries are to be accessed, serial search may be substantially faster than direct access; on the other hand, transmitted data volumes will probably increase, of course.

\(^1\) Note that the property file actually has the function of a directory to the object file.
Example: Let us compare the strategies

$S_1$: directory/file search, and
$S_2$: serial scan of the object file

in the situation illustrated by figure 4. According to $S_1$ we start by retrieving from the property file $F_1^P$ the entry with the key

$\langle A_1^a, A_2^a \rangle = \langle v_1^a, v_2^a \rangle$

There are 10 000 objects referred to in the files. Suppose that a proportion $p$ ($0 \leq p \leq 1$) of these have the property $(A_1^a = v_1^a) \wedge (A_2^a = v_2^a)$. Then there are 10 000p object references in the retrieved entry, which, accordingly will occupy

$q = \left\lceil \frac{10 000p}{[2 000 - 2]/13}] + 1 \approx [65p] + 1 \right\rceil$

blocks. The expected access time will be

$1.05 q t_d^p$

and the expected transport size

$2 000q$

positions. The next step would be the accessing of the 10 000p entries of the object file. The expected access time for this will be

$1.2 \times 10 000p t_d^o = 12 000p t_d^o$

and the expected transport size

$2 000 \times 10 000p$

positions.
<table>
<thead>
<tr>
<th>attribute type</th>
<th>( f_1^o )</th>
<th>( f_1^p )</th>
<th>attribute size (pos)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1^a )</td>
<td>a</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>( A_2^a )</td>
<td>a</td>
<td>c^2</td>
<td>1</td>
</tr>
<tr>
<td>( A_1^β )</td>
<td>a</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>( A_2^β )</td>
<td>a</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>( A_1^x )</td>
<td>i</td>
<td>i</td>
<td>13</td>
</tr>
<tr>
<td>( A_2^x )</td>
<td>a</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>( A_3^x )</td>
<td>a</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>( A_4^x )</td>
<td>a</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>( A_5^x )</td>
<td>a</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>( A_6^x )</td>
<td>a</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

- **average number of objects per entry**: 1 | 100
- **average number of objects per block**: 20 | 95
- **block size**: 2000 | 2000
- **average number of accesses per entry**: 1.2 | 1.05
- **average direct access time**: \( t_d^o \) | \( t_d^p \)
- **average serial access time**: \( t_s^o \) | \( t_s^p \)

*Figure 4*
Under strategy \( S_2 \), serial scan of the object file, access time will be

\[
(10000/20) \times t_s^o = 500 \, t_s^o
\]

and transport size will be

\[500 \times 2000\]

positions. Let us summarize the results:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Access time</th>
<th>Transport size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( 1.05qt_d^p + 12000pt_d^o )</td>
<td>( 2000q + 10^7 \times 2p )</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>( 500t_s^o )</td>
<td>( 10^6 )</td>
</tr>
</tbody>
</table>

The first terms of the sums will certainly always be small compared to the second terms. This means, by the way, that it may well be worth considering a change of strategy from \( S_1 \) to \( S_2 \) even after the first step of \( S_1 \) has been carried out; actually carrying out this step could be the best way of estimating \( p \), the value of which determines which of \( S_1 \) and \( S_2 \) is best.

If we assume \( t_d^p = t_d^o = 40 \) ms and \( t_s^o = 1 \) ms we get

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Access time</th>
<th>Transport size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( 480p )</td>
<td>( 10^7 \times 2p )</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>( 2 )</td>
<td>( 10^6 )</td>
</tr>
</tbody>
</table>

Remark. In this example we have not considered the possibility of reducing the total direct access time to the object file by sorting the object references into the physical order of the file entries.
9.2.2 Other than α-attributes in the compound attribute key

In the basic retrieval situation defined in the previous section we assumed the key of the compound property file to be formed by all the α-attributes of query (q1), and no others. Now suppose that there is a compound property file, whose key is formed by the α-attributes $A_1^a, \ldots, A_n^a$ and some other attributes $A_1^x, \ldots, A_n^x$. Such a file may also be used in the first step of the processing of query (q1); the second step would still consist of accesses to the object file to look for the β-values.

Example. (Cf figure 5) Suppose we want the values of the β-attribute INCOME for all objects having the value B on the α-attribute REGION. There is a property file $F_1$ with the compound attribute key $\text{SEX} \times \text{REGION}$ and an object file $F_2$ containing the attribute INCOME. $R(\text{SEX}) = \{M, F\}$, $R(\text{REGION}) = \{A, B, C\}$, and thus $R(\text{SEX} \times \text{REGION}) = \{<M, A>, <M, B>, <M, C>, <F, A>, <F, B>, <F, C>\}$. A feasible way of processing the request would consist of the steps

(a) generate the entry points $<M, B>$ and $<F, B>$ using information about $R(\text{SEX})$

(b) access the two generated entries of $F_1$; this will yield a set of object references

(c) access the corresponding object entries of $F_2$ and fetch the INCOME-values

In general, if the compound property file key contains beside the α-attributes $n_x$ x-attributes, the ranges of which contain $r_1^x, \ldots, r_n^x$ values respectively, the processing of the query according to the strategy outlined above will generate accesses to

(d) $g = r_1^x \times \ldots \times r_n^x$
<table>
<thead>
<tr>
<th>attribute</th>
<th>file access type</th>
<th>$F^0_1$</th>
<th>$F^P_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>α REGION</td>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>β INCOME</td>
<td>a</td>
<td></td>
<td>c²</td>
</tr>
<tr>
<td>x CIVIC REG NO</td>
<td>i</td>
<td></td>
<td>i³</td>
</tr>
<tr>
<td>SEX</td>
<td>a</td>
<td></td>
<td>c¹</td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIONALITY</td>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5
different entries of the compound property file. If \( g \) is large in comparison with the total number of objects represented in the property file, many entries will be empty or non-existent. Nevertheless, it will usually cost at least one secondary storage access to learn that an entry is empty or non-existent. On the other hand, if \( g \) is small in comparison with the total number of objects represented in the property file, each generated entry will contain a lot of object references, maybe spread over several access blocks; we have then a situation very similar to that described in the previous section.

If \( g \) is not small it may in the general case be advantageous to replace the step corresponding to (b) in the example above with a serial scan of the compound property file.

**Example.** Suppose there are 10,000 objects and \( r_1^X = r_2^X = r_3^X = 10 \), and suppose there are on the average 100 object references per property file access block. A serial scan of the property file will cost some 100 serial accesses, whereas direct access after entry generation is likely to cause more than \( 10^3 = 1,000 \) direct accesses.

**Remark.** There is an important exception to the rule that a large \( g \) will cause at least an equally large number of accesses. This exception occurs when the entries of the compound property file have been "physically" or "logically" ordered according to the values of the \( \alpha \)- and \( x \)-attributes in such a way that the entries containing objects with the \( \alpha \)-property occur sequentially. Even if the hierarchical ordering should not be perfect with respect to the particular query, the number of accesses may be reduced.

**Example.** Suppose a property file with the compound attribute key \( \alpha_1^a, \alpha_2^a, X_1^x, \alpha_3^a, X_2^x, X_3^x \) has been hierarchically ordered in the way indicated by the writing of the key. \(^1\) Suppose that the range of each \( \alpha \)-attribute contains 10 values, and that the

---

\(^1\) The values of \( A_1^a \) vary most seldom and the values of \( A_3^x \) most often when we scan the entries of the file according to the hierarchical sequence.
range of each x-attribute contains 7 values. The processing of a query containing $A_1^\alpha$, $A_2^\alpha$, and $A_3^\alpha$ as the $\alpha$-attributes would then generate $7^3 = 343$ entries of the property file to be accessed. If say 300 of these entries were empty or non-existent many accesses would be in vain. However, the access work may be considerably reduced by taking the following 7 entries as origins for serial searches:

$e1: \langle v_1^\alpha, v_2^\alpha, 1, v_3^\alpha, 1, 1 \rangle$

$e2: \langle v_1^\alpha, v_2^\alpha, 2, v_3^\alpha, 1, 1 \rangle$

$e3: \langle v_1^\alpha, v_2^\alpha, 3, v_3^\alpha, 1, 1 \rangle$

$e4: \langle v_1^\alpha, v_2^\alpha, 4, v_3^\alpha, 1, 1 \rangle$

$e5: \langle v_1^\alpha, v_2^\alpha, 5, v_3^\alpha, 1, 1 \rangle$

$e6: \langle v_1^\alpha, v_2^\alpha, 6, v_3^\alpha, 1, 1 \rangle$

$e7: \langle v_1^\alpha, v_2^\alpha, 7, v_3^\alpha, 1, 1 \rangle$

$v_1^\alpha$, $v_2^\alpha$, and $v_3^\alpha$ are the constant values of the $\alpha$-attributes of the query. The ranges of the attributes are supposed to be initial segments of the set of positive integers.

If empty entries are left out in the property file the serial searches would actually start with the first non-empty successors of the entries stated above; only the $(343 - 300) = 43$ non-empty entries would be accessed. If the empty entries were not left out in the sequence, we would still have to access 343 different entries, but only 7 of these would be direct accesses, which are often more time-consuming than serial accesses, and many sub-sequences of requested property file entries would be contained in the same access block.

End of example.
With the types of secondary storage devices available today at most one ordering of the property file entries may be physically implemented at a time. By implementing more sequences, corresponding to different hierarchies of the attributes of a compound key, by means of data pointers, we may make it occur more often than otherwise that the actual number of property file secondary storage accesses necessary to process a query will be less than \( g \), as defined in (d). On the other hand the data pointers cause space costs and maintenance efforts.
9.2.3 \( \beta \)-attributes in the compound attribute key

If all \( \beta \)-attributes as well as all \( \alpha \)-attributes of a \((q1)\)-query are contained in the key of a compound property file, there is no need to access any object file. By letting the \( \beta \)-attributes vary independently of each other over their respective ranges, we obtain the entry point values of the property file entries to be accessed.

**Example.** (Cf figure 6). Suppose the value of the attribute SOCIAL CLASS is requested for all female inhabitants of region B, and suppose there is a property file with the compound attribute \( A = \text{REGION} \times \text{SOCIAL CLASS} \times \text{SEX} \). \( R(\text{REGION}) = \{A, B, C\} \), \( R(\text{SOCIAL CLASS}) = \{1, 2, 3\} \), and \( R(\text{SEX}) = \{M, F\} \); thus \( R(A) \) contains \( 3 \times 3 \times 2 = 18 \) elements, and there are equally many entries in the compound property file. The processing of the query causes access to 3 of these entries, namely

\[
\begin{align*}
e1: & \quad <B, 1, F> \\
e2: & \quad <B, 2, F> \\
e3: & \quad <B, 3, F>
\end{align*}
\]

The processing strategy illustrated by the example is particularly advantageous if the result of the \((q1)\)-query is to be sorted according to the values of the \( \beta \)-attributes. This might be the case, for instance, if the \((q1)\)-query is a sub-query of a table output request.

If, beside the \( \alpha \)- and \( \beta \)-attributes, the key of the compound property file contains one or more other attributes, \( x \)-attributes, the process planning process has to make considerations similar to those accounted for in the previous section to determine

- whether there is any hierarchical ordering of the entries of the compound property file, which may be used to reduce the actual number of accesses
<table>
<thead>
<tr>
<th>Attribute</th>
<th>File Access Type</th>
<th>$F_1^D$</th>
<th>$P_1^D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEX</td>
<td>a</td>
<td></td>
<td>$c_3^1$</td>
</tr>
<tr>
<td>REGION</td>
<td>a</td>
<td></td>
<td>$c_1^1$</td>
</tr>
<tr>
<td>SOCIAL CLASS</td>
<td>a</td>
<td></td>
<td>$c_2^2$</td>
</tr>
<tr>
<td>CIVIC REG NO</td>
<td>i</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIONALITY</td>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 6*
whether a serial scan of the property file is to be preferred to the direct access approach.

As a matter of fact these considerations are equally applicable when there are no $x$-attributes, because the generation of entries by variation of the $\beta$-attributes is perfectly analogous to the generation of entries by variation of $x$-attributes as described in the previous section; compare the example given above!
9.2.4 Scattered α-attributes

Consider a situation like that of figure 7. The α-attributes of query \( q_1 \) are scattered over several property files, and none of them contains all α-attributes. If we do not choose to scan the object file, we have to match entries of the property file in such a situation.

We shall look for the typical problems in connection with scattered attributes by studying the example of figure 7. To make the analysis a little more concrete there are a few numerical assumptions stated in the extra rows and column of the relevance matrix.

Let us examine four process design strategies, \( S_1, S_2, S_3, \) and \( S_4 \), which are all logically feasible. To describe the four strategies, we use an "almost-formal" short-hand notation, which is supposed to be self-explanatory.

**Strategy \( S_1 \)**

(1) match \( (P_1^d(A_1^α=v_1^α, A_3^α=v_3^α), P_3^d(A_2^α=v_2^α, A_4^α=v_4^α, A_5^α=v_5^α)) \)

giving \( \{o_1\} \)

(2) access \( (F_1^o, \{o_1\}) \) fetching \( (A_1^β, A_2^β) \)

**Explanation.** The entries of the property file \( P_1^d \) satisfying the condition \( (A_1^α=v_1^α) \land (A_3^α=v_3^α) \) should be matched against the entry of the property file \( P_3^d \) with the key \( \langle A_2^α, A_4^α, A_5^α \rangle = \langle v_2^α, v_4^α, v_5^α \rangle \). The objects resulting from this matching should be accessed from the object file \( F_1^o \); these object file entries contain the requested β-values.

**Analysis.** \( P_1^d \) contains \( r = 10 \) entries satisfying the condition \( (A_1^α=v_1^α) \land (A_3^α=v_3^α) \). The object references of each entry are sorted; \( A_1^x \) is the object identifying attribute according to the relevance matrix. Each of the ten entries is matched

\[ \text{ Cf section 9.2.2.} \]
<table>
<thead>
<tr>
<th>Attribute type</th>
<th>$F^C_i$</th>
<th>$F^P_1$</th>
<th>$F^P_2$</th>
<th>$F^P_3$</th>
<th>$F^P_4$</th>
<th>$10\log r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1^a$</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>$A_2^a$</td>
<td>a</td>
<td>c$^1$</td>
<td>c$^3$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$A_3^a$</td>
<td>a</td>
<td>c$^1$</td>
<td>c$^2$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$A_4^a$</td>
<td>a</td>
<td>c$^2$</td>
<td>c$^3$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$A_5^a$</td>
<td>a</td>
<td>c$^2$</td>
<td>c$^3$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$A_1^b$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>$A_2^b$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>$A_1^x$</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$A_2^x$</td>
<td>a</td>
<td>c$^3$</td>
<td>c$^1$</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$A_3^x$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$A_4^x$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>$A_5^x$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>$A_6^x$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

average number of objects per entry

average number of objects per block

block size

average number of accesses per entry

average direct access time

average serial access time

$r = \text{number of attribute values}$

total number of objects = $10^6$
against the single entry of $P_3^d$ with the particular $\langle A_5^a, A_2^a, A_4^a \rangle$ key; these object references are sorted, too. According to the numerical assumptions of the example an expected number of

$$10 \times 3.0 + 1 \times 3.0 = 33$$

accesses have to be made to the property files in order to accomplish the matching. The expected transport size is

$$33 \times 4000 = 132000$$

decimal positions (pos). We make the additional assumption that

$$10^6 / r^5 = 10$$

object references "survive" the matching process. Then on the average

$$10 \times 1.2 = 12$$

accesses will have to be made to the object file $P_1^o$, and

$$12 \times 2000 = 24000$$

pos will have to be transported.

**Strategy $S_2$**

(1) match $(P_2^d(A_1^a = v_1^a, A_2^a = v_2^a, A_4^a = v_4^a), P_4^d(A_3^a = v_3^a, A_5^a = v_5^a))$

   giving $\{ o_1 \}$

(2) access $(P_1^o, \{ o_1 \})$ fetching $(A_1^\beta, A_2^\beta)$

**Explanation.** The entries of the property file $P_4^d$ satisfying the condition $(A_5^a = v_5^a) \land (A_3^a = v_3^a)$ should be matched against the entry of the property file $P_2^d$ with the key $\langle A_2^a, A_4^a, A_5^a \rangle = \langle v_2^a, v_4^a, v_5^a \rangle$. The objects resulting from this matching should be accessed from the object file $P_1^o$; these object file entries contain the requested $\beta$-values.
Analysis. The situation is perfectly analogous to that of strategy 1. The analysis and the calculations will be the same.

Strategy S_3

(1) match \( (P^D(A_1^a=v_1^a, A_2^a=v_2^a, A_4^a=v_4^a, A_2^a=v_2^a, A_4^a=v_4^a) \), \( (v_3^a, A_2^a=v_2^a) \) giving \( \{o_1\} \)

(2) access \( (F_1^O, \{o_1\} \) testing \( \{v_3^a\} \) and fetching \( \{A_1^a, A_2^a\} \)

Explanation. According to this strategy we match the entry of \( P_2^D \) with the key \( \langle A_2^a, A_1^a, A_4^a \rangle = \langle v_2^a, v_1^a, v_4^a \rangle \) against the entry of \( P_3^D \) with the key \( \langle A_2^a, A_1^a, A_4^a \rangle = \langle v_2^a, v_1^a, v_4^a \rangle \). By giving up the condition \( \langle A_3^a = v_3^a \rangle \) in the matching process we only have to match two entries against each other, but on the other hand more objects will "survive" the matching process; more precisely we make the additional assumption that

\[ 10^{6}/10^4 = 100 \]

objects will have to be accessed in the second phase. During the latter process we have to test the value of \( A_3 \) in the transmitted object file entries so that the requested \( \beta \)-values are fetched in the appropriate cases only.

Analysis. According to the numerical assumptions of the example an expected number of

\[ 3.0 + 3.0 = 6 \]

accesses have to be made to the property files in order to accomplish the matching. The expected transport size during this phase is

\[ 6 \times 4000 = 24000 \]

pos. Then on the average

\[ 100 \times 1.2 = 120 \]
accesses will have to be made to the object file $F_1^o$, and

$120 \times 2000 = 240000$

pos will have to be transported.

**Strategy $S_4$**

(1) \[ \text{scan } (F_1^o) \text{ testing } (A_1^a = v_1^a, A_2^o = v_2^a, A_3^a = v_3^a, A_4^o = v_4^a, A_5^o = v_5^a) \]

and fetching $(A_1^b, A_2^b)$

**Explanation.** This is the serial search strategy, which may sometimes be better than any directory/file search, for instance because each serial access is usually faster than each direct access, which may outweigh the effect of a possibly greater number of accesses with the scanning strategy.

**Analysis.** According to the numerical assumptions of the example

$10^6/20 = 50000$

accesses have to be made during the scanning of the object file $F_1$. The transport size will be

$50000 \times 2000 = 10^8$

decimal positions.

**Summary**

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Access time</th>
<th>Transport size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$33t_d^p + 12t_d^o$</td>
<td>156 000</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$33t_d^p + 12t_d^o$</td>
<td>156 000</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$6t_d^p + 120t_d^o$</td>
<td>264 000</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$50000t_s^o$</td>
<td>100 000 000</td>
</tr>
</tbody>
</table>
If we assign reasonable values to the parameters, we find that $S_1$ and $S_2$ are the best strategies in this example.

Remark. According to $S_1$ and $S_2$ we had to match one entry of one file against several entries of another file. With slightly different assumptions we might have had to match many entries of one file against many entries of another file. Then, if we cannot keep all involved entries of at least one of the files in primary storage simultaneously, we have to sort the object references of each entry, if this has not been done before, merge the object references of different entries of the same property file and store them as a temporary file on secondary storage, before we can start the matching process as described above. The sorting and merging will require extra CPU time, which might not be important, and extra accesses when writing and reading the temporary files, which will probably have significance, although these small and temporary sets of data will of course be stored on much faster secondary storage devices than the permanent data base files.
9.2.5 Overlapping relevant files

Let us investigate a few typical examples of process design situations, where at least two of the relevant files overlap each other.

First consider the example of figure 8. We have there a relevant object file, containing among other things the β-attributes, and this object file has been overlapped by three property files, each of which contains one of the α-attributes of the query. More precisely, it is the object identifying values of the attribute $A_1^X$ that the four files have in common. To show this we have affixed a subscript "o" to the "i"s in the $A_1^X$ row.

In the first extra row below the kernel of the relevance matrix, the existing overlaps have been indicated. We distinguish between overlapped files, indicated by "x", and overlapping files, indicated by a reference to the overlapped file. It is difficult to give a general, formal definition of this distinction, but a little loosely one could say that of two overlapping files we call that one overlapped which has been least affected to its inter- and intra-entry structure by the overlap operation.

If we disregard for a moment the overlap relationships between the files, figure 8 shows a situation characterized by scattered α-attributes. According to 7.4.4 the typical strategy to use in such a situation would be to match property files against each other and access entries in the object file corresponding to objects surviving the matching process. According to the numerical assumptions of figure 8 this strategy would cause an expected number of

$$2000 + 1000 + 500 = 3500$$

direct accesses to the property files to be followed by, say

$$10000/(r_1^o \times r_2^\alpha \times r_3^\alpha) = 10000/(5 \times 10 \times 20) = 10$$

---

1 The relevant files of a query are the files of the columns of the relevance matrix of the query.
<table>
<thead>
<tr>
<th>attribute type</th>
<th>$p_0$</th>
<th>$p_1^D$</th>
<th>$p_2^D$</th>
<th>$p_3^D$</th>
<th>range cardinality</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>$\tau$</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>overlaps</th>
<th>$x$</th>
<th>$p_0^F$</th>
<th>$p_1^F$</th>
<th>$p_2^F$</th>
<th>10</th>
<th></th>
</tr>
</thead>
</table>

| average number of objects per entry | 1 | 2000 | 1000 | 500 |
| average number of objects per block | 5 | 1 | 1 | 1 |
| block size | 700 | 700 | 700 | 700 |
| average number of accesses per entry | 1.3 | 2000 | 1000 | 500 |

<table>
<thead>
<tr>
<th>average direct access time</th>
<th>$t_d$</th>
<th>$t_d$</th>
<th>$t_d$</th>
<th>$t_d$</th>
</tr>
</thead>
</table>

| total number of objects = 10 000 |

**Figure 8**
direct accesses to the object file. Of course it would be stupid not to recognize and take advantage of overlap relationships that do exist. With the assumptions of the example it would obviously be a much better strategy

(a) to access the appropriate entry of the relevant property file with the least average number of objects per entry, i.e. $F_3^p$ and

(b) to group this property file accessing process with a process during which the object file entries overlapped by the accessed property file entries are tested upon their $A_1^a$- and $A_2^a$-values; in appropriate cases the $b$-values are fetched.

This strategy would cost only

$$\min(2000, 1000, 500) = 500$$

direct accesses to the $(F_3^p, F_1^o)$ file overlap pair.

**Remark.** With the latter strategy it does not make us any better off that the object references within each property file entry are sorted, which they are according to the assumptions of figure 8. With the former strategy, however, this assumption could be essential.

Next we consider the example of figure 9. There it is the property file that has been overlapped by an object file. More concretely the situation could be as follows.

---

1. Note that we discuss here the strategy to choose at processing time, when we have to accept the existing file structure. Quite another problem, to be considered at file structuring time, is whether at all files should be overlapped with each other. If the files of the example had not been overlapped in the first place, there would certainly have been more than 1 object per block in the property files, and the matching would not have caused so many accesses.
<table>
<thead>
<tr>
<th>file access attribute type</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\alpha</code></td>
<td>( A_1^\alpha )</td>
<td>( a )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td></td>
<td>( A_2^\alpha )</td>
<td>( a )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td></td>
<td>( A_3^\alpha )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td><code>\beta</code></td>
<td>( A_1^\beta )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( A_2^\beta )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td><code>\chi</code></td>
<td>( A_1^\chi )</td>
<td>( i )</td>
<td>( i_0 )</td>
</tr>
<tr>
<td></td>
<td>( A_2^\chi )</td>
<td>( a )</td>
<td>( i_0 )</td>
</tr>
<tr>
<td></td>
<td>( A_3^\chi )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( A_4^\chi )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( A_5^\chi )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( A_2^\chi )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td>overlaps</td>
<td></td>
<td>( P_1 )</td>
<td>( x )</td>
</tr>
</tbody>
</table>

Figure 9.
\[ A_1^\alpha = v_1^\alpha, A_2^\alpha = v_2^\alpha \]

\[
\begin{array}{l}
A_1^x \{ 0_1 \}, A_3^\alpha \{ 0_1 \} \\
A_1^x \{ 0_2 \}, A_3^\alpha \{ 0_2 \} \\
A_1^x \{ 0_3 \}, A_3^\alpha \{ 0_3 \} \\
A_1^x \{ 0_n \}, A_3^\alpha \{ 0_n \}
\end{array}
\]

that is, an access block containing references to objects having the property

\[(A_1^\alpha = v_1^\alpha) \land (A_2^\alpha = v_2^\alpha)\]

also contains, beside each object reference \(A_1^x \{ 0_1 \}\), the value of the attribute \(A_3^\alpha\) for each object having the property stated above.

If \(P_2^O\) had not been overlapping \(P_1^D\) as indicated by figure 9 we would have had to use a strategy consisting of the steps

(a) access the entry \(\langle v_1^\alpha, v_2^\alpha \rangle\) of the property file \(P_1^D\)

(b) for each object reference in the property file entry

access the object file \(P_1^O\) test \(A_3^\alpha\) for \(v_3^\alpha\) and fetch the \(\beta\)-values in appropriate cases

Recognizing the overlap we may instead use a strategy consisting of the steps

(a) access the entry \(\langle v_1^\alpha, v_2^\alpha \rangle\) of the property file \(P_1^D\)
    and test the overlapped \(P_2^D\) entries for \(A_3^\alpha = v_3^\alpha\)

(b) for each object reference surviving the \(A_3^\alpha\)-test

access the object file \(P_1^O\) and fetch the \(\beta\)-values
Under normal circumstances, the latter strategy will of course imply fewer accesses than the former.

As a third example of process design in connection with overlapping files we consider the situation represented by figure 10. We suppose that this situation has been caused by a query asking for

"the median income of all females in region C".

This query is not itself of type (q1) but a straight-forward analysis would generate the (q1)-query

"for all persons having the property \((\text{SEX} = F) \land (\text{REGION} = C)\) retrieve the values of the attribute INCOME"

The retrieved INCOME values would then be sorted and the one in the middle would be delivered as the reply to the original query.

Processing along these lines would involve the property file \(F_1^p\) and the object file \(F_1^o\); this kind of processing design problem has already been treated in 7.4.2. It is well worth considering, however, to make use of the object file \(F_2^o\), which is overlapping \(F_1^o\).

It is the key \(\text{REGION} \times \text{SEX} \times \text{INCOME CLASS}\) of the compound object file \(F_2^o\) which overlaps the key \(\text{REGION} \times \text{SEX} \times \text{INCOME CLASS}\) of the compound property file \(F_2^p\). This means that the objects referred to in \(F_2^o\) are so-called compound or aggregate objects, each of which is defined as a set of objects having a particular, possibly compound, property in common. Beside the object identifying compound attribute stated above, \(F_2^o\) contains only the attribute FREQUENCY. The value of this attribute for a particular compound object is assumed to give the number of individual objects making up the compound objects, or, in other words, the number of persons having a particular \(\text{REGION} \times \text{SEX} \times \text{INCOME CLASS}\) combination.
<table>
<thead>
<tr>
<th>attribute</th>
<th>file access type</th>
<th>$F_1^\circ$</th>
<th>$F_2^\circ$</th>
<th>$F_1^P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>REGION</td>
<td>$i_1^\circ$</td>
<td>$i_2^\circ$</td>
<td>$c_1^\circ$</td>
</tr>
<tr>
<td></td>
<td>SEX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>INCOME</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{x}$</td>
<td>CIVIC REG NO</td>
<td>$i$</td>
<td></td>
<td>$i_1^4$</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDRESS</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCCUPATION</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INCOME CLASS</td>
<td></td>
<td>$i_3^\circ$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FREQUENCY</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

overlaps

Figure 10
An access block of the $<p_2^o, p_1^p>$ file overlap pair could have the following appearance

| REGION = x |
| SEX = y   |
| INCOME CLASS = z |
| FREQUENCY = n |

$0_1$
$0_2$
$0_3$

$0_n$

Taking the file structure, as given by the relevance matrix, into account, a feasible and efficient way of processing the original query would be the following

(a) for each income class $z_i$ ($i = 1, ..., z$) access the entry $<c, p, z_i>$ of $p_2^o$ and store the frequency value of the compound object in a primary storage table

(b) calculate cumulated frequencies according to the ordering that is assumed to exist among the income classes; find the middle income class $z_{i_0}$

(c) access the entry $<c, p, z_{i_0}>$ of $p_1^p$

(d) access the object entries of $p_1^o$ referred to in the property file entry and calculate the median value of the INCOME attribute among these objects

We see that by using the compound object file in a few preliminary steps of processing we limit considerably the access and calculation work that has to be done in the last step.
This last example could also be used to illustrate a desirable data base function that we may call "interactive modification of user requests", IMUR. When the processing of the median income query has passed step (b) above the IMUR function could present the partial result that

"the requested median is within the income interval $z_{10}$" and ask the user if he is satisfied with this reply to his query. The IMUR function could also supply the user with information about how much it would cost in terms of hours and dollars to proceed through steps (c) and (d) in order to get a more precise answer.
10. Summary and bibliography

10.1 The infological approach in retrospect

In chapter 1 of this report we argued that there is an urgent need for so-called infological models in the development and operation of data bases. We even suggested that the lack of such models might be an important explanation of the fact that there are surprisingly few successful implementations of concepts like "integrated file system", "management information system", and "data base system". As long as the data representations handled by computers covered only small, non-integrated slices of reality, the "reality \rightarrow storage" mappings were fairly straight-forward. The development and operation of information systems, which were by and large automated versions of old manual systems, could be satisfactorily managed by subject matter specialists with only a minimum of EDP education, or by EDP specialists who had been briefed about the subject matter area. As a contrast, the development and operation of large integrated information systems requires precise communication and efficient cooperation between lots of people with different specialties and different educational backgrounds. Infological models specified by the project teams in accordance with the theory and guidelines developed in this report could hopefully serve as adequate communication interfaces in large data base undertakings involving people from several fields.

Infological models are intermediary models. They are related to, but not mere generalizations of computer-oriented people's "datalogical" models of organized sets (files) of data. Similarly, they are related to, but not identical with any of the subject matter models used in economical, sociological, technical, and other disciplines. As was shown by figure 5 in chapter 1 the development of data base systems in accordance with the methodology recommended in this report involves the mapping of subject matter models into infological models, and the mapping of infological models into datalogical, computer-oriented models. The two mapping processes are called "the specification process" and "the design process" respectively.
During the specification process the concepts and rules of infological theory should be used. The conceptual framework which is the foundation of infological database theory was defined and discussed in chapters 2 and 3 of this report. The infological framework contains concepts belonging to two distinguishable spheres: "the object system sphere", and "the information sphere". The object system is the slice of reality that the database should contain information about. According to infological theory the building blocks of the object system are objects, properties, object relations, and times. The building blocks may be combined into certain basic structures, called elementary constellations, or e-constellations. E-constellations which have existed, exist, or will exist are called e-facts. The e-facts of an object system form a subset of the set of valid e-constellations, which is made up of all e-constellations which are not "unthinkable".

It is often practical to group the object system entities into classes of "similar" entities, entities "of the same kind". Thus a set of individual objects may constitute an object group or an object type, properties form attributes, and e-constellations make up e-constellation types.

The information sphere of the infological framework comprises concepts by means of which information about an object system may be formally described and analyzed. The reference concept is fundamental in this sphere. References refer explicitly or implicitly, uniquely or ambiguously to objects, properties, object relations, times, e-constellations, attributes, and other object system entities. They may be combined into reference expressions, which are themselves references with object system targets. The basic structures of references are called elementary messages, or e-messages. E-messages refer to e-constellations, and they may be regarded as "minimal" information structures. A complete e-message will convey well-defined knowledge to a person with a compatible frame of reference, whereas a sub-reference of an e-message, which is not identical with the complete e-message itself, will not do so.
It is important to distinguish between the infological frame-work and infological theory on the one hand, and particular infological models on the other. It is only the latter, specified for particular data bases, which are models of reality. The infological frame-work provides the concepts by means of which these models are formulated, but it is not itself a model of reality. Possibly it could be regarded as a general meta-model of all particular infological models.

One consequence of our separation of the general infological frame-work from the particular infological models is that infological theory alone will never suffice to determine what particular infological model should be chosen for a particular data base. The specification of certain entities as objects, properties, object relations, etc, will always ultimately be a question of judgment. What general infological theory could do is to provide rules of thumb or even more formal guidelines to the specification decisions. Infological theory could also help by systematically analyzing the consequences of looking at the same piece of reality through different infological glasses.

Thus the problems which will appear during a specification process cannot always be solved by a glance into a manual. There will have to be discussions within the project group which is responsible for the development of the data base. Hopefully infological theory could help to structure these discussions and to make them converge towards a particular model which is understood and accepted by all members of the project team.

The design process is the process which logically succeeds the specification process during the development of a data base system. Chapters 4-7 of this report are intended to show that a lot of important data base design problems may be stated and analyzed in precise infological terms without anticipating datalogical decisions as to how the information contents of the planned data base should be represented and structured into entries and files.
From an infological point of view a data base is a black box reservoir of e-messages, an "artificial" extension to the "natural" information sources of the people using it. With the infological approach "artificial" data bases and "natural" ones, contained in human minds, seem to have a great deal in common. Thus figure 1 in chapter 1 and figure 3 in chapter 4 may be applied to both categories of data bases.

Figure 3 in chapter 4 identifies three basic functions of the infological data base: the schema, the nucleus, and the filter. The schema subsystem contains the particular infological model which has been specified for the data base. It is the origin of semantic and deductive capabilities. The nucleus contains the e-messages which are physically stored in the base. Together the schema and the nucleus determine the effective information contents of the data base. The filter subsystem should protect recognized quality, integrity, and privacy interests of data base users and information suppliers and of other concerned parties, e.g., persons about whom information is stored.

It is important to realize that a data base always has an environment, and that this environment is not at all homogeneous but consists of several identifiable subsystems with different requirements and desires. In chapter 4 we proposed the division of a typical data base environment into eight subsystems: the observed object system, the controlled object system, information supply, information consumption, data base goal stating and decision making, data base administration, other data bases, and the computer subsystem. In chapter 5 it was pointed out that this subdivision could be used among other things for the identification of major types of interactions and transactions between the data base and its environment.

As another classification basis for interactions and transactions between the data base and its environment we used in chapter 5 the so-called S-O-R model. This led us to
a discussion of "spontaneous" versus "triggered" data base behavior and to further sub-classification of the basic interaction categories. Recall figure 2 in chapter 5. After a systematizing analysis of the infological effects of different kinds of data base modification transactions, we turned our interest to retrieval queries.

Equipped with the infological concepts introduced in chapters 2 and 3, we found in chapter 5 that a broad category of retrieval queries conform to one and the same basic infological pattern, the αβ-query pattern. This pattern covers data base queries which request the retrieval of single e-messages or unstructured conjunctions of single e-messages.

In chapter 6 we found that the αβ-pattern may in a natural way be generalized into an αβγ-pattern covering queries which request (a) structured conjunctions of e-messages, (b) fabrication of so-called aggregate messages, or a-messages, from sets of e-messages, and (c) structured conjunctions of a-messages. Lists of e-messages sorted by one or more arguments, and multi-dimensional tables of averages and percentages are examples of what a data base system might produce in response to common types of αβγ-queries.

In connection with the identification and analysis of the two basic retrieval query patterns we also discussed in chapters 5 and 6 how the ultimate data base users could and should be able to communicate their information needs. On the one hand natural language was ruled out as an inadequate communication mode for broad categories of information consumers with precise, complex information needs. On the other hand it was argued that it is highly desirable and perfectly realistic to develop data base interaction languages which permit the information consumer to express his information needs in a result-oriented, non-procedural way. Hopefully suggestive examples of such expression modes, based upon the general infological frame-work, were discussed.
In the communication and illustration of information needs both verbal and graphical techniques should be exploited. It needs further investigation to determine which techniques or mixes of techniques should be used in which communication and illustration situations. Another design problem which calls for further research, even on the infological level, is how a data base system should best assist an information seeker, who is not acquainted with the particular infological model underlying the data base with which he is interacting. A similar problem, which is likely to appear in combination with the former, is the problem with the information seeker who is only vaguely aware of his own information needs when he starts his conversation with the data base system.

It should be a challenging task for designers of future data base systems to design the software and languages of these systems so as to meet advanced infological requirements. The designs should make it possible for the information consumers to fully utilize the information potential of complex data bases without having to know anything about the internal, datalogical properties of the information reservoir. By means of result-oriented languages it should be possible to create and maintain a feeling with the ultimate user of a data base that he is interacting with a black box of infologically structured information, rather than with a set of data files structured on datalogical grounds into consolidated files, inverted lists, linked networks, and so on. This design task is difficult, particularly when the information needs to be communicated between the information consumer and the data base are themselves infologically complex. Yet it is hoped that this report has shown that the task could and should be attacked now.

For those who are to perform advanced data base design tasks the infological view of a data base as a black box reservoir of information will certainly not be sufficient. However, it is highly desirable that the datalogical concepts and models that the designers use are compatible with the infological approach. Only then will it be possible to "translate"
external, infological requirements upon a database into internal, datalogical database properties in a systematical way. Unless such "translations" are possible, the whole idea of an infological approach to databases could rightfully be questioned. In chapters 8 and 9 of the report we tried to demonstrate that the infological framework may in fact be extended in a very natural way so as to cover the computer-oriented, representational aspect of a database as well. The gap between the infological and the datalogical sphere is bridged by our file concept which on the one hand connects very smoothly on to the infological message type concept, but which on the other hand seems to allow all known file structures of any empirical importance to be systematized in a formal, though straightforward way.

Whether we approach the database concept from the infological or the datalogical side, a database is a system. From an infological point of view a database may be regarded as a system consisting of the schema, nucleus, and filter subsystems and interacting by means of information transactions with an environment, which in turn may be divided into several subsystems. From a datalogical point of view a database may be looked upon as a system of processes using, producing, and transforming database internal and external resources, e.g. data resources. Both as an infological and as a datalogical system a database is typically an imperceivable system. Thus general systemeering theory and methodology for imperceivable systems should be applicable to database design problems. This was discussed in chapter 7 of the report. We found that there are four kinds of activities which have to go hand in hand during the systemeering of a database, namely (a) the definition and analysis of the goals of the database, (b) the definition and analysis of a functional subsystem structure of the database, (c) the transformation of goals on a higher functional level into equivalent goals on a lower functional level, and (d) the specification of a particular infological model for the database.
As a high-level goal structure which should be appropriate for most databases to be designed, we suggested a goal structure with four components: "usefulness", "non-harmfulness", "economy", and "viability". As a high-level functional structure which should similarly be worth considering in any database project, we suggested a structure consisting of eight subsystems, visualized in figure 12 of chapter 7. Finally we exemplified how it should be possible to continue the break-down into lower-level goals and functions.

When the systemeering of a planned database has been driven to a sufficiently detailed level, the bulk of data base design problems may be sorted into two categories: "file design problems" and "process design problems". In chapters 8 and 9 of the report we tried to analyze these problem areas systematically with the conceptual tools of the datalogically extended infological frame-work. We recognized many sub-problems which have already been thoroughly investigated in the literature, but we also discovered more or less white areas.

In chapter 8 we defined a set of file structuring operators by means of which it should be possible to transform any set of e-files, reflecting directly the e-concepts of a particular infological model, into an almost arbitrary datalogical file structure. Formally defined file structuring operators could be of great help in the systematization and documentation of "manually" performed file design. If file design and redesign is to be fully or partially automated, formally defined file structuring operators, or equivalent tools, are indispensable.

For many databases the file design task will in itself contain a difficult systemeering problem because of the large number of e-files involved. We discussed in chapter 8 how this problem may be tackled by means of a so-called θ-matrix. Such a matrix is filled with rough estimates of the relative frequencies or "importances" of different kinds of θ-queries, and on this basis the file system may be
partitioned into more tractable subsystems, so-called \(\alpha\beta\)-complexes, or directory/file-complexes. An exemplifying, quantitative analysis of the problem of designing a directory/file-complex was also given in chapter 8.

Chapter 9 was devoted to a study of the planning, coordination, and scheduling of data base processes. We used the term "process design" for these kinds of activities. We found that there are characteristic differences between "conventional systems", and data base systems with respect to the process design problems. As to data base systems many process design problems cannot be solved at design time, before the system is put into operation. Instead they have to be solved automatically and dynamically at operation time by the data base system itself. Among other things a data base system has to perform precedence analysis and solve process grouping problems dynamically. These are tasks which are normally performed by possibly computer-assisted system engineers, programmers, and operating staff in the development and running of a "conventional" data processing system.

A substantial part of chapter 9 was devoted to a study of transaction-oriented process design problems in connection with retrieval queries of \(\alpha\beta\)-type. Some typical processing situations and strategies were identified and accompanied by exemplifying quantitative analyses. For illustrations we used a tool called "relevance matrix". It was argued that such matrices could turn out to be useful internal working tools for the data base systems themselves.

We hope to have convinced the reader of this report that the infological approach to data bases is an approach which a little better than alternative approaches looks after the justified interests of the ultimate data base users, at design time as well as at operation time. We also claim to have shown that the advantages of the infological approach do not require in return any real sacrifices, but certainly some re-indoctrination, on the part of datalogically oriented computer professionals. Finally we believe that
the infological approach lays a long missing solid ground for further research on data bases and data base design. The evidence in support of this hypothesis is scattered all over the report, which may in fact be regarded as a problem survey and a research program to at least the same extent as it is an account of research results which have already been achieved.
10.2 Bibliographical remarks

In this report I have tried to accomplish an integrated view towards the myriad of problems facing the designers of a large-scale data base. The basic method has been to attack the subject from the infological angle of approach. Much has been written about data processing problems in connection with data bases. However, it has seldom been attempted to relate the different data base problems to each other by giving them well-defined places in an integrated theoretical framework, and when such attempts have actually been made, it has usually not been from an infological point of departure. As a matter of fact, what I have labeled as "the infological approach" is an approach which is little known by people who have not come in contact with "the Scandinavian tradition" in information processing.

It should not be surprising then that the number of books and articles which are highly relevant to this report is relatively low. On the other hand there is an almost inexhaustible quantity of "semi-relevant" literature. It has not been feasible to include all imaginable references of the latter kind in the bibliography of this report. Instead I have tried to assemble a sample of references which together with their direct references should provide a reasonably complete collection of literature worth consideration in further data base research. Some of the Scandinavian papers in the list of references may be difficult to retrieve in other places of the world. A good idea might then be to contact me under the address of

Bo Sundgren
National Central Bureau of Statistics (SCB)
Fack
S-102 50 STOCKHOLM
Sweden

Information exchange with data base researchers will be welcome!

The references contained in the bibliography have been roughly classified into a number of groups. Group A contains references
to some of the philosophical, mathematical, psychological and
linguistical literature with bearing upon the reported research.
As has been pointed out in the report, there are several re-
search areas which call for joint efforts by behavioral scientists,
data base specialists, linguists, and others. Naturally, the
objectives of researchers from different disciplines are only
partially overlapping. For example, psychologists and linguists
first of all want to understand prevailing phenomena like the
behavior of human beings of flesh and blood, and the nature and
structure of natural languages. A data base researcher, on the
other hand, first of all wants to explore how to design artifi-
cial extensions to the human mind, for example data bases and
more or less artificial languages for interaction between "na-
tural" human beings and the designed artifacts. However, slightly
differing objectives of the members of joint research projects
should only stimulate progress within the disciplines concerned.

The literature in group B of the list of references deals with
such things as the nature of systems and systems design and
control. Literature on goal-stating and decision-making also
belongs to this group. As was emphasized in chapter 7 of the re-
port there is not at present a unanimous approach to systems.
Thus authors like Ashby, Churchman, Langefors, and Miller & Starr,
who are all included in the bibliography, represent different
lines of thinking.

In group C I have collected some works with a relatively broad
and deep coverage of fundamental information processing and data
base problems. Here are found several books and reports belonging
to "the Scandinavian tradition" in information processing, of
which this report is an outgrowth. The works of Langefors (48)
and Nordbotten (55) are the classics in this tradition. The book
by Lefkovitz (51) is another classical work which has inspired
me and many other researchers in the data base and file organiza-
tion field. Basic philosophical problems in connection with data
bases and information systems have been thoroughly treated by
Churchman (44) and his followers (53).

Group D in the bibliography contains some of the articles and
reports which explicitly alleges to deal with data base problems.
Many of the references included should be well-known to those
who have followed the debate on "data base management systems" which has taken place much as a result of the CODASYL proposal (75), the first version of which appeared in 1969. Hopefully this report has shown that the problems treated in the DBMS debate, and tackled by manufacturers of so-called generalized data base management systems, constitute only a small subset of "all" problems in connection with the design and operation of data base systems. The CODASYL proposal has made the debate circle very much around the problem area of user languages, where "user" does not mean the ultimate user of the data base, i.e. the information consumer, but rather "application programmer". Languages for other data base interactors, e.g. information consumers, information structurers, and data structurers (cf section 5.2.1) have not been very much discussed.

Many people in the data base field have devoted a lot of time and energy to systematic description of existing data base management systems. See for example references (73), (74), and (84). However, it is important not to confuse the feature lists and similar systematization tools used in such undertakings with theoretically well-founded conceptual frameworks.

Systematic descriptions of existing languages and software, like the ones made by CODASYL and MITRE, have their primary interest as market surveys at a particular point of time. If one wants to create a basis for future research and development it seems sounder to do what the joint GUIDE-SHARE data base group have done in (83), i.e. to present a list of requirements and desirables. In combination with a general data base theory such lists could create a sound environment for fast, demand-controlled progress in the data base field.

Since the so-called "Codd's Wallop" (78), there has been much ado in the computing world about "relational data bases". No doubt Codd has given valuable contributions to any general data base theory to emerge. However, the marketing of Codd's ideas might have given somebody the false impression that the relational approach to data bases would solve the bulk of data base problems, and in particular that it would solve most user problems; once again "user" means here "application programmer" rather than "ultimate user" or "information consumer".
The difference between Codd's relation concept and the object relation concept used in this report should be pointed out. They are both compatible with the mathematical relation concept, but whereas the domains of "object relations" may only contain objects, the domains of "Codd's relations" may contain entities belonging to any infological category, e.g., attribute values and times. As a result Codd's relations have the character of normalized files, dynamically created by an application programmer, whereas object relations are meant to have an intuitive appeal to reality-oriented information consumers without data processing knowledge. Object relations are often binary and seldom have degrees greater than three. Codd mentions relations of degree 32 as perfectly realistic.

Despite what has been said in the "CODASYL versus IBM" DBMS debate, and despite what has been said in this report, it would be far too pessimistic to regard "the CODASYL school", "the IBM school", and "the infological school" as three separate lines of data base thinking, without possibilities of being synthetized into a unanimous general data base theory. I regard the article by Senko et. al. (58), which has just been published, as a token of this. The article seems to indicate that "the IBM school" is moving towards an infological view of data bases which is similar to the approach advocated in this report.

In group E of the bibliography I have collected references to papers and books dealing with particular data base functions. The data base subsystem structure suggested in chapter 4 of the report has been used in order to classify the material into sub-groups. The reports in sub-group E.1 deal with the data base schema function and deductive sub-functions. Sub-group E.2 containing literature on the data base nucleus function has been further sub-divided into E.2.1 and E.2.2. E.2.1 contains material on the problem of designing a file system (cf. chapter 8 in the report), whereas the papers and books in E.2.2 concentrate upon the problems of designing, scheduling, and coordinating the processes of data bases and other data processing systems (cf. chapter 9 in the report). The books and papers in sub-group E.3 are relevant to the problems in connection with the data base filter function and
its quality and protection sub-functions.

The references in group F of the bibliography constitute a small sample of the host of articles which have been written on specific design problems, like how to organize a particular file (rather than a file system), how to tackle specific problems which are typical of specific file organization methods, and how to decide upon block lengths.

Finally, group G in the bibliography contains references to some papers which are relevant to the material in the report although they do not fit into any of the categories A–F. For example, reference (207) is the manual of a programming language for production of statistical tables. The language has been developed at the National Central Bureau of Statistics in Sweden, and it represents a step in the direction of more result-oriented languages, i.e., a step in the direction propagated for in this report.
10.3 List of references

A. Philosophy, mathematics, psychology, and linguistics

1. Antal, L.
   Content, Meaning, and Understanding
   Mouton & Co. The Hague, 1964

2. Ayer, A.J.
   Language, Truth, and Logic
   London 1936

3. Glusjkov, V.M.
   Simulering av mänskligt tänkande/Simulation of Human Thinking
   Undated document from the USSR Academy of Sciences

4. Gårding, L.
   Modell och verklighet
   Dagens Nyheter 1966-07-21

5. Hempel, C.G.
   Fundamentals of Concept Formation in Empirical Science
   University of Chicago, 1952

6. Madsen, K.B.
   Almen psykologi
   København 1966

   Human Problem Solving

8. Rudén, B.
   Språkliga problem vid intervjuer och enkäter, en del-rapport
   National Central Bureau of Statistics, Sweden, 1973

9. Sigurd, B.
   Språkstruktur
   Wahlström & Widstrand, Stockholm, 1967
10. Suppes, P.
   Introduction to Logic
   Van Nostrand, 1957

11. van der Waerden, B.L.
    Moderne Algebra
    Springer, Berlin, 1931

12. Wedberg, A.
    Filosofins historia, I-III
    Bonniers, Stockholm, 1958, 1959, 1966

13. Wärneryd, K.E.
    Ekonomisk psykologi
    Natur och Kultur, Stockholm, 1967
B. Systems, goals, and economy

21. Ashby, W.R.
An Introduction to Cybernetics
Chapman & Hall, London, 1956

22. Carzo, R. and Yanouzas, J.N.
Formal Organization, A Systems Approach
Irwin-Dorsey, 1967

23. Churchman, C.W.
The Systems Approach

24. Emery, J.C.
Organizational Planning and Control Systems
Macmillan, New York, 1969

25. Holmberg, P.
I stället för slutrapport
Tiden 1972:1

26. Langefors, B.
System för företagsstyrning
Studentlitteratur Sweden, 1968

27. Langefors, B.
Satisficing, a "must" for system control
University of Stockholm 1970 (IB-ADB 70:4)

28. Marschak, J.
Problems in Information Economics

29. McKean, R.N.
Efficiency in Government through Systems Analysis
Wiley, 1958
30. Miller, D.W. and Starr, M.K.
   Executive Decisions and Operations Research
   Prentice-Hall, 1960

31. Musgrave, R.
   The Theory of Public Finance
   New York 1959
C. Fundamentals on information processing and databases

41. Arvas, C.
Joint Use of Data Banks
(Conf. Eur. Stats./WG.9/78)

42. Arvas, C.
Data Structures: Management and Storage
National Central Bureau of Statistics, Sweden, 1972

43. Bubenko, J.
Databehandlingsteknik
Studentlitteratur Sweden, 1968

44. Churchman, C.W.
The Design of Inquiring Systems
Basic Book, New York, 1971

45. Engles, R.W.
A Tutorial on Data-Base Organization
IBM, Poughkeepsie, New York, 1970

46. Firth, J.W.
The Idea of a Data Base
National Central Bureau of Statistics, Sweden, 1973

47. Hsiao, D. and Harary, F.
A Formal System for Information Retrieval from Files

48. Langefors, B.
Theoretical Analysis of Information Systems
Studentlitteratur Sweden and Auerbach, 1966

49. Langefors, B.
Introduktion till informationsbehandling
Natur och Kultur, Stockholm, 1968
50. Langefor, B.
Data Bases, Goals and Definitions

51. Lefkowitz, D.
File Structures for On-Line Systems
Spartan Books, 1969

52. Lundeborg, M. and Andersen, E.S.
Systemering - informationsanalys
Draft, to be published

53. Mitroff, I.I., Williams, J., and Rathsohl, E.
Dialectical Inquiring Systems: A New Methodology for Information Science
Journal of the American Society for Information Science, Nov-Dec 1972

Computer Simulation Techniques
Wiley, 1966

55. Nordbotten, S.
Automatic files in statistical systems

56. Prywes, N.S.
Structure and Organization of Very Large Data Bases

57. Samuelson, K. (ed)
Mechanized Information Storage, Retrieval, and Dissemination
North-Holland, Amsterdam, 1968

Data Structures and Accessing in Data-Base Systems
IBM Systems Journal No 1 1973
59. Stamper, R.K.
Logical Data Structures
Proceedings, BCS Conference on Data Management, 1970

60. Öberg, S.
Ett system för socio-demografisk statistik (SSDS),
grundläggande begrepp
National Central Bureau of Statistics, Sweden, 1972

61. Knuth, D.E.
The Art of Computer Programming
Addison-Wesley, Reading, Mass, 1968
D. Data base management systems

71. Bucknell, M.
A Plex System for Data Base Organisation
University of Stockholm and National Central Bureau
of Statistics, Sweden, 1973

72. Canning, R.G.
The Debate on Data Base Management
EDP Analyzer, Vol.10, No.3, 1972

73. CODASYL Systems Committee
A Survey of Generalized Data Base Management Systems
May 1969

74. CODASYL Systems Committee
Feature Analysis of Generalized Data Base Management Systems
ACM, May 1971

75. CODASYL Systems Committee
Report of the CODASYL Data Base Task Group
ACM, April 1971

76. Codd, E.F.
A Data Base Sublanguage Founded on the Relational Calculus
IBM Thomas J. Watson Research Center, New York, 1971

77. Codd, E.F.
Further Normalization of the Data Base Relational Model
IBM Thomas J. Watson Research Center, New York, 1971

78. Codd, E.F.
A Relational Model of Data for Large Shared Data Banks
Communications of the ACM, Vol.13, No.5, 1970

79. Durchholz, R. and Richter, G.
Das Datenmodell der „Feature Analysis of Generalized Data Base Management Systems"
Angewandte Informatik 12, 1972
80. Engles, R.W.
An Analysis of the April 1971 Data Base Task Group Report
A position paper presented to the CODASYL Programming Language Committee by the IBM representative to the Data Base Task Group, May 1971

81. Everest, G.C. and Sibley, E.H.
A Critique of the GUIDE-SHARE DBMS Requirements
Proceedings ACM-SIGFIDET

82. Fry, J.P.
Generalized Data Base Management Systems: An Introduction
University of Michigan, 1970

83. Joint GUIDE-SHARE Data Base Requirements Group
Data Base Management System Requirements
November 1970

84. MITRE Corporation
Data Management Systems Survey
January 1969

85. Tibell, D.
I vilken utsträckning tillgodoser förslaget från CODASYL Data Base Task Group önskemålet om data-oberoende
University of Stockholm, 1972

86. Rovner, P.D. and Feldman, J.A.
The LEAP Language and Data Structure
Proceedings IFIP 68, Edinburgh

87. Strnad, A.J.
The Relational Approach to the Management of Data Bases
Proceedings IFIP 71, Ljubljana
88. Sundgren, B.
Mängdlärans relationsbegrepp som hjälpmedel vid databasdesign
Statistisk Tidsskrift (Statistical Review), 1971:5
E. Particular data base functions

E.1 Schema functions

91. Aurbakken, E. 
Dokumentasjon av data i Statistisk Sentralbyrå 
National Central Bureau of Statistics, Norway, 1973

92. Canning, R.G. 
The "Data Administrator" Function 
EDF Analyzer, Vol.10, No.11, 1972

93. Deming, W.E. 
Statistical Adjustment of Data 
New York 1938, 1943, 1964

94. Felle, S. 
Imputering, amputering, eller... 
ADB-information M73:4, National Central Bureau of 
Statistics, Sweden, 1973

95. Jönrup, H. 
Korstabulering med utnyttjande av supplementär infor- 
mation samt vid partialt ofullständiga data 
National Central Bureau of Statistics, Sweden, 1972

96. Karlsson, K. and Nilsson, E. 
Variabelkatalogarbetet vid SCB 
National Central Bureau of Statistics, Sweden, 1973

97. Sandewall, E. 
Representing Natural Language Information in Predicate 
Calculus 
Meltzer & Michie (eds), Machine Intelligence, Vol.6, 
1971

98. Sandewall, E. 
Formal Methods in the Design of Question-Answering 
Systems 
Artificial Intelligence, vol.2, 1971
99. Simmons, R.F.
Natural Language Question-Answering Systems: 1969
Communications of the ACM

100. Svensson, E.-C. and Johannisson, J.
Fipoteket inom TIPS-projektet
Databehandling 6, 1970

101. Tärnlund, S.-Å.
Resolutionsprincipen ett logiskt system för konstruktion av en semantisk databas
University of Stockholm, 1973
E.2 Nucleus functions

E.2.1 Files

File Organization Schemes Based on Finite Geometries
Information Control 12, 1968

112. Arvas, C.
Some Aspects on the Analysis of File Systems
In "File 68 Working Papers", Studentlitteratur, Lund
Sweden, 1968

113. Lowe, T.C.
The Influence of Data Base Characteristics and Usage
on Direct Access File Organization

114. Lum, V.Y.
Multiattribute Retrieval with Combined Indices
Communications of the ACM, Vol.13, No.11, 1970

115. Lum, V.Y., Ling, H., and Senko, M.E.
Analysis of a Complex Data Management Access Method
by Simulation Modeling
AFIPS Proceedings, FJCC, Vol.37, 1970

116. Madnick, S.E. and Alsop, J.W.
A Modular Approach to File System Design
IAG Quarterly Journal, Vol.No.3

117. Martin, L.D.
A Model for File Structure Determination for Large
On-Line Data Files
In "File 68 Working papers", Studentlitteratur, Lund
Sweden, 1968

118. Mullin, J.K.
Retrieval-Update Speed Trade-Offs Using Combined
Indices
Communications of the ACM, Vol.14, No.12, 1971
119. Severance, D.G. and Merten, A.G.
Performance Evaluation of File Organizations through Modelling

120. Waters, S.J.
File Design Fallacies
Computer Journal, Vol. 15, No. 1

121. Wong, E. and Chiang, T.C.
Canonical Structure in Attribute Based File Organization
Communications of the ACM, Vol. 14, No. 9, 1971
E.2.2 Processes

131. Arvas, C. and Sundgren, B.
Parvisa processgrupperingar

132. Bubenko, B., Berild, S., et.al.
Analysis of Real Time Systems (ARTS), handbook and
program package
CADIS Project Group, University of Stockholm

133. Bubenko, J. and Ohlin, T.
Introduktion till operativsystem
Studentlitteratur Sweden, 1970

134. Colin, A.J.T.
Introduction to operating systems

135. Martin, J.
Design of Real-Time Computer Systems
Prentice-Hall, 1967

136. Nunamaker, J.F.
A Methodology for the Design and Optimization of
Information Processing Systems
AFIPS Proceedings, Spring Joint Computer Conference,
Vol.38, 1971

137. Nunamaker, J.F., Nylin, W.C., and Konsynski, B.
Processing Systems Optimization through Automatic De-
sign and Reorganization of Program Modules
Purdue University, 1972

138. Sundgren, B.
A Rewarding Rewording of the Process Grouping Problem
National Central Bureau of Statistics, Sweden, 1969

139. Vyssotsky, V.A., Corbató, F.J., and Graham, R.M.
Structure of the Multics Supervisor
Proceedings, Fall Joint Computer Conference, 1965
140. Watson, R.W.
Timesharing System Design Concepts
McGraw-Hill, 1970
Filter functions

151. Arvas, C., Granqvist, L., and Ohlsson, G.
Automatic editing in the yearly survey of manu-
factoring in Sweden, GPI
National Central Bureau of Statistics, Sweden, 1973

152. Brodda, B.
Folkontroll i söksystem
Forskningsgruppen för kvantitativ lingvistik, Stock-
holm, 1973

153. Conway, R.W., Maxwell, W.L., and Morgan, H.L.
On the Implementation of Security Measures in Informa-
tion Systems
Communications of the ACM, Vol.15, No.4, 1972

154. Fellegi, I.P.
On the Question of Statistical Confidentiality
Journal of the American Statistical Association,
Vol.67, No.337, 1972

155. Hoffman, L.J.
Computers and Privacy: A Survey
Computing Surveys, June 1969

156. Hoffman, L.J. and Miller, W.F.
Getting a Personal Dossier from a Statistical Data
Bank
Datamation, May 1970

157. Ivanov, K.
Quality-Control of Information
Royal Institute of Technology, Stockholm, 1972

158. Nordbotten, S.
Konfidsiell behandling av data, informasjonsnytte
og klassificering av data
Statistisk Tidsskrift (Statistical Review) 1968:5
159. Offentlighets- och sekretesslagstiftningskommittén (OSK)
Data och integritet
SOU 1972:47

160. Olsson, L.
Measures to protect privacy in a statistical database

161. Oswald, T. et al.
Strukturering av säkerhetsproblem vid ADB
Statskontoret, Sweden, 1973

162. Rodin, G.
Metoder för kryptering av datorlagrade data
Royal Institute of Technology, Stockholm, 1972

163. Sundgren, B.
Security and Privacy of Statistical Databases
Statistisk Tidskrift (Statistical Review) 1972:4

164. Zarkovich, S.S.
Quality of Statistical Data
UN-FAO, Rome, 1966
F. Specific design problems

171. Bell, J.A.
The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering
Communications of the ACM, February 1970

The Linear Quotient Hash Code
Communications of the ACM, November 1970

173. Buchholz, W.
File Organization and Addressing
IBM Systems Journal, June 1963

174. Ghosh, S.P. and Senko, M.E.
File Organization: On the Selection of Random Access Index Points for Sequential Files
Journal of the ACM, Vol.16, No.4, 1969

175. Hu, T.C.
A Comment on the Double- Chained Tree
Communications of the ACM, Vol.15, No.4, 1972

176. Johnson, L.R.
An Indirect Chaining Method for Addressing on Secondary Keys
Communications of the ACM, Vol.4, No.4, 1961

177. Kaimann, R.A.
Enter to the File, Randomize or Index I-II
Data Processing Magazine, Dec 1966 and Dec 1968

178. Lam, V.Y., Yuen, P.S.T., and Dodd, M.
Key-to-address transform techniques: A fundamental performance study on large existing formatted files
Communications of the ACM, Vol.14, No.4, 1971

179. de Maine, P.A.D. and Springer, G.K.
The COPAK Compressor
180. Morris, F.
Random Input Search Techniques: A Timely Review
Software Age, January 1970

181. Morris, R.
Scatter Storage Techniques
Communications of the ACM, Vol.11, No.1, 1968

182. Mullin, J.K.
An Improved Index Sequential Access Method Using
Hashed Overflow
Communications of the ACM, Vol.15, No.5, 1972

183. Nijssen, G.M.
File Organization, An Introduction
Amsterdam 1969

184. Nijssen, G.M.
Efficient Batch Up-Dating of a Random File
Proceedings ACM-SIGFIDET Workshop on "Data Description,
Access, and Control", 1971

185. Radke, C.E.
The Use of Quadratic Residue Research
Communications of the ACM, February 1970

186. Rouqué, E.
Bestämnning av blocklängd
National Central Bureau of Statistics, Sweden, 1971

187. Seppälä, Y.
On Optimization of the Maintenance of a Register
BIT, Vol.6, No.3, 1966

188. Stanfel, L.E.
Tree Structures for Optimal Searching
189. Sundgren, B.
Addressing i direktninnen med hjälp av divisionrestmetoden/Direct Addressing Using the Division-Remainder Method
Statistisk Tidskrift (Statistical Review), Sweden, 1969:3

190. Sundgren, B.
Direktminnesteknik: begreppsbildning och systemkonstruktion
National Central Bureau of Statistics, Sweden, 1971

191. Thorburn, H.W.
Några metoder för synonymhantering vid direktadressering
ADB-information 71:36, National Central Bureau of Statistics, Sweden, 1971

192. Waters, S.J.
Blocking sequentially processed magnetic files
Computer Journal, Vol.14, No.2

193. Weingarten, A.
The Eschenbach Drum Scheme
Communications of the ACM, July 1966

194. Williams, J.G.
Storage Utilization in a Memory Hierarchy When Storage Assignment is Performed by a Hashing Algorithm
G. Miscellaneous

201. Bachman, C.W.
Data Structure Diagrams

202. Bachman, C.W.
The Evolution of Storage Structures

203. Canning, R.G.
That Maintenance "Iceberg"
EDP Analyzer, Vol.10, No.10, 1972

204. Edström, O.
Man - Computer Decision Making
Gothenburg Studies in Business Administration, 1973

205. Felme, S.
Tabellsummor

206. Filotti, I.
Report on systems and languages for the establishment of files and compilation of statistical tables
Conf. Eur. Stats. /WG.9/86

207. TAB68 Project Group
TAB68 Handbook
National Central Bureau of Statistics, Sweden, 1973
I skrifter serien URVAL publicerar statistiska centralbyrån vetenskapliga artiklar och avhandlingar, metodbeskrivningar, referensböcker m.m i anslutning till den officiella statistikproduktionen i Sverige. Hittills har utkommit:

1 Rolf Vallerö: Svensk handels- och sjöfartsstatistik 1637—1813. En tillkomstshistorisk undersökning.


3 Gustav Sundbärg: Bevölkerungsstatistik Schwedens 1750—1900. With preface and vocabulary in English.


7 Bo Sundgren: An Infological Approach to Data Bases.

ISBN 91-38-01750-4