CORD an Object Model supporting Statistical Summary Information for Management Decision Making

PAUL ANTHONY GOLDER

Doctor of Philosophy

ASTON UNIVERSITY

October 1997

Aston University

CORD an Object Model supporting Statistical Summary Information for Management Decision Making

PAUL ANTHONY GOLDER

Doctor of Philosophy

1997

Information systems have developed to the stage that there is plenty of data available in most organisations but there are still major problems in turning that data into information for management decision making. This thesis argues that the link between decision support information and transaction processing data should be through a common object model which reflects the real world of the organisation and encompasses the artefacts of the information system. The CORD (Collections, Objects, Roles and Domains) model is developed which is richer in appropriate modelling abstractions than current Object Models. A flexible Object Prototyping tool based on a Semantic Data Storage Manager has been developed which enables a variety of models to be stored and experimented with. A statistical summary table model COST (Collections of Objects Statistical Table) has been developed within CORD and is shown to be adequate to meet the modelling needs of Decision Support and Executive Information Systems. The COST model is supported by an statistical table creator and editor COSTed which is also built on top of the Object Prototyper and uses the CORD model to manage its metadata.

Keywords

Object modelling; Statistical Summary Tables; Statistical Metadata; Management Information; Decision Support; Object Database Management.

Acknowledgements

My thanks to Vivienne for inspiration, encouragement, support and a lot of checking.

Copyright

CORD an Object Model supporting Statistical Summary Information for Management Decision Making

Paul Anthony Golder

List of Tables

Chapter 1

Information from Data

1.1. MOTIVATION FOR THE RESEARCH

The motivation for this research arose from a series of events and experiences which are useful to recount as they help to justify the research and to establish the problem which the research addresses.

A research student of mine, Kevin Lawson (1987), examined some of the requirements for intelligent statistical packages, and in particular focused his attention on the nature of variables and the appropriateness of particular statistical operations on them. It became apparent that, for this work to be useful in real world information systems as distinct from the structured world of statistical analysis, there was a bridge to be built which would enable the metadata implicit in an information system to inform any system attempting to carry out statistical analysis on the data held in the information system.

A consultancy project for Aston Business School concerned the development of an Executive Information System to facilitate access to summary information about the strategy and operations of the Business School. The project was only partially successful because, although it was relatively straightforward to build a vehicle for delivering executive information, it emerged that the most laborious task was the definition of the data and summaries to be presented. In many cases the data was already available in a computerised form but without the necessary metadata to enable summaries to be generated without considerable effort.

Another consultancy project for a small company encountered similar problems in that there were several computerised operational systems in place and more being introduced, but it was very difficult for the Managing Director to extract management information triggering the complaint "I have all the data but no information".

1.2. THE PROBLEM

The decision maker in an organisation has a very simple requirement: s/he needs access to the right information at the right time. Despite 40 years of computer support for business activities and great progress in the power of hardware and the quality of software, most decision makers are still frustrated by their inability to satisfy this need. This is all the more surprising when the abilities of modern computing systems to manage, to transmit, to locate and to reproduce vast quantities of business data are well known. Many decision makers will have access to a personal computer which may well be networked to the corporate data-server. Nevertheless it is not a simple task to obtain the right information at the right time.

The user attempting to access organisational data will find that the information system appears to "know" little or nothing about the organisation and the way it operates. The user has to repeatedly restate basic facts such as that *Accounts* is a *Department*, that *Departments* have *Managers* and that *Managers* have *Names*. Thus a simple SQL query could have the form:

```
SELECT manager.name FROM department, manager
WHERE department.name = "Accounts"
AND department.manager_id = manager.manager_id;
```

It is clear that current systems do not contain (or are unable to access) knowledge about the structure of the organisation which they support.

The distinction between data and information has been made by many authors and the problem can be seen as a simple issue, "Why when we have so much data easily accessible in most organisations is it so hard to find the appropriate information?". The existence of this problem, first discussed by Ackoff (1967), is well known and although there is an abundance of partial solutions on offer, no evidently satisfactory comprehensive approach has been reported.

This author will argue that the heart of the problem lies in the mapping of the detailed data which is generated by the activities of the organisation to the models

of the organisation which underpin the decision maker's approach to his/her tasks. It will be argued that this mapping is not well supported by current design methodologies or by current decision support systems and tools for creating them. Finally this thesis will propose a set of tools and methods for reducing the mismatch.

1.3. THE REFERENCE DISCIPLINES

The problem outlined above is situated within the domain of research conventionally referred to as Information Systems; however as Robert Blanning et al (1995) quote in their introduction to a recent special issue of Decision Support Systems, "The reference discipline of (information systems) is computer science, or experimental social psychology, or alas the International Case Clearing House". In the same issue, Stuart Madnick (1995) in his paper "Integration technology: The reinvention of the linkage between information systems and computer science" identifies what he sees as the directions in which information systems research should go. Resisting the temptation to digress into applications of information in management, such as marketing and strategy, he suggests as examples of key integration technology research issues: "data semantics acquisition", "data quality", and "evolving semantics", suggesting that in addressing these sorts of issues the discipline will be addressing key business problems. In particular he observes that an "organisation can be simultaneously 'data rich' and 'information poor' if they do not know how to identify, categorise, summarise and organise the data". The author supports this argument and observes that the issues explored in this research fall clearly into the domain of "integration technology" as identified by Madnick and thus have as reference disciplines *computer science* and *information systems*. In particular, the construction of databases, the design of information systems, the creation of decision support systems and the specification of statistical summaries are of concern to this work and material from all these areas is brought together in this thesis.

1.4. CURRENT APPROACHES TO THE PROBLEM

The problem outlined is well documented and many authors are engaged in research which addresses it. As always researchers bring their own specialism to bear on any problem and the different approaches derive from existing schools of research in computer science. Two main themes can be identified under which research into the wider area of this problem domain is currently being carried out.

The *engineering approach*, which focuses on the system design process, is concerned with successful elicitation of requirements and construction of robust systems to meet these requirements. The tools being developed by this approach are those of Computer Aided Systems Engineering (CASE), formal methods, and more recently Object Oriented Analysis and Design. These methods acquire and build models of the organisational structure, and these knowledge bases inform the design process and may be significant in the maintenance and evolution of the applications. Nevertheless the knowledge is embedded in the design is implicit rather than explicit, and is not generally available to the user. Thus this approach, whilst valuable for the construction of many applications, is not directly applicable to the construction of decision support systems where the user wishes to address and model ad-hoc problems.

Another approach is the application of *artificial intelligence*. Given an "unintelligent" information system the aim is to facilitate the user's access. An expert system front end can incorporate the knowledge of an expert user and can deduce user's intentions. Alternatively a natural language interface can achieve the same results. Both approaches require encoding of the organisational knowledge in an appropriate form. This knowledge is then accessed by an inference engine which draws conclusions about the request being submitted.

Both approaches suffer from the same shortcoming, which is that they both separate data structure from organisational knowledge. Thus they treat the fact that "customer X has an overdrawn credit account" in a completely different way to the fact that "a customer providing evidence of credit worthiness may be

granted credit account facility". Applications based on these approaches will inevitably mean that the user has to navigate two different systems. The seamlessness of this navigation from the user's point of view will depend on the quality of the user interface and will therefore be application dependent.

1.5. THE RESEARCH DOMAIN

From these experiences, and many similar examples encountered in teaching and consulting across the spectrum of statistical analysis, decision support systems, management information and information systems design, a solution domain emerged.

To tackle the problem it will be necessary to exploit and integrate several current research directions. Current approaches to organisational modelling will be assessed. OOA is an obvious candidate together with the work on data modelling inherent in Object Oriented database development. The gap between operational data and information for decision making is similar to the work on statistical databases and statistical data modelling, so this literature will be relevant as will current work in the design and construction of Decision Support Systems.

1.5.1 Conceptual modelling

First, it became clear that the problem area was not at the implementation level. That is, it was not concerned with how statistical packages operate nor how data is stored in databases, but how knowledge about these two fields is managed and exploited. To characterise current practice, a model is constructed when an information system is designed and built, and this is then lost or forgotten once the system becomes operational. But the operational system cannot tell the user much about the world it serves, and to understand the operational system, particularly why things happen the way they do, requires reference to external documentation, or usually discussion with members of the original project team. Thus when a user wishes to produce management information, summaries and aggregates; let alone more sophisticated forecasts, they have to provide a lot of extra information

to make this possible and in particular, knowledge about the types of variables and about their domains. This information is typically referred to as metadata within the statistical community, but is identical in nature to the conceptual models of the information systems community.

A major theme of this research is the construction of appropriate models to represent the problems being addressed by the development of an information system. The accepted terminology for high level abstract models which the developer uses to communicate with the problem owner is *conceptual models* and the process of deriving such a model is called *conceptual modelling*. The consideration of different approaches to conceptual modelling means the discussion of different frameworks for achieving a conceptual model. Logically one should refer to such as a *conceptual modelling framework, toolkit or method* however this is clumsy and we will follow convention and talk about a *conceptual model* when it is wished to refer to the method and not just the instantiation of a particular modelling approach.

1.5.2 Domains of discourse

It also became clear early in the research that there are several different areas to this study which many researchers have inadequately differentiated. A conceptual model is traditionally presented as a model of the real world or at least the part of the real world of interest to the organisation, commonly referred to as the Domain of Discourse or Universe of Discourse (UoD). However there are several other domains which need to be crossed in creating a path between the decision maker and the information they require. These are illustrated in figure 1.1. There is the information system itself, an analogue hopefully of the real world, but often we do not distinguish between them and will refer to *customer* when we mean *a customer's record*. The author was once told by an enquiry clerk at Paris Charles de Gaulle that the plane he had arrived on had not yet landed. It took some persuasion that landed it had, but the information system had not been appraised of the fact. Within the information system there are other domains all of which

contain representations of the real world: there are logical data models and flow charts and binary information stored on disc. These do not directly interest us but often confusion can arise when for example we are told that the department to which an employee is assigned is a character string with a maximum of 20 characters. Somewhere else in the world of interest there will be a manager with a problem. The term *problem domain* is used for any representation of the complex interactions, variables and objectives that make up the problem. There will also be a domain in which information is presented, structured in a manner pertinent to the problem domain. The language of this domain will be one of statistical summaries and management reporting: breakdowns, averages, totals etc.

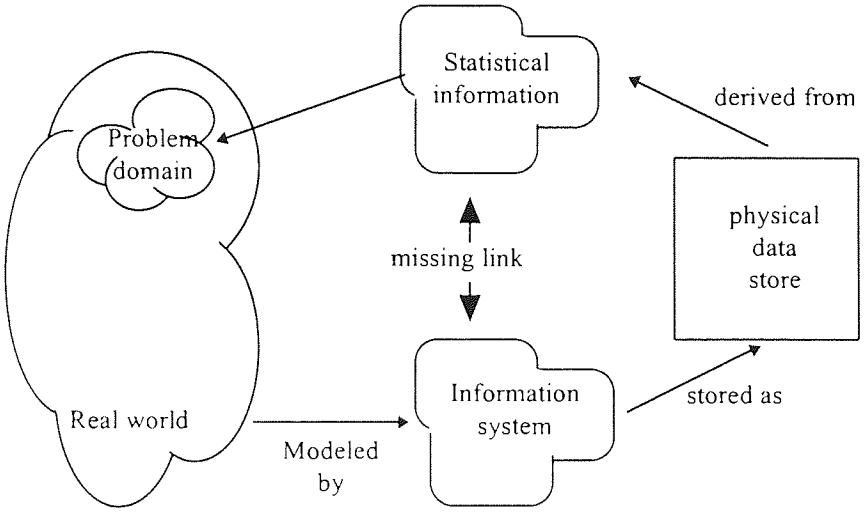

Figure 1. 1 : The various domains implicit in the research

1.5.3 Object orientation

It also became clear that any new approach to this area would have to draw on recent work in object oriented analysis and design. It was going to be necessary to create richer conceptual models which were capable of expressing not only the structure of the real world but also the content of the information systems and the structure of the statistical summaries describing them. It was expected that merely extending conventional modelling tools such as the E-R model would not be a satisfactory way of describing the different domains to be spanned. The richness of an object model was likely to be needed and the discipline of the object

oriented approach would be essential for controlling the complexity in the overall model.

1.6. WHAT IS AN INFORMATION SYSTEM?

Whilst there is a lot of discussion about the nature and construction of information systems, there is relatively little work which discusses the purpose, not of an individual system but of information systems in general. The work of Stafford Beer (1979) and the cybernetic fraternity has given us a useful way of looking at the role of information systems in organisations. Information is what flows in control loops which are central to the survival of any self regulating system. Although the cybernetic model has its critics it is useful for us to take on a key proposition: that is that the control system needs to contain a model of the system being controlled. It follows that if an organisation's information system has this control function it must contain or at least be in conversation with a model of the organisation. This is not very different from the idea of van Gigch and Le Moigne (1990) that the identity of an organisation is held in its memory, and that the information system is the organisational memory.

However, the development of information systems has never seemed to realise their promise, and although to some extent this may reflect over optimistic promotion of "business solutions", there is also reason to believe that there are many problems inherent in the development of such systems.

The literature on the failure of conventional approaches to the development of information systems starts with Ackoff's paper (1967) "Management Misinformation Systems". Ackoff argues that there are basic misconceptions about information systems that contribute to their lack of success.

Give them more is the view that managers need more information to enable them to make adequate decisions. The counter position is that in fact managers suffer from an information overload and that a minimum set of information should be provided to support the decision process in question. The tendency to provide too

much irrelevant information results from a lack of understanding of the problem and the decision process appropriate to its resolution. The growth of large corporate databases has made it easier to provide too much information, although the considerable improvement in quality of user interfaces to database systems may help the decision maker to be more selective in extracting relevant information.

The manager does not need to understand the information system. This view ignores the reality of the decision making process. It is usual for a decision maker when faced with information to want to know how it was assembled, and this is particularly true of condensed information which is far removed from the original elementary data. This need to understand the process reflects the need to establish confidence in the information prior to basing a decision on it. The user needs to understand the process at a conceptual level. It is not the intricacies of the database and its associated physical storage but the logical operations in deriving the figures presented which the user needs to understand.

The wider the view of the organisation the better. This position assumes that if managers have access to information about the rest of the organisation, they will be able to take better decisions. Ackoff argued that such information may easily be misinterpreted by someone not involved in its derivation, and may thus have deleterious consequences on the decisions being made. Ackoff was arguing for a very hierarchical approach to management which may in part have its origins in the management culture of the time and the technology of the late sixties. Whilst the danger of misusing information is a real one, the danger of taking decisions based on a myopic view of the current situation is also real. It is likely that current thinking would stress the communication of situation reports between parts of the organisation, so that any decision maker could inform him/herself of the wider context of their current problem. There is a considerable difference between encouraging someone to look at the notes and discussion documents of another and keeping colleagues informed about objective changes and approved policies.

Data makes decisions. This position suggests that information necessarily contributes to improved decision making. Ackoff argued that without understanding the decision process and identifying the appropriate information required by it, it is not automatic that merely providing more (irrelevant) information will aid the making of the decision.

Although written thirty years ago Ackoff's critique is still appropriate today. Madnick in 1995 still considers it relevant to repeat the familiar adage that data richness and information poverty remains with us and this despite the substantial growth in power of hardware and usability of software over the last 30 years. The matching of problem domain to information clearly remains a challenge.

1.7. THE THESIS

This author will argue that the heart of the problem lies in the mapping of the detailed data which is generated by the activities of the organisation to the models of the organisation which underpin the decision maker's approach to his/her tasks. It will be argued that this mapping is not well supported by current design methodologies or by current decision support systems and tools for creating them (see figure 1.1). This research makes several contributions to the solution of this problem:

- A review of existing conceptual modelling frameworks and, in particular, the modelling of behaviour, leads to the proposed CORD (Collections, Objects, Roles and Domains) Model.

- A conceptual model which cannot usefully be implemented or exploited is of limited value. The Object Prototyper has been developed which is a tool for managing object models and is the basis for both developing and testing the usability of the CORD model.

- A review of the literature on management reporting and statistical summary tables identifies the essential features of management reports. The literature

reflects a confusion between the logical and the presentation issues, so a conceptual model is developed, the COST (Collections of Objects Summary Tables) model, which separates these two levels of abstraction.

- The COST-table and COST-view models are both defined within the COST framework and a software tool is constructed, COSTed, which enables tables and views to be defined using a wimp interface, as well as the definition language of the COST model.

These improvements in conceptual modelling, the definition of management summaries, and the integration of a management summary into a standard object model, are all major contributions of this thesis.

1.8. TESTING OF THE THESIS

In developing and testing conceptual models it is not appropriate to consider proving something to be true or false ,or even to hope to discover something. The natural focus of the research in this area is that of generating descriptors of a view of a section of the real world, but in accepting that the information system of the organisation is also part of the real world, this language of description also needs to be able to describe the IS. The section of the real world under discussion is that of management information systems. This is much narrower than all computer systems, and the techniques explored and proposed are unlikely to be relevant to command and control systems, or network management systems, or a large number of other applications of computing. The justification for focusing on such a restricted area is its significant economic importance as represented by the number of people employed in it. The other justification is that the interface between technical specialist and non technical users and clients is much more significant in this area than in the specification of a highly technical application.

Every specialist area has its own language for describing it, and the fact that some of these languages use very familiar terms does not make them less technical than a language structured entirely in a mathematical symbolism. This research aims

to develop a language and to test this language. How can it be tested and what would constitute a validation of the work? In the review many such languages are encountered, each developed to solve a perceived problem with the (then) current state of conceptual modelling. There is an evident tension between semantically rich descriptions with a large number of different constructs which are not always easy to match to the real world, and semantically poor languages with a few constructs which mean that the modeller needs a lot of supplementary information to help understand the model.

Although it seems to have received little discussion, the implication is that there is an ideal modelling language which is not too poor to be able to represent the richness of a real world domain nor too rich to make the modelling process impossibly complex (figure 1.2). The author suspects that there is not such a language. The suitability depends not only on the richness and structure of the language but on the complexity of the real world domain, and the purpose of the modelling, and the experience of the players in the modelling process - the modeller and the user / client - will also moderate the success of a particular approach to conceptual modelling.

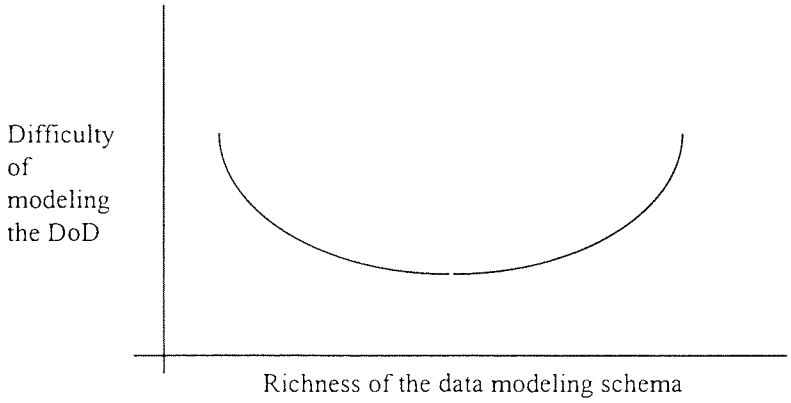

Figure 1. 2 : Matching model to problem domain

It would be a very poor modelling method which could not be shown to be superior to all others for a particular problem description. Unfortunately, because of the high dimensionality of the real world, the success of problem domain modelling languages is not transitive and it does not follow that because A has

been shown to be superior to B and B superior to C that A is any advance on C.

How then is it proposed to test the models developed in this thesis? The main requirement is *consistency*: that is, the language should be applicable in a consistent way without arbitrary treatment of particular situations. The language should be *robust*: that is, it should apply to situations for which it was not specifically intended. Thirdly the language should be *parsimonious*: that is, it should contain the minimum of constructs necessary to describe the domains for which it is intended. In this particular case robustness is of the essence, as it is intended to develop a modelling language which will not only represent in a satisfactory way the section of the real world being modelled, but also permit the description of elements of the information system appropriate for summarising and reporting on the real world. These are two distinct domains (although many authors do not emphasise the distinction) and it is the bridging of the gap between these domains which is the object of this research.

- The testing of the thesis by the definition of a suitable conceptual modelling framework which has been tested and implemented in an Object Oriented application layered on a relational database management system gives credence to the validity and potential applicability of the CORD and COST conceptual models. This too is a substantive contribution of this thesis.

1.9. STRUCTURE OF THE THESIS.

The thesis falls naturally into two main parts:

In *Section A*, the current state of conceptual modelling is reviewed. As the field of conceptual modelling is very large, it has been found necessary to partition the discussion into several distinct areas. From this review a data model is proposed and tested which synthesises current thinking. This is the CORD model.

Most modelling of the kind described in *Section A* is concerned with the representation of the problem domain, and will include reference to real world

objects. However, if the manipulation of management reports and summaries is to be included within the DoD so that the data management system can reference them, this will involve an extension of the DoD to include Management Information System constructs.

In *Section B,* a survey of the main ideas underlying Information Systems in organisations is carried out, and the particular case of statistical summary tables is reviewed at length. This results in the definition of a Model for statistical summary tables (COST) which is defined in CORD and tested by implementing it in the Object Prototyper.

Finally, in the last section, chapter fifteen, the thesis is reviewed and its contribution to knowledge assessed.

Section A

Conceptual Modelling and the CORD model

A.1. INTRODUCTION

This section introduces the first main part of the thesis. This part is concerned with the way that the problem domain is modelled and the extent to which the richness inherent in any real world DoD can be reflected in formal conceptual models. The review covers more than the usual areas of concern to conceptual modellers and includes a particular examination of the nature of elementary data and the structure of domains which are normally associated with attribute values. As a result of the review a new conceptual model is proposed, justified, and tested

A.2. CONCEPTUAL MODELS

As stated in the introduction (section 1.5.1), this research is mainly concerned with conceptual models. The main stages in developing a database application in the Avison (1992) methodology are *Business analysis*; *Conceptual Modelling*; *Logical Modelling* and *Physical design*. In the business analysis phase the problem, as perceived by the user, is identified and the business solution is explored: this constitutes the problem domain. In the conceptual modelling phase an abstract model of the problem is created which defines the domain of discourse. This has no direct relationship with any computerised solution or any particular database technology. In the logical stage, the conceptual model is mapped into a database logical model, which is then converted to specific database objects in the physical design stage. The most complex step in this process is the construction of an appropriate conceptual model, because this is the move from a very informal and ill defined business analysis to the formality of a modelling framework. The usual implication that this is a structured capture of information is challenged by Beynon-Davies (1996), who describes it as more of a negotiation between interest groups. Thereafter the conversion from conceptual to logical, and from logical to physical, are important steps with serious implications for the performance of the system, but are relatively structured processes.

Conceptual modelling is:

> "the process of identifying appropriate concepts and linking them with real world phenomena." (Beynon-Davies 1996)

There are many conceptual models in current use and even more described in the research literature. All conceptual models use a number of abstractions to place a structure on the very large number of concepts in the real world which the typical problem domain encompasses.

Most conceptual modelling frameworks have an associated formalism which usually includes a specific notation. In terms of the validity of a conceptual modelling approach, the notation it adopts is not relevant, and in general in this thesis there will be no further discussion about notation unless it contributes to our understanding of the concept being explored.

A.3. OVERVIEW OF SECTION A

In this first section of the report the current state of conceptual modelling is reviewed. As the field of conceptual modelling is very large, it has been found necessary to partition the discussion into three distinct areas. Chapter two examines the current thinking on the structure of data. This area has not been the focus of much research effort in recent years, having been thoroughly studied in the 60s and 70s when authors were trying to formalise data processing systems. However, because the reporting and summarising of information is dependent on the basic *nature of the data* being summarised, it is necessary to ensure that the latest thinking in this area has been incorporated into the review. In chapter three the more conventional aspects of conceptual modelling, *objects and classes* are reviewed. Here there have been many recent developments which, following on from the classic Entity Relationship model of Chen (1976), have seen the emergence of object oriented approaches to the analysis of systems. The main contributions of different OO approaches are examined. Next, chapter four looks at the current state of research in the *modelling of the dynamics of data*. Unlike the previous two chapters which reflect in general a convergence of thinking

dominated by several well used practices, dynamic modelling reflects less consistency of approach.

These review chapters are followed by the first chapter reflecting a substantial integrating contribution of this thesis. The concepts of *Entities and Roles* are introduced in chapter five which, building on the work of Wieringa (1991) and having aspects of many other approaches, tries to formalise the dynamic aspects of conceptual modelling. Chapter six returns to one of the major legs of the Entity Relationship model, and examines what is left of *relationships* after the richer object modelling constructs of inheritance and the encapsulation of attributes have been isolated. Here the discussion returns to the nature of elementary data and the role of domains, and hence brings up-to-date our understanding of the nature of data as reviewed in chapter two.

In chapter seven, a conceptual model, the *CORD* (Collections, Objects, Roles and Domains) model, is proposed based on ideas developed in the previous five chapters. This conceptual model is tested in chapter eight by building a software tool, the *Object Prototyper*, to hold the model. The Object Prototyper is intended as a high level tool for holding and manipulating conceptual models, and this is used to manage the CORD model.

A data model is a set of constructs which can be mapped to elements of the domain of discourse, and provide an abstract representation of the DoD which is the basis for discussion, reasoning and subsequent system design. It is not difficult to propose a plausible data model. However this research was not driven by simply the desire to model aspects of the real world not adequately covered by existing modelling frameworks, but by the wish to move seamlessly from a conceptual representation of the problem to the production of management reports in the form of statistical summary tables. It was thus necessary to demonstrate that the model is consistent enough and practical enough to support such value added applications. The approach to validating the model was to create a meta-object Prototyper which would manage data models, and within this general

structure to define the particular model, and show that realistic examples could be entered and retrieved. Although the model developed could be represented with a variety of graphical notations, the need to define it and to be able to enter and store elements of a conceptual model dictated that the main representation would be a textual programming-language-like definition.

The second part of the thesis builds on the CORD object model to examine how management summary information, in particular statistical summaries can be modelled and managed in an object environment. This second part is introduced in Section B where a review of the contribution of the first part of the thesis also appears.

Chapter 2

The Nature of Data

2.1. INTRODUCTION

To achieve any comprehensive view of the relationship between decision support systems and the underlying data which they manipulate, there is a need to explore the nature of this data and attempt to identify the range of types and structures which may present themselves. This thesis will focus on traditional text and numeric data which is still the bulk of information manipulated in Management Information Systems. Richer data such as images are clearly relevant to some decision making in the commercial field, and may well be significant in other areas but the analysis will not be extended to this wider field.

2.2. THE STRUCTURE OF DATA

Data is so familiar and has existed for so long that it is too easy to take it for granted. The major texts and much of the research literature focus on relatively high level constructs such as records and forms which will be discussed later. McLeod (1976) discussed domain definition exclusively in the context of semantic integrity maintenance in a relational database and while the problem of domain validation is addressed in detail he does not add to the understanding of domains per se. Curtice and Jones in 1981 thought it worth restating the "fundamentals of data element definition" of data and Kent (1986), whilst concentrating his discussion on entities and records, does also discuss "the fundamental semantics of information representation in records". A synthesis of these two contributions will be presented as a formulation of the conventional data processing view of data.

2.3. DATA PROCESSING APPROACH

Observing that a simple data value, whether numeric "2345" or textual "MIX", cannot, without the specification of a context "telephone number", "age of a monument" or "date in roman numerals", be interpreted, one is led to assert that a

data item is a pair: a value and a definition of what that value means. The definition of the data item contains several types of information. Curtice and Jones (1981) distinguish three aspects of this definition. The *physical* group concerns the external representation of the value on the communication or storage medium. There have to be rules which enable the symbol itself to be recognised, issues of punctuation layout etc. Such rules would enable us to recognise (in appropriate contexts) that 1/2 0.5 ½ 2^{-1} consists of four distinct interpretable groups of symbols not a single symbol string. The *representational* group informs us as to how the symbols are to be interpreted, i.e. what they stand for. Thus 1/2 could mean between 1 and 2, 2^{-1} could be some type of footnote and ½ could refer to the first out of two items. The *contextual* group tells us what the presence of the symbol at this place at this time stands for. Thus ½ could be the annual percentage rate of increase of inflation or the age of a child in years. This corresponds to Kent's (1986) three level model whereby the definition of an elementary data item includes the specification of *what* each field contains (e.g. money, date), *how* the contents of the field refer to the entity (that is the representation used), and *why* the field is present in the record (what information is being conveyed by its presence).

In practical information systems data is commonly stored or located in fields, positions on a disk or spaces in a form. This corresponds to the physical level of data abstraction. The semantics of the field will be discussed below.

It is frequently useful to define the domain from which a data value is drawn. The domain is a concept drawn from mathematics, which defines the range of possible values that a particular data item can take. As such it corresponds to the representational part of the total definition. In practice it is hard to define a data value without implicitly or explicitly describing its extension. The extension or the set of things to which it can apply is its domain.

Having defined the domain it is possible to define what Curtice and Jones call a data element. This is a conceptual level definition which is independent of the

representation or coding used or the physical way in which the code is stored and which defines the meaning to be attached to this code here and now. This corresponds with Kent's Why? which uses the context of the field in a record to communicate the immediate meaning of the data element.

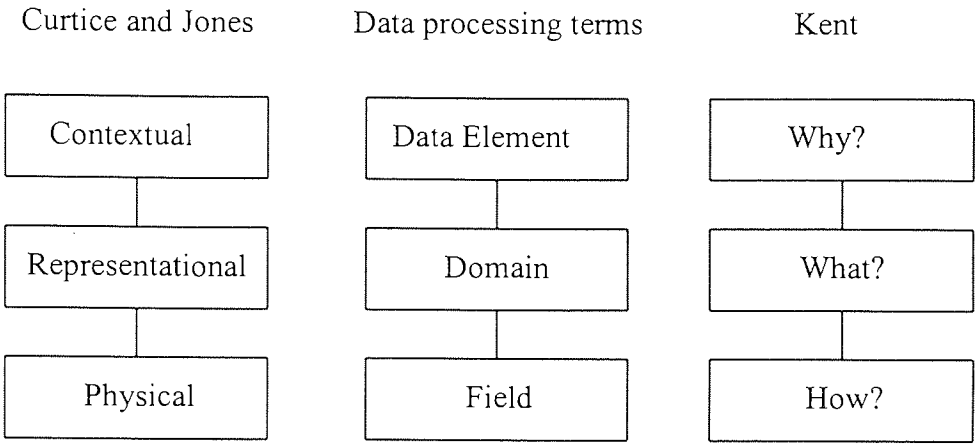

Figure 2. 1 : Levels of data abstraction

The authors of the two papers spent some time discussing the issues which arise when the domain for a particular data element consists of a set of identifiers e.g. employee_id. Although this can be treated in the above model, it is more useful to explore this in the context of the object model of data which will be presented later.

2.4. SEMANTIC APPROACH

An alternative approach is to ask what a simple data element *means*. A data element is a statement which can be presented in a natural language form and can thus be examined for meaning like any other statement. The formal study of meaning of statements is the field of semantics. It is thus relevant to ask what semantics can tell us about data elements and their definitions. The first stage is to outline the relevant semantic concepts and then to apply them to data element definitions.

2.4.1 Semantic Concepts

The basic concepts introduced by (and quoted more or less directly from) Hurford

and Heasley (1983) are :

2.4.1.1 Utterances, sentences, propositions

An *Utterance* is any stretch of talk, by one person, before and after which there is a silence on the part of that person. An utterance is the use by a particular speaker, on a particular occasion, of a piece of language, such as a sequence of sentences, or a single phrase, or even a single word. A *Sentence* is neither a physical event nor a physical object. It is, conceived abstractly, a string of words put together by the grammatical rules of language. A sentence can be thought of as the ideal string of words behind various realisations in utterances and inscriptions. A *Sentence* is a grammatically complete string of words expressing a complete thought. A *Proposition* is that part of the meaning of an utterance of a declarative sentence which describes some state of affairs.

2.4.1.2 Reference

By means of *reference,* a speaker indicates which things in the world (including persons) are being talked about.

2.4.1.3 Sense

The *Sense* of an expression is its indispensable hard core of meaning. An *Analytic* sentence is one that is necessarily true as a result of the senses of the words in it. A *Synthetic* sentence is one which is not analytic, but may be either true or false depending on the way the world is. A *Contradiction* is a sentence that is necessarily false.

2.4.1.4 Types of expression

A *Referring Expression* is any expression used in an utterance to refer to something or someone (or a clearly delimited collection of things or people), i.e. used with a particular referent in mind. An *Equative Sentence* is one which is used to assert the identity of the referents of two referring expressions, i.e. to assert that two referring expressions have the same referent. A *Generic Sentence*

is a sentence in which some statement is made about a whole unrestricted class of individuals, as opposed to any particular individual.

2.4.1.5 Predicators and predicates

The *Predicator* of a simple declarative sentence is the word (sometimes the group of words) which does not belong to any of the referring expressions and which, of the remainder, makes the most specific contribution to the meaning of the sentence. A *Predicate* is any word (or sequence of words) which (in a given single sense) can function as the predicator of a sentence. A Predicate usually links two arguments. For example, 'John works in Finance'. *Works in* is a two place predicator linking John and Finance. Predicates can also have one or three places. Examples are 'John is tall' and 'John gives advice to clients'. *Is tall* is a one place predicate qualifying John whilst *gives* is a three place predicate linking John, clients and advice.

2.4.1.6 Context of references

The *Universe Of Discourse* for any utterance is defined as the particular world, real or imaginary (or part real, part imaginary) that the speaker assumes he is talking about at the time. The *Context* of an utterance is the small subpart of the universe of discourse shared by the speaker and hearer, and includes facts about the topic of the conversation in which the utterance occurs, and also facts about the situation in which the conversation itself takes place. The *Definiteness* of a noun phrase is the assumption by the speaker that the hearer will be able to identify the referent of the noun phrase, usually because it is the only thing of its kind in the context of the utterance, or because it is unique in the universe of discourse. A *Deictic* word is one which takes some element of its meaning from the situation (i.e. the speaker, the addressee, the time and the place) of the utterance in which it is used.

2.4.1.7 Stereotypes and extensions

The *Extension* of a one place predicate is the set of all individuals to which that

predicate can truthfully be applied. It is the set of things which can potentially be referred to by using an expression whose main element is that predicate. A *Prototype* of a predicate is an object which is held to be very typical of the kind of object which can be referred to by an expression containing the predicate.

The *Stereotype* of a predicate is a list of the typical characteristics of things to which that predicate may be applied. A tightly specified stereotype would contain the set of necessary and sufficient characteristics to identify an instance of the predicate.

	Thing (or set of things) specified	Abstract specification
Pertaining to all examples	EXTENSION	SENSE
Pertaining to typical examples	PROTOTYPE	STEREOTYPE

Table 2. 1 : Relationships between semantic concepts.
(from Hurford and Heasley 1983)

2.4.2 A semantic analysis of elementary data items

Having laid the foundations, what does semantic theory have to tell us about a typical data element?

Name	Position	Department	Age
J Jones	Manager	Accounts	46

The manager is "J Jones"
The manager is 46 years old
They are 46 years old.

Figure 2. 2: A record and three sentences derived from it

The record in figure 2.2 constitutes an utterance which could have been presented in many forms and languages. The three sentences are (representations of) propositions which may be implied by the utterance. All three are simple equative synthetic sentences, however they differ in several important ways. In the first

sentence *is* is the predicate and *J Jones* a referent whereas in the second and third *is 46 years old* is the predicate.

The third is obviously a deictic sentence, the sense of which can only be identified by knowing who *they* are. In fact on reflection, it is clear that all the above statements require some knowledge of the immediate context of discourse to enable them to be understood. It may be implicit which company is being discussed but it is not explicit, moreover, age is meaningless without a date, and a units of measurement for age which was not explicitly given has had to be added.

The extensions of some of the predicates are not clear because of the rather overloaded use of the word *is* to link referents. The statements could be reformulated so that the extensions of the predicates are more clear.

> The age of J Jones is ...
> The name of the manager of the accounts department is ...

In these formulations the extensions are clearer (the set of ages of people, the set of names for accounts managers).

2.4.3 Conclusions from the semantic analysis

The semantic analysis has focused attention on two main issues: the significance of the context of discourse in establishing the meaning of a data element, and the existence of two distinct types of predicate within data element definitions (the one place predicate *is 46 years old* and the two place predicates *is named*, *has position*, *is in department*).

2.5. OTHER CONTRIBUTIONS TO DATA DEFINITION

The focus of definition of data in physics and other natural sciences is on the concept of dimension and the units of measurement within that dimension. Thus the height of an object is a measurement of type distance, the surface area is distance^2 and volume is distance^3. Generally what is being measured is of more

significance than how it is being measured. This represents a two level model with the dimension being represented as the conceptual level and the units of measurement as a representation level.

Other work has focused on units of measurement, identifying different classes or levels of measurement with real scales, discrete and ordinal scales of measurement (Lawson 1987).

The contribution of natural sciences to our understanding of data definition is to create a distinction between the domain from which the representation is drawn and the conceptual level of the measurement being represented.

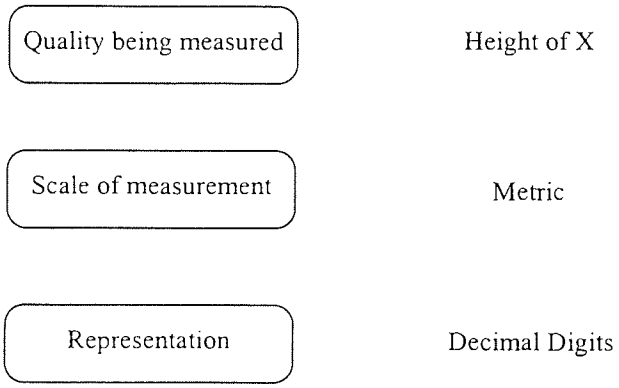

Figure 2. 3 : Dimensions and scales of measurement

2.6. CURRENT PARADIGMS

Having examined basic data elements consideration must be given to basic collections of information. It would be surprising to find simple data elements such as "he is 46" occurring except in the form of spoken utterances because there is so much of the context of discourse which would need to be defined in most other circumstances. Over the centuries during which data has been manipulated in written form, some basic structures have become commonplace and influence the way we expect to see data presented and communicated. There are three main views of data in every day use, these are: the form, the table and the report.

2.6.1 The form

The simple form or fiche is very widespread. The form has a title and series of

prompts associated with spaces to enter (or display) variable data. The meaning of an elementary data item is defined by its position on the form, and prompts adjacent to each position help identify its meaning as do appended notes, footnotes, rubrics etc.

The semantics are simple, and the title (sometimes implicitly rather than explicitly) defines the context of discourse and the extension i.e. the class of objects to which the form can be appropriately applied. Usually each form applies to a single referent. The fields within the form constitute either one or two place predicates relating to that referent. Notes on or with the form will often relate to the representation of an attribute to be used (e.g. weight in kgs.).

2.6.2 The table

The classic table presents itself as a heading, and a set of named columns and rows of instance data. That this is a primitive concept is illustrated by its use in a wide range of popular non technical contexts without detailed instructions as to how to interpret the information presented. The semantics of the table are relatively simple. Like the form the overall heading defines the class of objects that may appear as instances. Each row represents one such instance. Each column identifies an attribute of the instance and each attribute is represented in a consistent way over the instances.

2.6.3 The report

The report is structured like a series of nested tables with totals and sub totals. A greater attention is given to typographical and design issues to improve presentation and to facilitate interpretation. The semantics of the report are complex. Thus the meaning of a given data element will depend on its position on the page and its position in a logical hierarchy which is mapped to the page. Unlike tables, different types of object may be represented in one report and the extension of any row (which may take up several lines but is usually restricted to one line) depends on its position in an object hierarchy. It will be necessary to

return to the discussion of reports later in the thesis in the context of summary data but it is sufficient to say here that the complexity experienced by any user of a report generator is related in part to the multiple classes of object to which a single report can refer.

representation	number of objects referenced	number of classes of objects referenced
form	one	one
table	many	one
report	many	many

Table 2. 2 : Common data representations

2.7. DATA PROCESSING CONCEPTS

With the development of data processing systems the traditional ways of storing and communicating data elements were augmented by electronic encoding and magnetic storage. After one hundred years the concepts introduced by Hollerith and the punched card have enlarged the traditional paradigms which now include fields, records and files.

2.7.1 The field

The field is a logical unit usually represented by many symbols, whether holes in cards, magnetic charges or printed characters. The field is the smallest item of data representing a single logical element. The extension of a field definition or the meaning of a value in a field can only be known with reference to the system documentation because the uniform basic representation (binary coding) is completely indecipherable without such documentation.

2.7.2 The record

The record is a collection of fields and is thus an abstraction of the form whereby all predicates are specified implicitly by position or sequence and the context of

discourse is made explicit in a completely separate document.

2.7.3 The file

The file is a collection of records, usually of records of the same type, and is the basic form of secondary storage of such records. Files may be organised in a variety of ways for operational reasons such as to speed access, however such organisation does not have any semantic content as far as the user of the file is concerned.

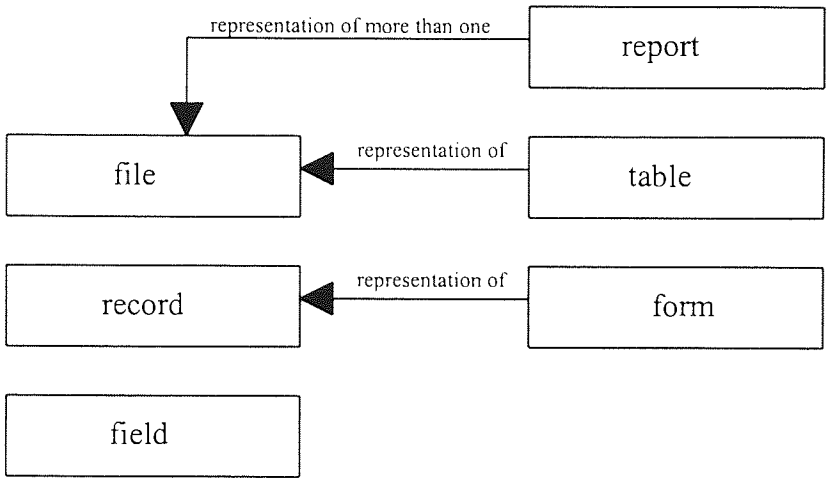

Figure 2. 4: Approximate relationships between data forms

2.8. NEWER DATA MODELS

The development of data modelling constructs and high level programming languages have also contributed to the understanding of the nature of data.

2.8.1 Entity relationship modelling

A major contribution to the modelling of data, or rather the construction of data models mapping to the real world, has been the Entity-Relationship (E-R) model of Peter Chen (1976). Although this is just one of many subsequent data models, the main emphasis since has been on enriching the structure of complex objects and relationships which has little relevance to this current discussion of

elementary data concepts.

The information about an entity is obtained by observation or measurement and is expressed by a set of attribute-value pairs (Chen 1976). Chen defines the attribute of an entity instance as a function which maps from an entity set to a value set or the Cartesian product of value sets.

In E-R modelling entities are identified which are real world objects of interest and associated with an entity are attributes which are characteristics of interest of those entities. Thus in the earlier example J Jones might be considered an entity and age, position and name attributes of that entity. Chen and many later authors leave the diagramming tools to represent classes of objects and any relationships between them, treating attributes at a lower level of detail by declaring value-sets first and attribute value-set pairs for each entity being modelled. Some authors show attributes within entity symbols on diagrams whilst others show them attached to entities as in figure 2.5.

It is usual for E-R models to represent relationships between classes and to refer to attributes (figure 2.5) rather than particular instances and their values (figure 2.6). The semantics are simple for attributes: a member of an entity class can have certain attributes and an instance of that class has a value for that attribute drawn from the domain of possible values. It is also possible to specify extra constraints associated with a particular attribute pair. It may be possible (in a particular DoD) for an instance not to have a value for an attribute. It may be legitimately absent, not applicable or simply not known.

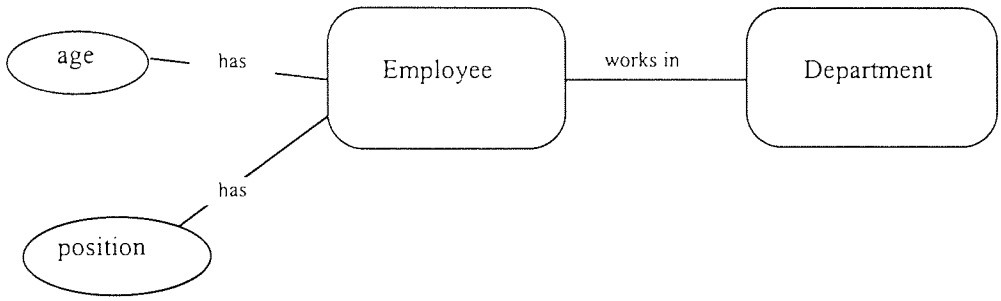

Figure 2. 5 : A conventional E-R model

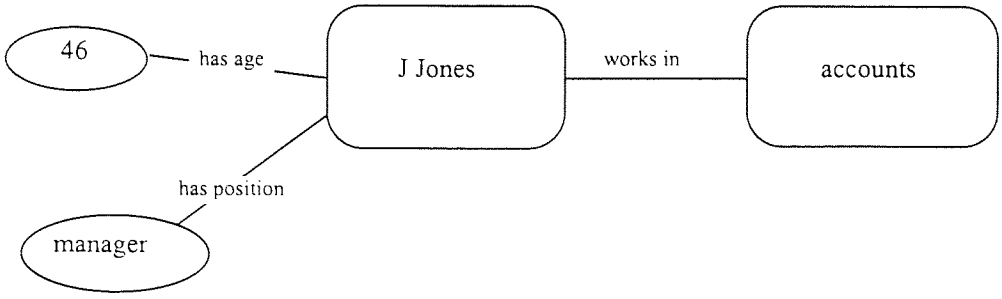

Figure 2. 6: An instantiation of Figure 2.5

2.8.2 Object Model

In the desire to build implementation independent data models, analysts have borrowed from Object Oriented programming and have used the Object Oriented approach for the analysis phase of a project. This use of Object Oriented Analysis (OOA) brings together some of the diverse strands in data element definition discussed above and develops a data model which is implementation independent.

Object Oriented Analysis has been developed to enable the user to construct richer data models during the analysis phase than could be supported by the popular E-R model. OOA should thus address issues which are relevant to this research. Taking Coad and Yourdon (1991) as a typical example of a practitioner oriented method it is useful to examine their approaches to the identification and definition of data. The method is relatively rich in modelling constructs based on the Object and the class of Object. The authors distinguish between two concepts, the attribute and the instance connection. Attributes of a person would be such typical qualifiers as Name and Age. An instance connection on the other hand, establishes a relationship between two objects such as John (a person) works for Finance (a department) where Person and Department are both classes of objects within the DoD. The detail of the method reflects traditional E-R modelling. A many-many instance connection is examined for hidden objects and transformed into two one to many instance connections. Moreover repeated attributes are examined for potential hidden objects, thus a vehicle owner may have the vehicle identification number of his/her vehicle as an attribute but when allowance is made for more than one vehicle a repeated group of attributes is created which

leads to the identification of vehicle as a new Class&Object. The recommendations on identifying attributes, whilst firmly based in the modelling of the system not the real world, do attempt to keep the level of abstraction relatively high with advice not to normalise (an implementation issue), to identify atomic concepts such as address not the set {state, town street, number}. The definition of attribute given is "An Attribute is some data (state information) for which each Object in a Class has its own value." Instance connections are said to model association. "An Instance Connection is a model of problem domain mapping (s) that one Object needs with other Objects, in order to fulfil its responsibilities" (Coad 1991)

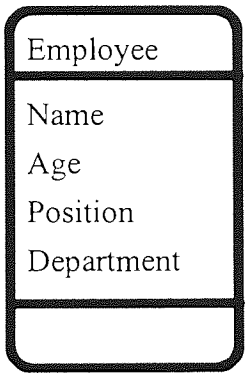

Figure 2. 7 : Representing Attributes in OOA

2.8.3 OMT

Rumbaugh et al (1991) introduce an Object Oriented methodology called Object Modelling Technique which it is claimed supports the whole of the system development lifecycle. OMT will be referred to when relevant elsewhere in this thesis however here the concern is with the OMT approach to defining basic data.

Rumbaugh defines an attribute as "a data value held by the objects in a class", "Name age and weight are attributes of person objects". To eliminate the confusion which exists between attributes and object references the reader is told that "An attribute should be a pure data value, not an object. Unlike objects, pure data values do not have identity." Advice is also given not to "bury pointers or other object references inside objects as attributes. Instead model these as

associations. This is clearer and captures the true intent rather than an implementation approach." (Rumbaugh 1991)

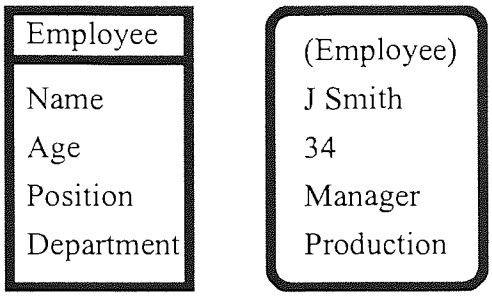

Figure 2. 8 : Attributes and an instance in OMT

2.9. CONCLUSIONS

Without presuming on later discussion as to what is an entity or object, a common thread from research to date into the basic nature of items of data is discernible. Because of the ubiquity of data and the long history of data management most authors have accepted and adopted a rather pragmatic approach which confounds representation and implementation issues and gives very little attention to conceptual matters.

An attribute value is the basic item of data and can be viewed in terms of the semantic analysis as a sentence with a one or two place predicate or from the data analysis point of view as having Contextual, Representational and Physical levels of analysis. From the current literature on data modelling an attribute will be considered to be a named mapping between instances of a class of objects and a domain. The context of an attribute is the class of objects to which it applies and the domain of discourse within which the model is being constructed.

For any given class, each mapping will have a unique name which, given the context, should enable the user to identify the Sense of the mapping. An attribute has not only a defined domain but may have other attributes. Its cardinality determines the maximum and minimum number of values from the domain that can be mapped for any one instance of the class. Typical cardinalities are 0:1 for an attribute which can take at most a single value and is optional and 2:4 for an

attribute which must have two values but may have up to four.

It is left to examine in later chapters the relationships between higher level constructs in a data model, entities, relationships etc. and to examine the nature of the domains to which attributes may be mapped.

Chapter 3

Objects and Classes

3.1. INTRODUCTION

In the last chapter the nature of a single observation was examined and a model for the lowest level of a set of data, the datum point, was developed. In this chapter the nature and definition of objects about which data is collected will be developed.

An essential stage in any data modelling process is to identify the Domain of Discourse (DoD) which is referred to by some authors as the Universe of Discourse. Jardine (1987) asserts that the ontological basis of a data model is the perceived Domain of Discourse which consists of a set of entities which are concrete or abstract objects. Although it is usual to discuss a Domain of Discourse which applies to the real world or problem space there are in fact several different domains with distinct constructs and different degrees of mapping from one to the other. Some of these different DoDs are to be found in the components of the information system itself. (Figure 3.1)

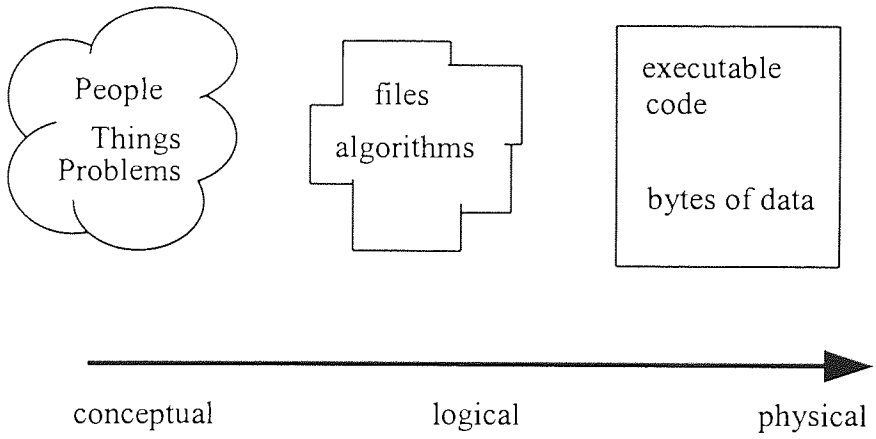

Figure 3. 1 : Different levels of abstraction

The literature on data modelling is rich in discussion of the appropriate elements to use in constructing a conceptual model of a particular Domain of Discourse. This chapter reviews some of the choices and identifies appropriate constructs for the purpose of this research.

3.1.1 Hierarchy of Domains of Discourse

It is sometimes implied that there is during the elaboration of a project one clear and distinct Domain of Discourse. Even when the problem domain is well defined, there are a series of nested domains, the boundaries of which are sometimes rather fuzzy. The different domains can be seen as a succession of descriptions. In discussing the organisation it is possible to ignore its information system (which may not exist!), but not the environment in which the organisation is situated. In discussion of the computerised information system the scope should include the IS, and hence the organisation and its environment. It is because of these multiple nested domains (Figure 3.2) reflecting different but valid views of the problem space that the phrase *Domain of Discourse* is preferred to that of *Universe of Discourse*.

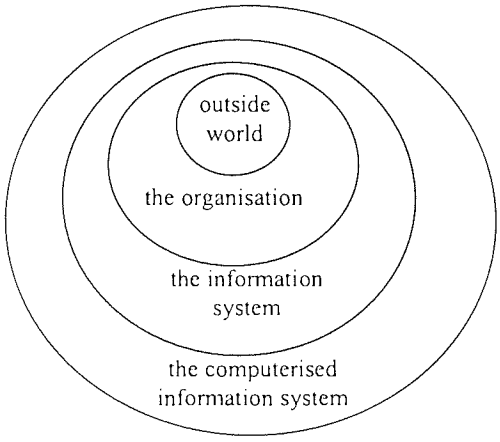

Figure 3. 2 : Nested domains of discourse

3.2. CONVENTIONAL DATA MODELS

Whilst there has been a long history of the development of models of data, and a variety of graphical models have been used to represent files and processes since the early days of computing, the evolution of conceptual data modelling as an attempt to model the real world, as a stage to implementation rather than as a means of representing a particular implementation, is relatively new.

In this respect the original data models are the relational data model (Codd 1970)

and the Entity Relationship data model of Chen (1976), both of which introduced the idea that a complex domain of discourse could be modelled with very few constructs.

3.2.1 Relational Model

The Relational Model (Codd 1970) is usually accepted as a logical model in terms of figure 3.1 and is neither a model or prescription for the physical organisation of a set of data nor a suitable conceptual model for representing a DoD. The Relational Model consists of three components: a structural part, a manipulative part, and an integrity part. The structural part is the familiar definition of relations (tables), the manipulative part is the set of operators which can manipulate those relations, and the integrity part is the set of rules which are required to capture all the semantics of the Domain of Discourse. Two important rules concern the maintenance of primary and foreign keys. The essential rule that all instances should be distinct and identified by a unique primary key, and that a foreign key should be consistent with the primary key of the host relation, are not usually enforced by current relational database management systems, but are essential to the consistent modelling of the Domain of Discourse.

Thus within a Relational Model just as within a relational data base there is little semantic richness. All classes, relationships, and frequently domains, are represented as tables of data. Although the integrity constraints should enforce consistency with the DoD they are usually expressed at a logical level such as "employee_id is NOT NULL". The semantics of the DoD behaviour which this is enforcing is not evident.

Thus, whereas the modelling constructs of the Relational Model are adequate to model data and can easily accommodate the typical conceptual models in use in many organisations such as accounting formats, it is only an experienced analyst who can make an immediate transition from the problem domain to a relational representation. Moreover the relational representation does not conveniently permit the representation of the full model, the possibilities implicit in the

relational functions, and the semantics implied by the integrity constraints.

3.2.2 Entity Relationship model

The Entity Relationship model (Chen 1976) is more usually regarded as a conceptual model in the terms of figure 3.1 and is less likely to be confused with the needs of the physical modelling environment. The number of elements in the model is very small: Entities, Attributes and Relationships. The E-R model is much more popular as a conceptual modelling tool than the Relational Model.

The basic elements are: Entities represent the objects in the Domain of Discourse (students, lecturers, rooms), Attributes are the attributes of those objects (height, age, size) and Relationships model the links between the entities (lecturer teaches student). Like the Relational Model, this small number of constructs makes for simple use. However there are no clear rules as to what constitutes an entity or an attribute and the relationship construct is heavily overloaded, being used to convey many different types of association between entities (figure 3.3).

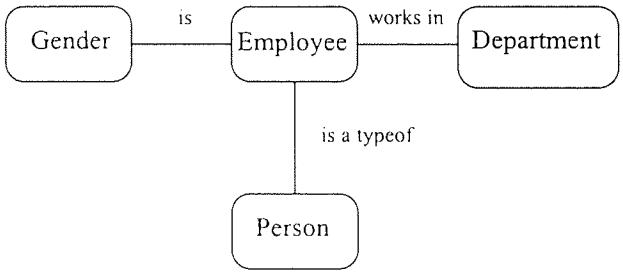

Figure 3. 3 : A variety of relationships

3.3. IMPROVING ON CONCEPTUAL MODELS

Given the limited expressiveness of both the Relational Model and the Entity-Relationship model, there have been a series of improvements proposed over recent years. The aim of all these improvements is to capture and enforce more real world semantics than is possible with the limited number of constructs E-R provides. The initial thrust was the paper by Smith & Smith (1977a) extending the logical constructs introduced by the relational model to include three abstractions - generalisation, aggregation and collections. However Avison and

Hassan (1988) identified a long list of abstraction mechanisms discussed in the literature, generalisation, aggregation, categorisation, association, specialisation, subsetting, role, grouping, characterisation, partition, classification and instantiation. They concluded that most of these concepts are equivalent to others; thus the semantic distinction between generalisation and specialisation relates to the process of defining such classes and not to the nature of the final class structure. Avison and Hassan identify two main distinct abstractions; the generalisation-categorisation-specialisation and the aggregation abstraction which are of course similar to the generalisation and the aggregation abstraction of Smith and Smith. Most subsequent authors have ignored the collection abstraction proposed by Smith and Smith (or have not distinguished it from a class or table). Generalisation and aggregation will be discussed here and collections will be returned to in the context of the CORD model.

3.3.1 Generalisation specialisation

A concept may be a sub type of another concept. It may be more specialised; for example an employee may be weekly paid and another may be an engineer. They are both still employees but have some of their attributes constrained. It is also possible to say that both employees and customers are persons. This is a generalisation which may not be merely gratuitous but may guide actions with respect to Health and Safety or Data Protection legislation etc. If, like the Post Office of old, cats are employed to keep down the mice then the generalisation into persons may not be valid or perhaps it may be possible to coerce a cat to be a person so that the generalisation may be valid.

Two distinct forms of the generalisation abstraction have been identified (figure 3.4). One is the category concept whereby every instance of the super class belongs to one and only one of the subclasses. Thus if all employees are either weekly paid or monthly paid these classes are a subset hierarchy relationship. Another type is the subset or orthogonal specialisation where an instance can belong to more than one of the specialisations. For example an employee may or

may not be an engineer and may or may not be a shareholder (Elmasri et al. 1985, Teorey et al. 1986)

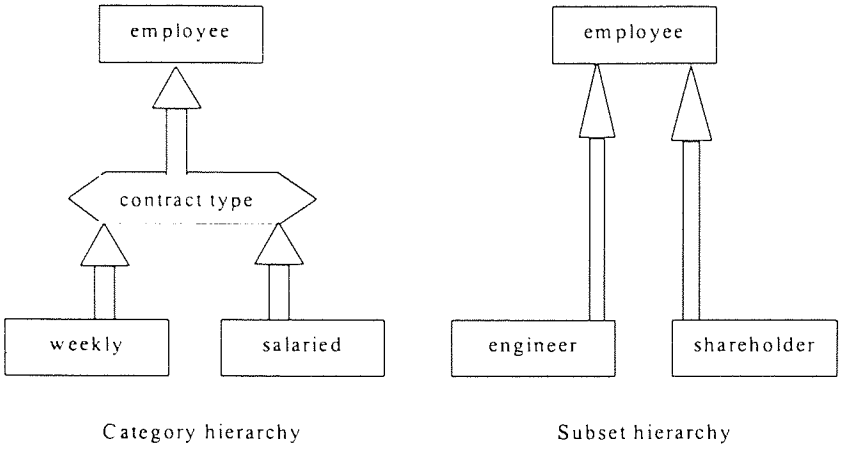

Figure 3. 4 : Two generalisation constructs (Elmasri et al 1985)

3.3.2 Aggregation

Smith and Smith (1977a) in their seminal paper introduced a conceptual model for relations. They proposed the abstraction Aggregation which would allow a "relationship between named objects to be thought of as a (higher-level) named object. For example, a certain relationship between a person, a hotel, a room and a date may be abstracted as the aggregate object *Reservation*. The name *Reservation* may be used without bringing to mind all the details of the underlying relationship."

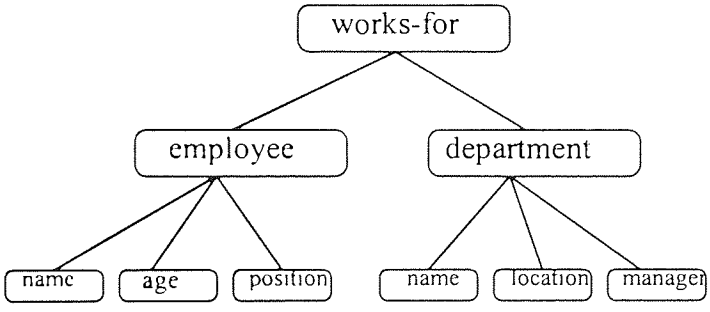

Figure 3. 5 : An aggregation hierarchy

An example of an aggregation hierarchy is shown in figure 3.5. An aggregation appears to be a conceptual device which describes that which is implemented by a record structure or a relation. The authors observe that some abstractions are

introduced to hide implementation issues that the user never wants to see but that abstractions such as aggregation allow context sensitive hiding of detail which in this context the user does not wish to address.

3.4. RECENT APPROACHES TO MODELLING ENTITIES AND OBJECTS

Problems with conventional conceptual models have promoted the development of new approaches to analysis. The work of Smith and Smith (1977a) has been continued by many authors seeking to develop practical modelling tools. Codd (1979) has developed RM/T, an extended relational model, and many authors have proposed extended E-R models. The most fundamental developments have borrowed not only from Smith and Smith but also from Object Oriented Programming and have resulted in various Object Oriented Analysis and Design methods. Various extended conceptual models will be reviewed.

3.4.1 The Extended Relational Model

As a result of the criticisms that were leveled at the relational model, Codd (1979) proposed the extended relational model. Codd's aim was to extend the original Relational Model to capture more semantic detail. The revised model Relational Model (Tasmania) RM/T models the world in entities. It defines an entity as 'any distinguishable object' where the object may be abstract or concrete. Some entities may well represent relationships, but are now clearly identified as such. Every entity in the database is defined to be an instance of at least one entity type. Entities of the same type have the same attributes. Entities can have subtypes and supertypes. Codd identifies three classes of entity:

Kernel entity - this represents a real-world object. A kernel entity is a fundamental entity that is neither associative nor characteristic. Examples of kernel entities are employee, project.

Characteristic entity - this describes some aspect of another superior entity.

Thus, the characteristic entity is existence-dependent on the superior entity. The effect of this is that the integrity rule is implicit in the model, whereas, in the relational model, explicit referential integrity rules are required to enforce such a condition. An example of a characteristic entity might be the *job history* of an employee. The job history can only exist if the employee exists.

Associative entity - this represents a many-to-many relationship between two otherwise independent entities. Associative integrity enforces the rule that an associative entity can only exist if all the entities that participate in the association also exist. Again this form of integrity is enforced by relational integrity rules in the relational model. An example of an associative entity might be the *assignment* of employees to projects.

The syntax of RM/T allows the class of an entity to be identified and therefore the system to restrict its manipulations to operations appropriate to that class.

By creating the concept of an E-relation as well as the traditional relational table which holds attribute data (P-relation) RM/T is able to implement specialization. The E-Relation is a table which holds the identifiers of entities which are members of a particular class. This enforces the ideas of object identity which were absent in the original relational model.

3.4.2 The Extended Entity Relationship Model (EERM).

Just as Codd (1979) found it necessary to acknowledge some of the problems with the relational model, so many authors have proposed extensions to the E-R model all with the same objective of increasing its semantic precision. The original model of Chen (1976) specified two entity types: strong entities rather like kernel entities of RM/T which exist on their own and weak entities which are existent dependent on others. In later modelling the use of cardinality constraints on relationships seems to be preferred to the identification of weak entities. In extending the E-R model Elmasri et al (1985) like other authors have focused on adding some form of specialisation. The distinction was also made between

subset hierarchies and generalisation hierarchies. A subset hierarchy is where it is possible for overlapping subsets to exist so that specialisations of employee could be engineer and shareholder which are not mutually exclusive. A generalisation hierarchy is where the specialisations are mutually exclusive so employees are either monthly paid or weekly paid but not both at once (figure 3.4).

3.5. OBJECT ORIENTED ANALYSIS

Object Oriented Analysis (Coad 1991) focuses attention on the identification of classes and objects with the problem domain. The aim is to identify Objects in the problem domain and classes of Objects. The notation is clearer enabling the user to distinguish the class from the instance of the class or Object and hence permits the identification of abstract classes which have no instances.

In suggesting likely objects for inclusion in a conceptual model Coad recommends that the analyst explores:

- Structures: the relationships between objects within the system;
- Other systems and devices which connect with the main system described by problem domain;
- Things or events remembered: Significant problem domain past events may be important objects in the system, even though they no longer exist or represent a transition in time;
- Roles played: some objects within the problem domain, in particular human actors may play several roles and although they are at one level members of the same class (e.g. clients) they may be sufficiently distinct in respect to the problem domain to need to be treated as different classes;
- Operational procedures: problem domain procedures such as *order verification* may be objects within the system, as at a later stage of development may be some of the procedures from the information systems itself for example *user registration.*
- Locations : sites and geographical locations may be objects;
- Organisational units: structures of the organisation will often be relevant objects, however it is necessary to be cautious of accepting them too early if business process redesign is a part of the project

The distinction between classes and objects is important and this is emphasised in the notation recommended for OOA as shown in figure 3.6.

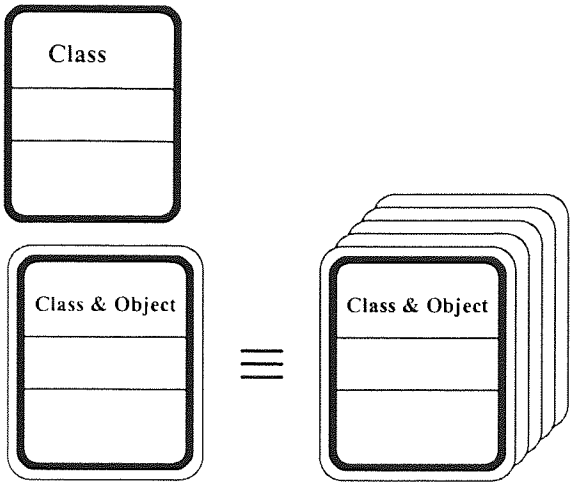

Figure 3. 6: Representing Classes and Objects in OOA

3.5.1 Characteristics of Objects

Coad (1991) identifies some of the characteristics of Objects against which a concept can be examined for inclusion in a list of objects.

Objects will usually have:

- needed remembrance: the system needs to know things about an object;
- behaviour: usually the object experiences change during the operation of the system under consideration;
- attributes: usually an object has multiple attributes of interest;
- instances: usually there is more than one instance of a given class.

Coad specifically identified derived results, reports and summaries as concepts not to be treated as Objects within the problem domain.

3.6. OMT

In OMT Object Modelling Technique Rumbaugh (1991) gives a general definition of what constitutes an object

"We define an object as a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand. Decomposition of a problem into objects depends on judgement and the nature of the problem. There is no one correct

representation."

3.6.1 Classes

Rumbaugh (1991) defines classes in the following way:

"An object class describes a group of objects with similar properties (attributes), common behaviour (operations), common relationships to other objects and common semantics." "Objects in a class have the same attributes and behaviour patterns. Most objects derive their individuality from differences in their attribute values and relationships to other objects." "Objects in a class share a common semantic purpose, above and beyond the requirement of common attributes and behaviour."

In the OMT notation, as in OOA, the distinction between classes and objects is preserved, as illustrated in figures 3.7 and 3.8.

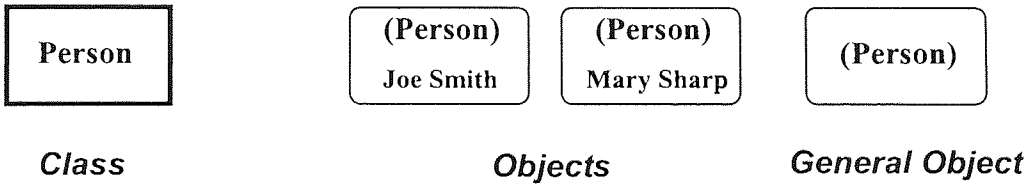

Figure 3. 7 : OMT notation for classes and objects

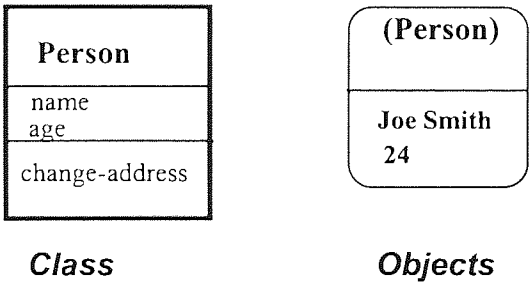

Figure 3. 8 : A class and object with attribute values

3.6.2 Associations

Rumbaugh (1991) defines a link as "a physical or conceptual connection between object instances", and an association as "a group of links with a common structure

and common semantics". "All links in an association connect objects from the same classes."

Relationships have classes and instances just like objects and this is reflected in the OMT representation, see figure 3.9. Relationships are the subject of more detailed discussion in chapter 6.

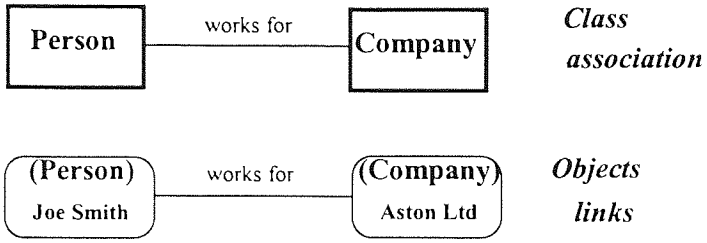

Figure 3. 9: Links and associations (from Rumbaugh)

3.6.3 Object structure

Much discussion of object modelling treats the many types of association as aspects of the one phenomenon and we can identify the following associations:

3.6.3.1 Generalisation/specialisation/inheritance

A class is defined as being a specialisation of another class. The sub class can be said to inherit certain characteristics from its superclass and the superclass is said to be a generalisation of the subclass (and other sub classes). In figure 3.10 a vehicle is shown as a generalisation of the concepts car and lorry.

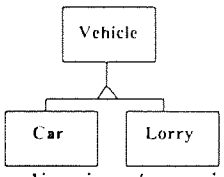

Figure 3. 10: Generalisation / specialisation structures

3.6.3.2 Whole-part structures

In OOA the whole part structure is identified as is the aggregation association in OMT. These are ways of representing the fact that an object may be composed of other objects. An example of a whole-part structure is shown in figure 3.11

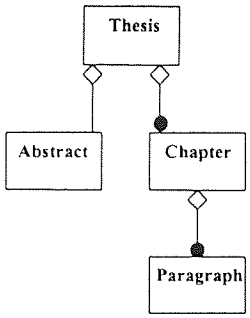

Figure 3. 11: Whole part structures

The whole-part abstraction is itself very rich and deserves further discussion. As it usually relates independent objects it will be discussed in Chapter 6 with other relationships.

3.7. ENTITIES AND CLASSES

In this chapter, in reviewing the various approaches to modelling the original authors' terms have been used. Terminology has evolved over time and it is worth clarifying the distinction if any between the terms Entity and Object. Both terms are used to describe the basic elements being modelled, the customers, parts and orders. Both can be generalised to give entity types and object classes which represent a number of similar entities and objects. Beynon-Davies (1996, p 186) suggests that the difference between an entity and an object is that entities are part of a static data description and objects aspire to define a behavioural component as well. As this research will, from the next chapter on, be concerned about behaviour, it is appropriate that the term *Entity*, unless deliberately referring to its use by other authors, will be superseded by the use of the term *Object* and that *Class* will be preferred to *Entity Type*.

3.8. CONCLUSIONS

The introduction of the Entity Relationship model has focused both researchers' and practitioners' attention on the advantages and problems associated with modelling objects in a complex real world with a limited set of constructs. Since Chen (1976) many authors have tried to find solutions to conceptual modelling which are an appropriate compromise between being sufficiently rich to give

realistic models and sufficiently simple to be practical in use.

The main features of a static model have emerged. It consists of

- objects and classes linked by
  - inheritance,
  - aggregation and
  - relationship structures.

Nevertheless the static model is only part of the story; in the design of an information systems the interest is in the dynamics of the system being modelled, and it is the management of change which is the challenge not the modelling of structure.

Chapter 4

Modelling the Dynamics of Objects

4.1. INTRODUCTION

Most object oriented data models appear to concentrate upon the static description of objects almost to the exclusion of their behaviour. This is evident in early models such as the E-R model which included no constructs for the representation of the behaviour of entities at all, the development of process modelling being treated separately through the use of functional analysis, data flow diagrams and flowcharting. The early attempts to integrate data and process models have culminated in Object Oriented Modelling approaches which specifically include specification of process or behaviour as part of the modelling approach. The main trends in the modelling of behaviour will be examined and discussed in this chapter.

4.2. STATES AND EVENTS; PROCESS AND BEHAVIOUR

An object modelled by a structure in an information systems undergoes change; if not we would not be interested in it. All change is reflected in the database representation of this object by changes in values of attributes or by the creation and deletion of relationships involving the object. At any point in time the object has a *state* in the real world and this is reflected by a set of values for its attributes in the information system.

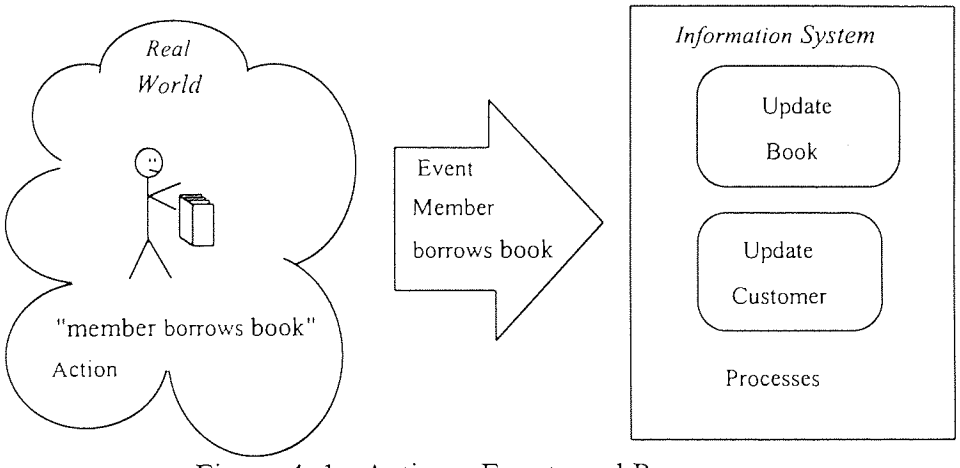

Figure 4. 1 : Actions, Events and Processes.

There is a lack of common vocabulary but *event* is commonly used to describe the

result of an *action* which changes the state of an object; this is an event in the life of the object. An event results in *operations* or *processes* which change the state of an object. Events are instances of a generic event for a particular instance of an object and as with objects it is not always necessary to distinguish between particular events (instances) or types of event that can occur. The relationships between these concepts are illustrated in figure 4.1

In the real world things happen and it is the role of an information system to keep track of these happenings. In the real world a customer acts and borrows a book, the book suffers the action. The whole process is termed an event, which needs to be reflected within the information system. In particular the customer record needs to be amended to reflect the fact that the customer has borrowed something, and the book record needs to be amended to reflect the fact that the book has been borrowed. There is a deliberate vagueness in the modelling process, the objects within a model are often referred to as if they represent the real world equivalents, however in fact they represent the information system objects which are intended to track the real world "partners". This does not matter so much when one is discussing attributes, since attributes of customer and attributes of the customer object are very similar, but it is important when actions are being discussed. The action of a customer borrowing a book is very different from the action of updating the customer record. For this reason it is preferable to see events as the mapping process between the actions in the external world and the processes within the information system. Although the terminology is not very rigorous, objects both outside and inside the system can be seen to be performing or suffering actions, but the co-ordination of such actions within the IS is carried out by events. Unlike the OOP approach where objects send messages to each other triggering off processes, events are seen as co-ordinating processes linking together actions of individual objects and ensuring that the necessary integrity constraints are respected, so that constraints in the real world are also respected within the information system. An event as it has been defined is not unlike a transaction in traditional IS; it cannot be partially carried out without the system

becoming inconsistent.

The term *behaviour* is used to describe the totality of processes which an object suffer and within the conceptual model defines all events in the real world that the objet can react to and all processes which result from them. Behaviour is object centred in that it attempts to represent in the conceptual model the main activities of the real world object. At a lower level behaviour will translate into specific actions and processes at the higher level the behavioural model will be concerned with the real world events which trigger change and the inter dependency between actions and processes

4.3. APPROACHES TO DYNAMIC MODELLING

4.3.1 The non equivalence of DFDs and Activity Graphs.

Falkenberg et al (1991) have compared the traditional data flow diagram and the petri net based activity graph approaches to modelling activity in an information system. They conclude that shallow analysis as in Olle et al. (1988) leads to a false impression that these approaches are essentially equivalent. However the authors demonstrate that this is not the case.

4.3.2 Action Modelling

Feldman and Fitzgerald (1985) introduce Action Modelling: an approach to specifying the dynamics of an information system which is based on a petri net model. They distinguish between user requirements modelling and technical requirements specification. The strength of Action Modelling is claimed to be that it focuses on the user requirements modelling process, whereas many dynamic modelling tools are focused on the technical requirements process, and that it parallels E-R modelling in its constructs and so provides the user with a consistent interface in exploring the conceptual model. In discussing the different approaches to process modelling, Feldman and Fitzgerald distinguish between *Functions* and *Actions*. *Functions* (taken from the functional analysis stage in

many systems analysis methods) describe "the general processing requirements of an organisation". *Actions*, on the other hand, they describe as reflecting "the actual behaviour of data". An elementary action "is part of a sequence of processing which is undertaken on a single entity without involving any other entities". *Events* and *Triggers* are also defined: *Events* are distinguished "by being outside the control of an investigated system world and are either interactions of the area under investigation (the system world) or are due to the passage of time e.g. end of month." *Events* trigger *Actions*. The claimed relationship between E-R data structures and Action Modelling gets lost as the details of the Action Modelling process emerge. However there is an echo of data driven structured programming in the relationship between data organisation and processing description.

4.3.3 Events and States

The enterprise modelling approach of Sen and Kerschberg (1987) is designed to model business processes and thus focuses on *Events* and *States*. They define an *Event* as "an occurrence of something in the environment under consideration".

- An event: has a start time and a finish time.
- An event: represents an action of some kind, and it is typically described by verbs.
- An event: can be external or internal to the DoD. An external event may cause internal events.
- An event: may have zero duration, i.e. it is instantaneous.
- An event: that has a non-zero duration may be decomposed into other events and is called a composite event.
- A state: is defined as a "static component of the DoD that provides status information regarding organisational information objects".

The actual modelling tools used by Sen and Kerschberg include E-S networks (Entity State networks) and diagrams and Activity nets - Activity being defined as a job a user or group of users performs. Both these tools are Petri net in style.

4.3.4 Life Rules

Jardine and van Griethuysen (1987) introduce life rules in their INFOMOD logic based modelling language. They argue that a Jackson (1983) based entity life history is only valid when entities have no associations with other entities. They prefer to use a set of transition rules. Their example defines the state of being *widowed* :

```
Person Marital-state = "widowed"
if and only if
    not Person married to PersonA
    and there is a PersonB :
    Person married to PersonB
    immediately precedes
    not Person married to PersonA and PersonB is-dead.
```

4.3.5 Action Concept

Put (1988) attempts to add a dynamic dimension to the E-R Model by introducing the *Action* concept. He rejects the Petri net approach because:

- Simple concepts, e.g. conditional branching, are difficult to represent;
- The level of description is often too low, with artificial synchronisation points, and transitions without real world meaning;
- Historical information must be dealt with explicitly, because a Petri net has no memory of past states;
- A modular approach is not advocated, such that nets may become extremely complex.

An *Action* "corresponds to a real world happening of interest, occurring instantaneously (it has no duration and cannot be decomposed into meaningful subactions." A real world phenomenon which is characterised by duration is represented by two *Actions*: a start process and an end process *Action*. An *Action* is the smallest possible conceptual unit preserving consistency.

Thus there is a link between the granularity of the data model and the time slices which are of interest. A sequence diagram is a tree with actions as the leaf nodes and the standard (Jackson 1983) constructs link branches together. Although each sequence diagram is an independent process, synchronisation occurs when action

nodes on different diagrams are simultaneous. Put describes a Prolog based system for updating databases using the *Action* concept as the information source.

4.3.6 OOA

In Coad's OOA (1991) the dynamics of the model are represented by defining *services*. The modelling tools used are Object State Diagrams, a state transition model, and Service Charts which are similar to traditional flow charts.

4.3.7 OMT

Rumbaugh (1991) defines a "dynamic model" of a system which consists of "multiple state diagrams, one state diagram for each class". Events are defined as "something that happens at a point in time". Events are unique, but can be grouped into classes having common structure and behaviour. An event conveys information from one object to another, and events have attributes which convey information. Like the term object, event may refer to the class or the instance. The sequence of event occurrences is called by Rumbaugh a scenario; an augmented scenario is also defined which shows for each event the objects exchanging events. States are defined as "the interval between two events" and a state diagram shows the permitted sequences of states, showing which events cause which state changes. An example of a state diagram is shown in figure 4.2.

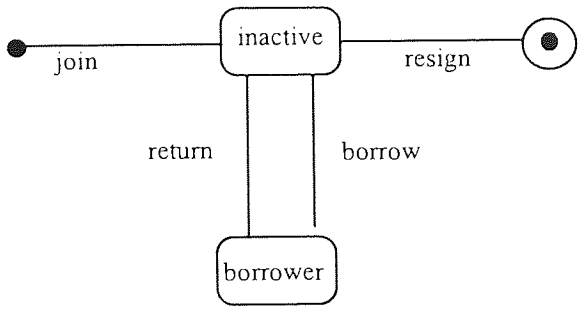

Figure 4. 2 : State diagram (after Rumbaugh)

Various enhancements of the original model are identified. Nested state diagrams, like levelled DFDs, enable detail of processes to be placed at a lower level reducing the complexity of any given state diagram. Activities are linked to states

and are performed while the system is in that state; unlike events, activities have duration. Actions are linked to events.

The State diagram in OMT is intended as a conceptual modelling tool which can represent a lot of the detailed dynamic properties of system. Design and implementation involves the change from the DoD oriented state model to the system oriented functional model which is based on a DFD.

4.3.8 CEM nets

Heuser et al. (1993) use a Petri net modelling approach to support the dynamic properties of a data model. They define "compact condition/event modelling nets" (CEM nets). The approach is to enhance the Petri net representation of processes by adding "dead links" to model static properties. They thus produce a combined modelling approach which incorporates both the structural relationships (constraints) modelled by an E-R diagram and the dynamic aspect of the system modelled by a Petri net approach.. They conclude that this integrated modelling helps distinguish between time varying sets and constant sets which have serious implications for implementation. Moreover they argue that the classic problem of how to model *car has colour* as a relationship or an attribute is helped by the integration of dynamic and static properties. In this context they observe that "An interesting future work could be to study the presented (sic) approach with object oriented modelling in view".

4.3.9 Behaviour Network Model for Conceptual Information Modelling

Kung (1993) offers the behaviour network model (BNM) as another integrating modelling approach which brings together an Extended E-R model and a Petri net type model of the dynamic aspects. It adds nothing to the collection of proposals already discussed.

4.4. THE ENTITY LIFE HISTORY

Jackson (1983) introduced into Jackson Structured Design the entity structure step. In the entity structure step "the developer specifies how the actions of each entity are ordered in time". What Jackson called entity structures are more commonly referred to as Entity Life Histories (ELH) (Ashworth 1992). The ELH can be represented as a tree structured diagram which supports the three standard constructs of structured programming: sequence, iteration and selection. It is relatively straightforward to derive a simple model which intuitively represents the behaviour of an object and accurately embodies many of the real world constraints on that behaviour. An example of a simple ELH is shown in figure 4.3. The traditional tree structure has an entity (object) name at the root and events at the leaf nodes. The non-leaf nodes are collectors defining the sequence of actions applicable to the lower level nodes. Whereas leaf nodes describe instantaneous actions, non-leaf nodes describe a pattern of events over time.

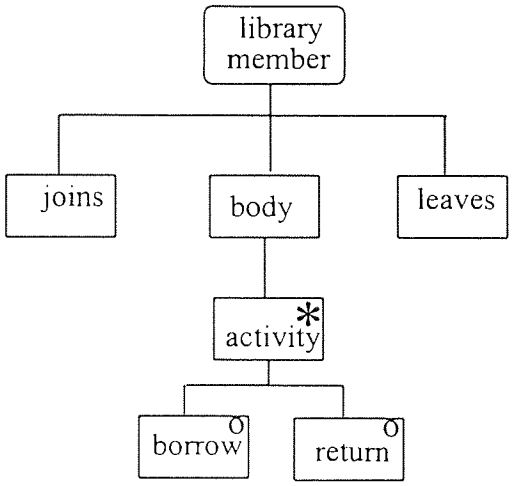

Figure 4. 3 : Library member model (after Jackson)

The example model states that a library member joins, is a member, and then leaves. The asterisk (*) notation means the activity is repeated many times, while the zero (o) means these are alternative actions. Thus the model states that the life of a member consists of repeated activity of either borrowing or returning a book. Sequences of events can be identified which are valid (such as Join, Borrow, Borrow, Return, Return, Leave) and distinguished from sequences which are not

valid (such as Leave, Join, Borrow, Borrow, Leave).

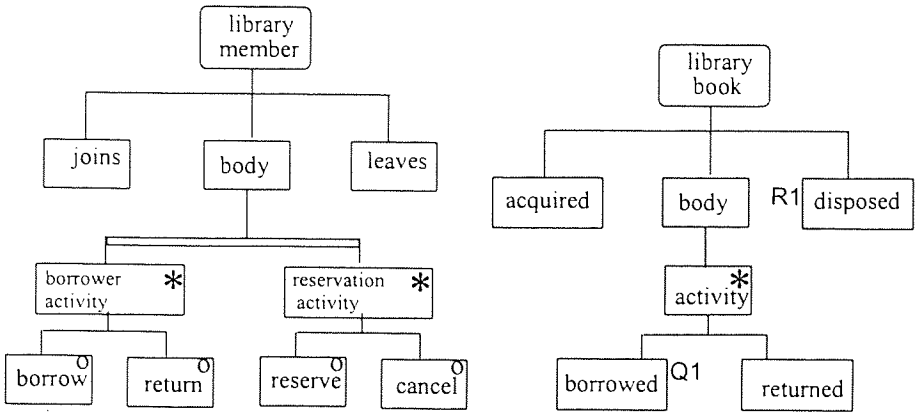

Figure 4. 4 : Examples of Entity Life Histories

Although the basic model will adequately describe many life histories, two other constructs have been added to deal with parallel activity and untypical exits from the normal sequence. These are both part of Entity Life Histories as used by SSADM V4 (Ashworth 1992). The need for parallel notation can arise from the fact that the entity can be engaged on more than one independent sequence of activities which it is wished to describe separately. The quit and resume feature is to deal with cases when processing at one level ends in a non-standard way, and this needs to be reflected in the subsequent processing at another level.. The use of these extensions to represent the parallelism of borrowing and reserving and the fate of a book which is not returned are shown in figure 4.4.

There is a difference of opinion as to how much information should be embodied within an ELH. Thus the "library member" ELH does not preclude a book being returned without being borrowed. Rather than seek for complex modelling, it can be argued that this is a property of a book and not of a member and would appear on the "library book" ELH as a sequence. Because the events "borrow" and "return" are synchronous between the two ELHs the necessary constraints apply to both ELHs.

4.5. CONCLUSIONS

Whilst there is considerable agreement about the static elements of an object

model, there is less consistency about the appropriate approach to modelling the dynamic aspects. Most approaches rely on specification of production rules or of using Petri net models. Neither method appears to be user friendly enough for the purpose of developing conceptual models. Moreover none of these methods describe the behaviour of an object but rather its response to certain triggers. On the other hand the Entity Life History approach seems more amenable to user oriented conceptual modelling as it permits the user to focus on the high level behaviour of the system without descending too rapidly into technical specification. The relationship between ELHs and the object model in particular with respect to subtypes and inheritance needs to be better understood before it could be incorporated into an Object Oriented approach to conceptual modelling. The modification to ELHs to permit parallel activity is a clue that the relationship of behaviour of objects and their subtypes and roles will benefit from further examination.

Chapter 5

Entities and Roles

5.1. INTRODUCTION

The review of the dynamic modelling of data in Chapter 4 revealed a range of approaches and tools which were mainly concerned with describing processes and showed little attention to the behaviour of objects as such. This no doubt reflects the traditional process orientation of much of modern systems development. The Entity Life History (ELH) has the potential to be a user friendly modelling tool which, focusing on the behaviour of objects, meets the need for a conceptual modelling tool which is compatible with object modelling methods. However this is not the only way of linking object modelling and behaviour. In his research into the nature of objects, Wieringa (1991) identified roles played by objects as an important abstraction and saw these as being temporal objects bridging the gap between object definition and object behaviour.

This chapter builds on the work of Wieringa exploring not only the nature of roles but also the implications of Inheritance on them, their relationships with the specialisation abstraction and the connection between both roles and specialisations and Entity life Histories. The aim is to identify a set of user friendly conceptual modelling tools which will enable behaviour to be integrated in the Object Model.

5.2. SUBTYPES, ROLES AND SPECIALISATION

A central element of any object oriented conceptual modelling is the establishment of subtype relationship between classes. The importance of this relationship is the inheritance of attributes and behaviour by specialisations of the superclass. The two main abstractions are the IS_A (generalisation) and the PartsOf (aggregation) abstractions (Smith 1977a,1977b). It is to be expected that a subtype will behave in most respects like its parent. However in most approaches to OOA subtyping is specified almost entirely in terms of the inheritance of attributes of objects, whereas it is equally significant with respect to

behaviour, an aspect represented well in Object Oriented Programming languages but under emphasised in most OOA.

In modelling, an object is often encountered which for part of its life displays different characteristics. An employee may for a period serve on a health and safety committee. A library book will periodically be out on loan. It is an important aspect of modelling that objects are gathered together in classes so that their characteristics can be economically described. The identification and representation of classes and the relationships between them is central to all conceptual modelling approaches. Object oriented methods add the important abstraction of subtyping, whereby classes can themselves be linked together, further economising on the description of their characteristics. In developing an information system, these classes derived during the analysis and conceptual modelling stages serve not only to define the storage requirements but also most of the integrity constraints which apply to the data, and, where behaviour has been modelled, most of the procedural aspects of the system being developed.

To model this important characteristic (i.e. that objects may have different periods of activity during their lives) Wieringa (1991) distinguishes two types of class: the *natural kind* and the *role*. An instance of a *natural kind* is a real world object which exists independently of any particular activity within the Domain of Discourse, and it cannot exist without being an instance of that kind. On the other hand a *role* instance can exist without playing that role (Wieringa 1991). Thus a female-person as a specialisation of person is distinct from teacher which is a role played for a period by a person. The person exists before being a teacher and probably continues to exist after being a teacher, thus the object instance has permanence outside of the role teacher. However a person is either a female-person or a male-person throughout their life and this specialisation is coterminous with the existence of person.

The distinction between Wieringa's natural kind and role are clearly important but are usually modelled by the same subtyping constructs (Eckert 1994a). This leads

to many problems in the conceptual modelling stage which may be inadequately resolved and remain to be solved in the design or implementation phases of the project. The distinction between the two Wieringa types can be made: referring to natural kinds by Smith and Smith's (1977a) term specialisation, and retaining the term role for Wieringa's role, and by using the general term parent to describe the superclass of the role or specialisation. The term role is also used by other authors in a similar way but usually as a way of viewing an object or of trying to understand relationships. For example it appeared in Chen's paper of 1976 where he uses role to describe the "function that (an entity) performs in a relationship". The strength of Wieringa's approach is to see role as a temporal specialisation.

In terms of the subtype relationship Wieringa asserts that "A role must be the subclass of at least one natural kind and that no natural kind is the subclass of a role". Elmasri (1992) has also discussed the concept of role in an identical way to its use in this paper. He enumerates some characteristics of roles, in particular their visibility within the owner entity.

- Their life span starts after the start of the parent entity;
- A role type is restricted to exactly one owner entity type;
- A role type can have only temporal attributes;
- Attributes of a role type are public to the owner entity type;
- Attributes of an entity type are public to all its associated roles;
- A role can access all relationship instances for relationship types in which the associated role participates.

It is not necessary to agree with all of these points. The first three points follow from the definition of a role as a temporal. With regard to 4 and 5, it is contrary to the principle of encapsulation to encourage communication between roles and owner entities. A minimum would be to expect the owner to track the existence of all live roles and depending on the needs of the application merely count the number of dead roles or keep a record of them. Any updating of the attributes of a role instance should be through the role itself and not directly by the parent object.

In general a role may be held concurrently with other roles and even multiple

instances of the same role may overlap in time.

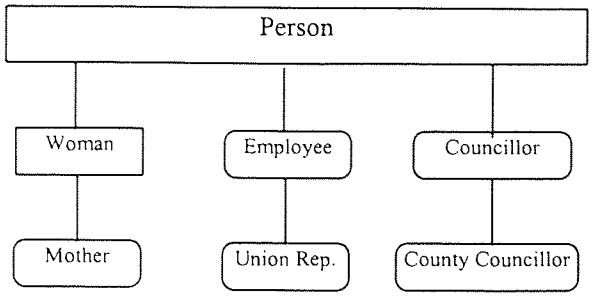

Figure 5. 1 : Subtypes and Roles

In the example represented in figure 5.1, *Woman* is a specialisation of *Person* whereas *Mother*, *Employee*, *Councillor* and *Union Rep.* are all roles adopted by a person for a part of their life with no particular sequence except that the role *Union Rep.* is dependent on the role *Employee* and *Mother* is dependent on *Woman*. *County Councillor* is a specialisation of *Councillor*. This chapter will continue to explore how the behaviour of a specialisation differs from that of a role and how specialisations and roles inherit behaviour from their supertypes.

5.3. OBJECT ROLE DIAGRAMS

Given the significance of roles in the behavioural modelling of an information system it is necessary to propose their representation within current modelling techniques. It is inadequate to include them within normal object hierarchy diagrams as their temporal aspects are lost. An alternative which was considered was to include them in ELHs but this also produced contradictions. The ELH is in general transition oriented and cannot easily represent a period without separately representing the start and end of the life of the role. Hence it was decided to propose a new modelling representation simply for the relationship between roles and their parent entities, the aim being to represent in an intuitive but rigorous way the main dependencies between an entity and the roles it can play. This modelling convention is called an Object Role Diagram (ORD).

Given that the conventional view would be to describe roles as happening "within" their parents and that time is commonly represented as flowing from left

to right across a page, a representation of roles as long boxes within longer boxes is proposed. As roles are always optional and may be played at any time during the life of the entity, they are represented by soft boxes.

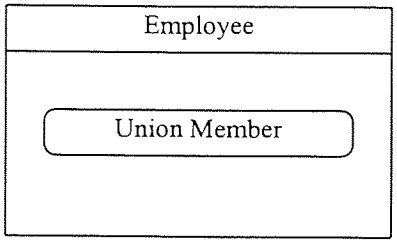

Figure 5. 2 : An ORD - Union Membership as a Role

The semantics of the representation in figure 5.2 is that:

- An employee may play the role union member,
- The employee can play that role more than once and may start and stop playing the role many times.
- An individual who is not playing the role employee cannot play the role union member (in this Domain of Discourse)

It is not necessary to repeat the phrase "playing the role" and where no confusion would arise the verb "to be" will be used instead. In the Domain of Discourse there will often be other more conventional phrases for "playing the role". Thus the preceding statements could reflect a conventional company rule:

"Any employee may join a union."

There is no implication of when, nor for how long, during the playing of the role Employee the role Union Member may be played, any more than there is in the company rule. The box representing the role Union Member is said to be free, as it shares no vertical lines with any other role. Such vertical lines are called synchronisation points, and as will be seen are used to model other temporal dependencies in the Object Role Diagram. Roles that share synchronisation points are said to be bound (temporally).

Thus in a company operating a closed shop agreement, the relationships in the previous example may be shown as in figure 5.3. As there is no doubt over the

duration of the "being union member" nor its presence, because of the closed shop agreement, it is not appropriate to use soft boxes but rather hard (square cornered) ones. As the 'role' is played throughout the life of the parent, it cannot intuitively be shown within the parent box.

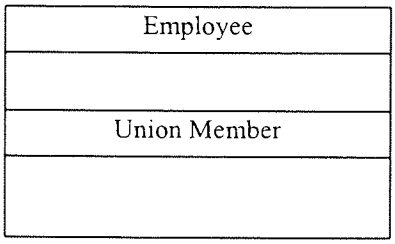

Figure 5. 3 : Not an ORD - Union Membership in a Closed Shop

This states that "an employee is always a member of a union". Such dependencies together with specialisations (some employees are male) which do not have temporal constraints attached to them are not Roles and are better displayed in normal tree structured hierarchy diagrams rather than Object Role Models.

5.3.1 Sub-Roles

Role players may play roles or sub roles.

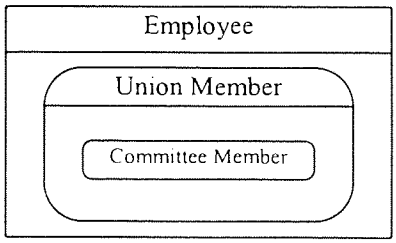

Figure 5. 4 : A sub role

Figure 5.4 represents the situation that an employee may be a union member and a union member may be elected to the branch committee.

5.3.2 Concurrency

It is also possible to express the fact that different types of role may be played concurrently, in an intuitive way, that is consistent with the way that sub roles are represented.

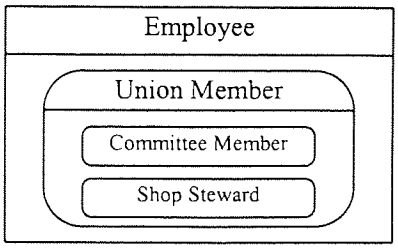

Figure 5. 5 : Concurrent Roles

Figure 5.5 represents potentially concurrent role playing within a role hierarchy. Thus a Union Member may be a Committee Member and may independently be a Shop Steward. There are no implicit temporal limits on the number of times each role may be played, and the model makes a clear statement that they may be played concurrently.

5.3.3 Temporally Mutually Exclusive

It is also necessary to be able to specify that two events cannot occur together.

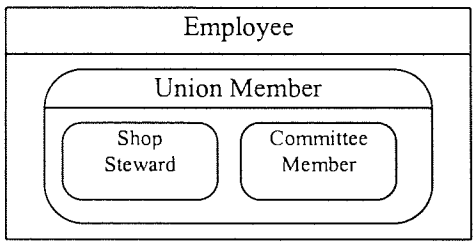

Figure 5. 6 : Mutually exclusive roles

Figure 5.6 represents the following rule: "A Shop Steward cannot be a committee member during his/her period in office". However, there is no implication on how many times or in what order a member may play the two roles, only that they may play any one or neither at any time during their union membership. Given that the horizontal dimension is being overloaded with mutually exclusive roles as well as the time domain, there is a possibility of confusion. However the absence of vertical synchronisation between any roles makes it clear that this represents disjoint activities.

5.3.4 Synchronisation

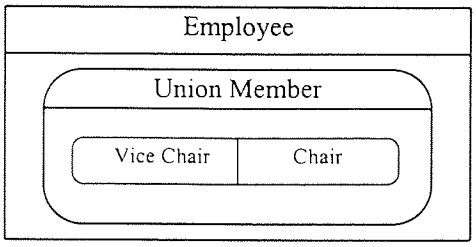

Figure 5. 7 : A sequence of roles

In some organisations there is a practice that the Vice Chair is automatically the next Chair (Chair elect) . By making the start of one role coincident with the end of the previous one, a necessary sequence is defined as shown in figure 5.7. Whilst a member does not necessarily become Chair a member that does will have to pass through the sequence Vice Chair / Chair. The use of synchronised roles although permitted in the notation is generally better supported by the basic ELH that can be used to represent a sequence of events relatively easily. As the end of one role in a sequence synchronises with the start of another the ELH approach which focuses on events rather than roles is often more appropriate. The ORD is best used to identify roles that can be played and any restrictions between them. Behaviour which occurs within the life time of a role is best modelled elsewhere.

5.3.5 The hidden role : further examples in object role modelling

A condition derived in the modelling that it may be required to represent is that of a prerequisite role. Thus it may be that "Any First Aider who wishes to take the Advanced First Aid Course must previously have completed the Introduction to First Aid Course". The temptation to represent this as in figure 5.8 makes several errors.

Firstly it focuses on the transitions rather than the roles. Thus, although being on a course is an activity of an employee which has a temporal aspect it is not, within this DoD, a significant role. Its function is to enable the employee to make the transition into the role of First Aider and it is a part of the life history of a First Aider and should appear there rather than in an Object-Role diagram.

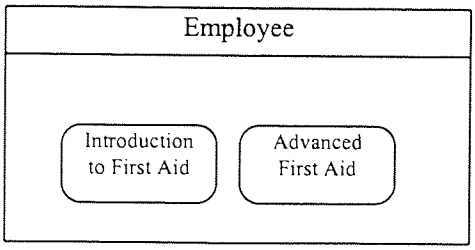

Figure 5. 8 : The missing roles

The modelling also ignores the fact that on completing the Introductory Course the employee becomes a *First Aider*, and it is *First Aiders* who may join the Advanced Course. This hidden role needs to be made explicit as it is likely that other attributes or behaviour will be associated with being a First Aider and these will need to be attached to a role. Figure 5.9 shows the roles of First Aider and Advanced First Aider rather than the training courses which produce them.

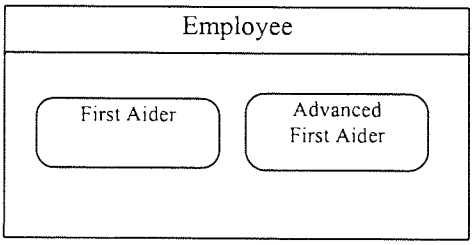

Figure 5. 9 : Roles replacing transitions

However, the modelling is still not correct. Given the semantics of Object-Role diagrams as they have been defined above, an employee could become an advanced first aider without having been a first aider.

In figure 5.10 the modelling is more accurate, representing as it does, Advanced First Aider as a role played by first-aiders. The implication of making the terminating borders concurrent is that on ceasing to be an advanced first-aider one is no longer a first aider.

If an alternative semantics was required, such that an advanced first-aider remains so qualified only if they take regular refresher courses, then the Advanced First Aider role could float within the First Aider Role. Again whilst the Object-Role diagram can be used to represent prerequisites, it may be best to leave the detail of

such dependencies to the ELH and focus on the main optional roles in the ORD.

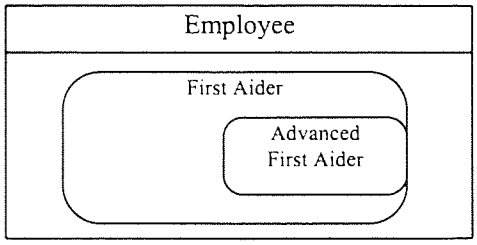

Figure 5. 10 : More accurate representation of advanced first aiders

5.4. RELATIONSHIPS

As Elmasri (1992) and Chen (1976) have suggested, there is an important duality between roles and relationships. Rumbaugh (1991)) also uses the term role to identify instances of objects taking part in an association with another object. Many relationships between object instances are temporal, in that they persist for a part of the life of one of the object instances only. Thus when a person becomes a teacher they gain a teaching post at a particular school. This would be represented in an E-R diagram as a conventional relationship. As a school will typically employ many teachers, and as a person may have many teaching appointments, this is a many-many relationship. See figure 5.11 for the E-R model of this relationship.

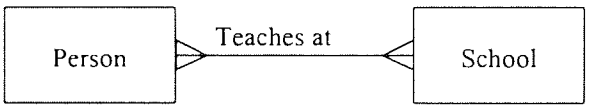

Figure 5. 11 : An E-R model of a relationship

An instance of the relationship involves one person and one school.

It appears that a person is playing the role of teacher which is concurrent with the school employing a teacher or that the role of teacher is a pattern of behaviour of a person while a particular relationship or association is extant.

The cardinality of the relationship is now expressed by the cardinality of role playing, i.e. a person may play the role of teacher many times, and a school may play the role of employer many times. This is shown in figure 5.12

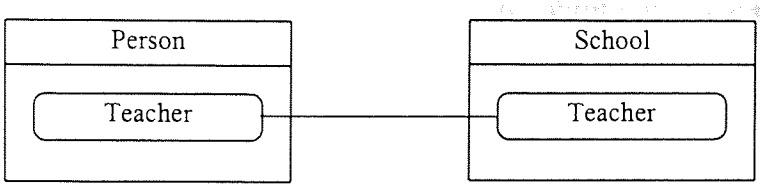

Figure 5. 12 : The Object Role diagram for figure 5.11

This link between roles and relationships is important from an object modelling point of view. It clearly distinguishes specialisations from roles in that specialisations involve only the object itself and are defined with reference to the class of objects; however roles are defined with reference to two or more classes of objects. The alternative formulation of multiple inheritance is inappropriate as although it may be plausible to formulate a teacher as a specialisation of person, inheriting the attributes of its person "parent", it is difficult to assert that a teacher is a specialisation of a school.

The nature of the Role abstraction becomes clearer when its congruency with the abstraction of relationship is understood. Whereas the two main abstractions Specialisation and Aggregation provide a rich modelling framework used independently to describe objects, Role is an abstraction which involves simultaneous use of the Subtype abstraction and the Part_Of abstraction in its instantiation.

It is a strength rather than a weakness of a data model that different modelling constructs interact, and an advantage of distinguishing roles is that it will focus the analyst's attention on relationships and associations which are implicit in the playing of each role.

5.5. FURTHER ROLE MODELLING

Role modelling introduced so far includes the basic structures of roles, concurrent roles and mutually exclusive roles. In the sense that concurrency corresponds to independence and mutual exclusion is a state of interdependency then these constructs cover the range of modelling that is likely to be encountered. An intermediate state would be one in which the fact that a instance of a class was

engaged in playing one type of role changed the probability of it playing another type of role. Although examples of such dependency abound they do not usually translate to a constraint on individual instances.

However there are other aspects of role modelling that it may be appropriate to include in a data model. Both cardinality and optionality of roles will be considered.

5.5.1 Cardinality of Roles

It is a fundamental attribute of role modelling that an instance may play many different roles and may play the same role more than once concurrently and consecutively. It is this aspect of role modelling which distinguishes roles so clearly from specialisations. So a person may have more than one job and be parent to more than one child. It may be appropriate to indicate the maximum number of instances that a role may be played concurrently or cumulatively by an object. The basic notation permits an object to play a role many times without constraint. A notation has been added to indicate an upper limit on the number of simultaneous or consecutive roles of a particular type that an object can play. Thus in a particular Domain of Discourse a person may not be able to hold more than five jobs concurrently or more than 20 in all and this is shown in figure 5.13.

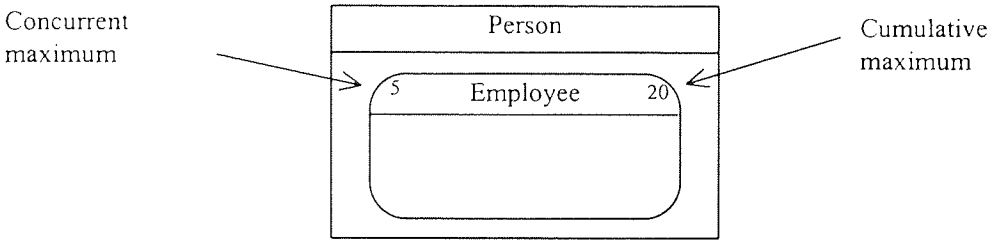

Figure 5. 13 : Cardinality of Roles

5.5.2 Required Roles

It is possible that an analyst may need to represent a real world condition of the type "All Employees must serve for a period as a First Aider". Clearly this translates as a requirement for a role instance of First Aider to have been played at

some time by an Employee. However it would be rather difficult to know how to enforce this rule until a Person tried to cease to be an Employee. In practice the rule is as unenforceable in the data model as it is in the real world. However it would be useful in developing applications to know that nearly all employees will play the role of first aider at some time and that it may be appropriate for promotion or other purposes to know that this has in fact happened. Rather than treating this as a dependency it is treated as a minimum number of occurrences of the role *First aider* and thus as a minimum cardinality constraint.

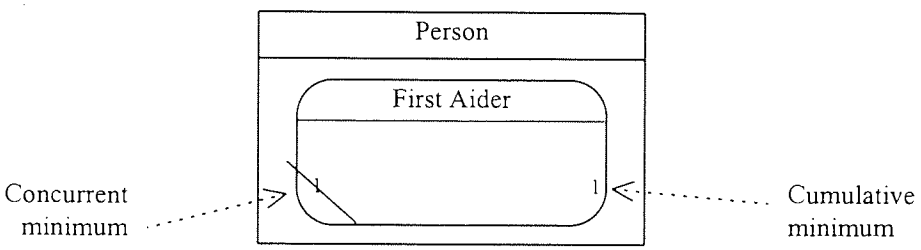

Figure 5. 14 : Required Occurrence of Roles

It is not appropriate to insist that this minimum is played concurrently because like the closed shop example First Aider would cease to be a role but become an attribute of employee. So while the unenforceable cumulative constraint can be represented in the notation, the semantically impossible concurrent constraint should not be shown, as indicated by the crossing out in figure 5.14.

There is no need to permit a notation to indicate that a specialisation is required of a Natural Kind nor that a role is required of a role. A role cannot be required of a specialisation or natural class because it would have the same life as the parent and thus not be role by our definition. Thus a single role cannot be required of another role nor a specialisation of a natural class.

5.6. ROLES AND LIFE HISTORIES

As both roles and entity life histories have temporal content then there are likely to be occasions when they can be used as alternatives in a modelling situation.

The first observation is that the use of roles considerably simplifies the modelling

of behaviour. By partitioning the behaviour of the library member into a Role_Of Person and two subroles, Borrower and Reserver it is not necessary to develop a complex life history for the Member (figure 5.15) but rather three simple life histories for the Member, the Borrower and the Reserver (figure 5.16).

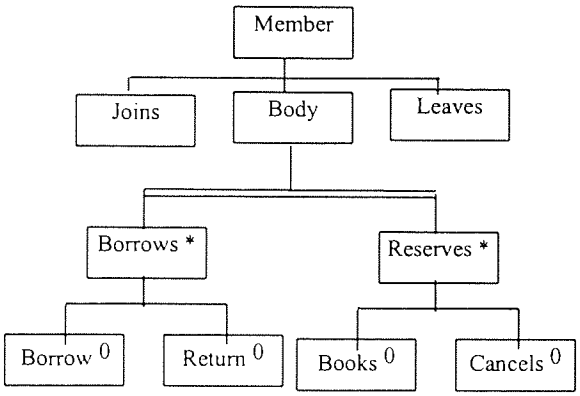

Figure 5. 15 : A complex Entity Life History

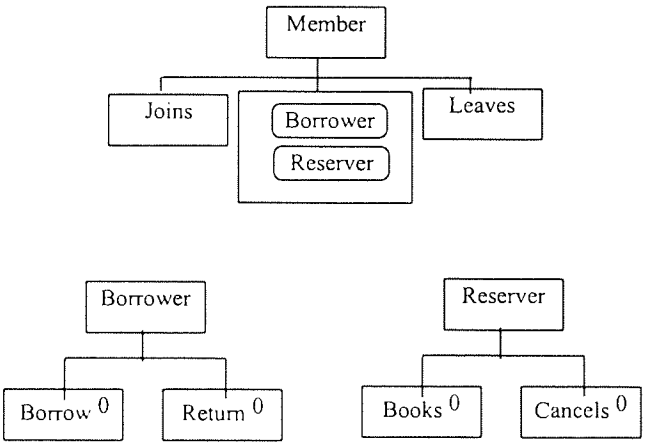

Figure 5. 16 : Three simpler ELHs

5.6.1 States and Roles

Any object has attributes which can take different values, when these values represent changes in the real world perceptions of the object then these are called states. In general, roles are identifiable by state variables, attributes which identify the current state of the object. In the limit it may be possible to define a distinct role for each combination of attribute values; however it is usual only to identify roles which have behavioural consequences. In fact many roles are not identified by attribute values during the initial stages of modelling, which is why methodologies such as SSADM and JSD advocate the allocation of state

indicators to the different stages in an ELH. In general, whilst every role will be associated with the values of some attribute, not every attribute value will be associated with a distinct role.

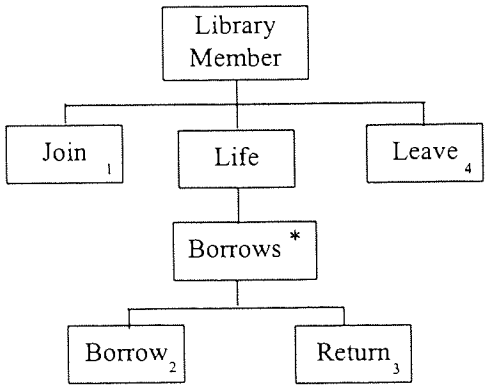

Figure 5. 17 : States and roles

State indicators are usually added to the terminal nodes. A leaf node represents an action which in principle has no intermediate values. A state indicator thus represents the last action performed by the object: it does not represent the state it is in although this can often be inferred from the last action performed. Finite state models on the other hand more clearly indicate the current state of the object, hence the possible actions and possible next states. An example of the use of state indicators is shown in figure 5.17. The convention of numbering states is an implementation issue. A more general approach is to consider that any object which has an ELH can respond to the query

```
object_id.last_action.
```

The use of role modelling simplifies the ELH but this does pose a question as to how to know what is the current status of an object which is playing several concurrent roles.

This is resolved by assuming that on object can also answer the question

```
objectid.current_roles
```

which would return a collection of role identifiers. If several types of role are possible then a selector may be appropriate

```
object_id.current_roles (type = Employee)
```

This would of course be equivalent to a query of the sort

```
all_objects (type = Employee AND is_role_of = object_id)
```

Whether objects remember their active roles and keep a record of them or all objects are scanned for appropriate dependencies is an implementation issue.

5.7. HIERARCHIES OF NATURAL KINDS AND ROLES

The previous discussion has described the Role abstraction and has established a diagramming tool for representing the temporal relationships between Natural Kinds and their roles (Wieringa 1991) and between Roles and Sub Roles. The sub role looks like a specialisation and it is appropriate to explore the possible inheritance hierarchies which may be present in an Object Role diagram. All modelling methods include the subtype property. Following Coad (1991) Specialisation is used to refer to subtyping where the subtype is a natural kind and following Wieringa a contingent subtype is called a Role.

The figure 5.18 summarises the possible relationships. A natural kind can be a specialisation of another natural kind, and a role can be a specialisation of a role. A natural kind can play a role and a role can play a role. No relationship is possible where by a role has a natural kind as a role or specialisation.

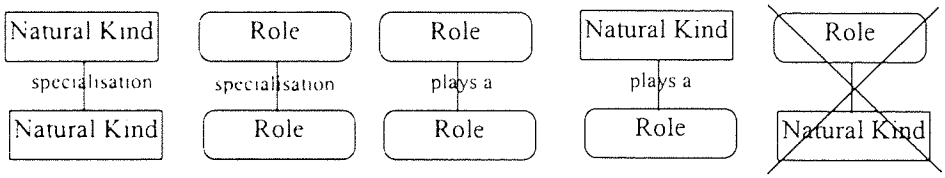

Figure 5. 18 : Specialisations of natural kinds and roles

From an object role modelling point of view the abstractions of specialisation and role playing are fundamentally different. However an analyst not aware of Role modelling will nearly always try to model roles as subtypes. It is useful to explore the different nature of the specialisation and role playing abstractions as they affect inheritance of attributes.

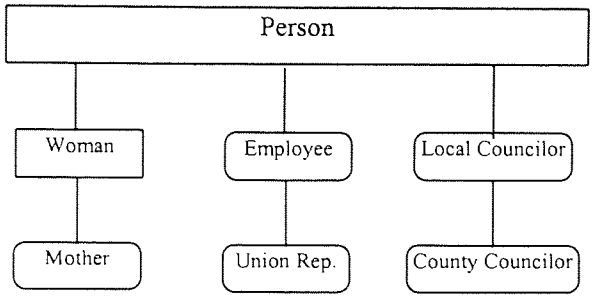

Figure 5. 19 : figure 5.1 reprised

It is an accepted part of object modelling that a specialisation inherits the attributes of its parent. In figure 5.19 a Woman is a specialisation of Person, and as such attributes of Person: Age, Height and Name would all apply to a Woman - Person. Mother on the other hand is a role. A mother may have as many "names" as children ("Ma", "Mommy") with respect to each playing of the role mother there will be different sons or daughters in law, different responsibilities etc. Because a specialisation **is** the parent i.e. there is no difference between the Person and the Woman-Person the issue of attributes is very clear. On the other hand a Role is distinct from the Person so the nature of the inheritance of attributes is different. There is some similarity with the two ways of passing parameters in programming: with specialisation there is only one instance of the parameter, i.e. passing by reference, with roles there is a type of passing by value enabling each role to overwrite the attribute independently.

5.8. LIFE HISTORY AND ROLES

Having identified the possible issues in the inheritance of attributes between roles and specialisations it is necessary to consider the inheritance of behaviour. It is evident from the definition of a specialisation and a role that these two abstractions will have very different properties when considering how they might inherit behaviour from a parent.

5.8.1 Specialisations

A specialisation must share the creation experience of its supertype, thus a woman is "created" at the same time as the supertype "person". As the specialisation

exists throughout the life of the supertype, there must be an attribute set at creation which distinguishes the specialisation and if the specialisation is relevant to the behaviour of the object within the DoD there will be occasions in its life history when a different sequence of events is determined by that attribute value. Thus the main interest in specialisation is that the specialisation experiences a distinct life history. However because of the subtyping relationship we would also expect a substantial similarity between the life history of a specialisation and its supertype. An example of a subtype is shown in figure 5.20.

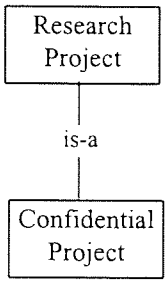

Figure 5. 20 : A specialisation of Research Project

There are two possible approaches. One is the overloading of the entire ELH of the supertype by that of the specialisation. Overloading occurs when a symbol has more than one meaning, usually in different contexts.

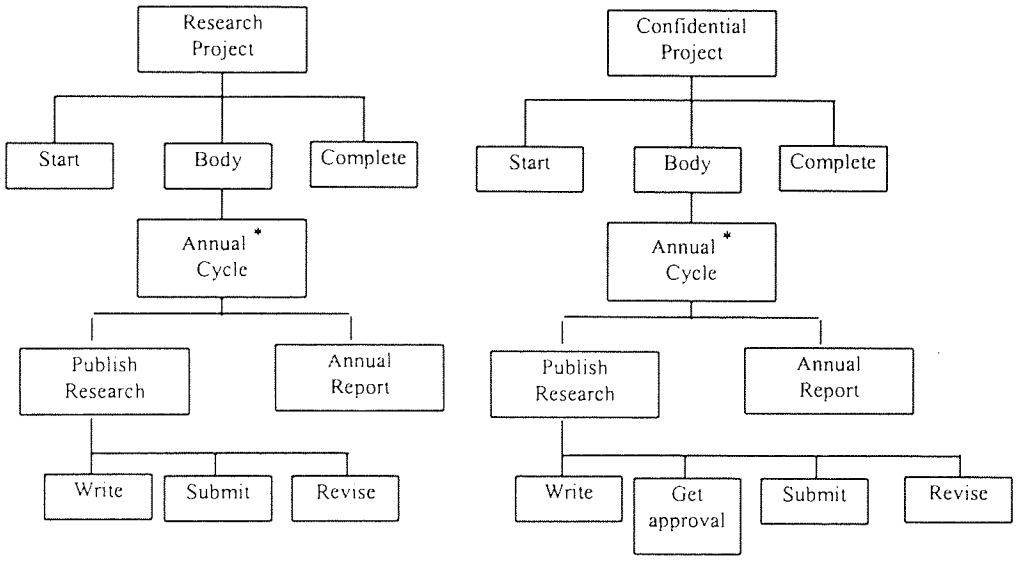

Figure 5. 21 : Two different ELHs completely defined

Thus the 'life of' the subtype could be different from the 'life of' the super type as shown in the example in figure 5.21.

This is not economic and could lead to maintenance problems for the many stages which will be common to both and to any other specialisation. The other is to only overload those actions which differ, as shown in figure 5.22.

Rather than specify in detail all equivalent stages in a subtype's ELH it is only necessary to specify the differences and in the way of an OO programming language the lowest level stage will overload the stage of the same name in the supertype ELH. Moreover it will not be necessary to go into lower detail of any stage which is equivalent in activity to the stage in the supertype ELH.

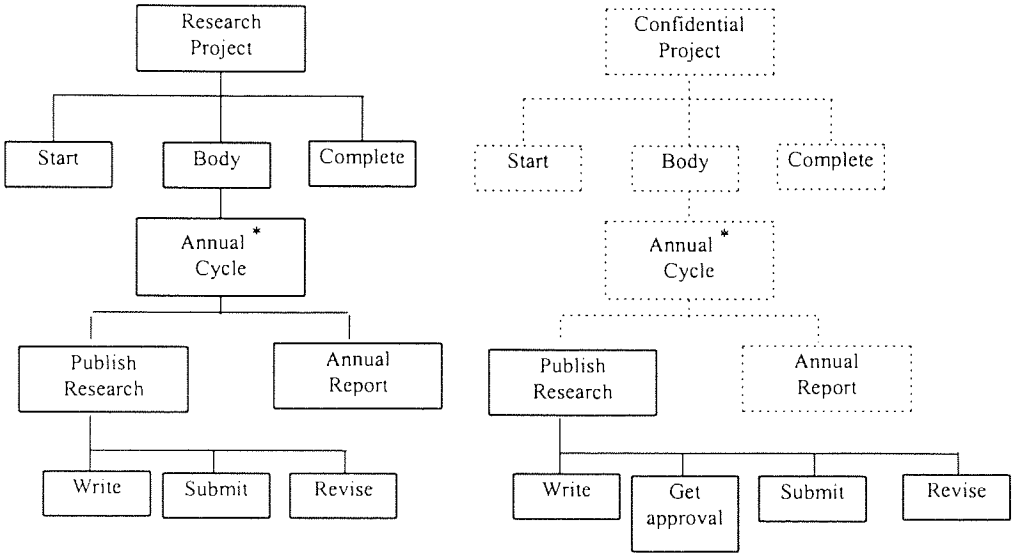

Figure 5. 22 : Only the different activity is defined

Thus using a self evident schema language one could write

```
Object.Research_Project
{ HasELH
 {
  Life : Start, Body, Complete;
  Start : Write Proposal, Get Funding ....;
  Body : (Annual Cycle*;)
  Annual Cycle : Publish Research, Write Annual Report
  Publish_Research : Write, Submit, Revise ;
  Write : ....
  .....
  }
}

Object.Confidential_Research_Project
                (IsSubtypeOf : Research Project)
{ HasELH
 {
```

```
    Publish_Research : Write, Get Approval, Submit, Revise ;
    Get Approval : Send Copy to Sponsor, Make changes ........
  }
}
```

In executing the Life history of a confidential project, in the absence of a full ELH the ELH of the parent would be followed but at each stage the lowest definition of any action would be used. When the ELH attempts to execute the Publish Research activity, it will find a local definition and this will be followed.

A consequence of the need to be able to overload and to use elements from the ELH of a supertype is that stages should be uniquely named within their higher level stage. This means that a full path name of an event can be given without ambiguity e.g.

```
Research_Project, Annual_Cycle, Publish_Research,
Get_Approval.
```

This requirement parallels the labelling of states rather than the use of numerical state indicators as discussed earlier.

5.8.2 Roles

Whereas a specialisation can inherit the entire life history of its supertype, this will not be true of roles. Because they are temporal entities the modelling of roles simplifies and replaces some of the life history modelling of the parent.

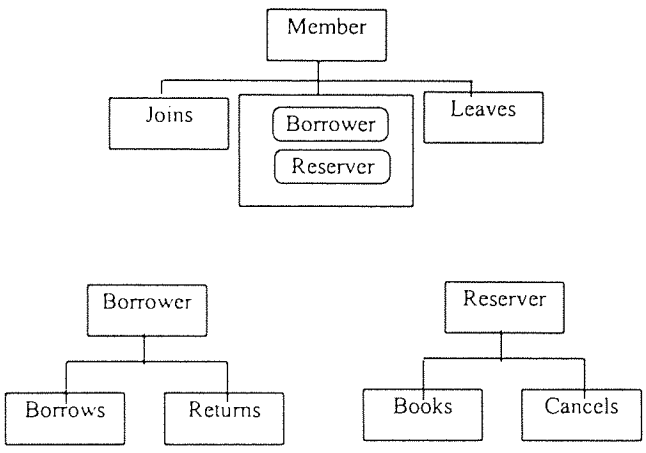

Figure 5. 23 : Life Histories and Roles (figure 5.16 reprised)

In the library example in figure 5.23, which simplifies the more traditional ELH

modelling of figure 5.16, the new model explicitly permits a member to borrow on many occasions, and to borrow many books simultaneously, and recognises that to complete a borrowing session a book has to be returned and that a member cannot return a book before borrowing it. It also makes it clear that to terminate a membership will require the termination of any borrowing roles still live, and to terminate a borrowing activity requires the return of the book. The member can also independently make and cancel reservations.

The implication of this is that whereas for a specialisation the action can be obeyed without confusion by the super type or the subtype

```
project_id. Publish_Research
confidential_project_id . Publish_Research
```

this would not be true of a role. In the following example

```
member_id.cancel
```

it may be clear that the member wishes to cancel a reservation but it will in general not be clear which reservation they wish to cancel. On the other hand

```
reserver_id.cancel
```

makes it explicit.

5.8.3 Termination

There are two termination dependencies for a role. The end of the life of the natural class will trigger the termination of all roles and implicitly all specialisations. Although at some stage the exact details of these forced terminations will need to be modelled, there is no need for this to be represented in a conceptual model. However, there may be roles which cannot themselves terminate until all sub roles have terminated. This is a more unusual situation and may need to be identified so that it is not overlooked in subsequent systems design stages.

5.9. SUGGESTIONS FOR SELECTING APPROPRIATE MODELLING TOOLS

This chapter has introduced and linked various modelling constructs which in less rich modelling environments are usually treated independently. It seems useful to make some recommendations to the modeller as to which abstraction to use under certain circumstances.

5.9.1 Specialisation

Use specialisation and inheritance trees to describe the relationships between parents and multiple mutually exclusive subtypes. Use specialisations of ELHs to describe events or activities which begin or end with the parent, which last as long as the parent or which usually occur and occur only once during the life of the entity.

5.9.2 Sequences

Use entity life histories to describe any unconditional sequence of events.

5.9.3 Roles

Use ORDs to identify optional and possibly multiple roles that may be played by an entity. Exclude the behaviour of those roles from the entity's ELH and specify an ELH for each role. If a potential subtype occurs at any time during the life of its parent, or it occurs several times and lasts for a short but observable time then it is probably best modelled as a role. If a potential role lasts only for a very short time or is a prerequisite for some other role it may be best modelled as an event in the ELH of the subsequent role.

5.10. CONCLUSIONS

The concept of Role has been introduced. A role is a temporal subtype which is nearly always associated with a relationship in a traditional E-R model. A

diagramming convention for Roles and Object Role Hierarchies has been introduced. This produces a considerable simplification in the modelling of behaviour. The combination of ELHs and Object Role diagrams has been promoted for the description of behaviour and the inheritance of behaviour has been discussed. The result is a simple more rigorous modelling of behaviour in a completely Object Orientated framework.

Chapter 6

Relationships and other Associations

6.1. INTRODUCTION

In the Entity Relationship Model which has been discussed frequently in earlier chapters there are two main constructs: the Entity and the Relationship. In the development of the Object Role Model most innovations have tended to identify specialised relationships which reduce the generic nature of "relationship". There is a need to review what remains of the concept Relationship and how is it to be represented in the Object Role Model.

6.2. RELATIONSHIPS AND AGGREGATION

6.2.1 Relationships in the ER model

Objects in the real world interact with each other and any conceptual model needs to be able to model such interactions. Chen (1976) introduced relationships explicitly into his data model, this being the major advance over the relational data model in which such connections are implicit in a network of posted identifiers and tables. Chen says "A relationship is an association among entities. For instance, 'father-son' is a relationship between two 'person' entities." This sentence carries a footnote which highlights directly some of the major problems with the practical application of ER modelling. "It is possible that some people may view something (e.g. marriage) as an entity while other people may view it as a relationship. We think that this is a decision which has to be made by the enterprise administrator. He should define what are entities and what are relationships so that the distinction is suitable for his environment."

Chen goes on to define a relationship as a tuple of entities each taken from an entity set. He also uses the term role to describe the function of a particular entity in the relationship (Chen 1976).

Thus if

```
(A. Andrews, B. Brown)
```

is a marriage relationship where

```
A. Andrews, B. Brown ∈ {Adult People}
```

The role of A. Andrews is husband and that of B. Brown is wife. A relationship may have attributes e.g. a marriage has a start date. The tuple (A. Andrews, B. Brown, 1/1/95) could be a marriage or a father daughter relationship together with the daughter's birthday. The relationship needs to be named if its meaning is to be evident.

Chen identifies two types of entity: a *Regular entity* carries its own identifier whereas a *Weak entity* requires a relation to identify it.

The cardinality of relationships, that is how many entity instances can participate in each relationship, is important to any implementation as well as to the validation of the conceptual model.

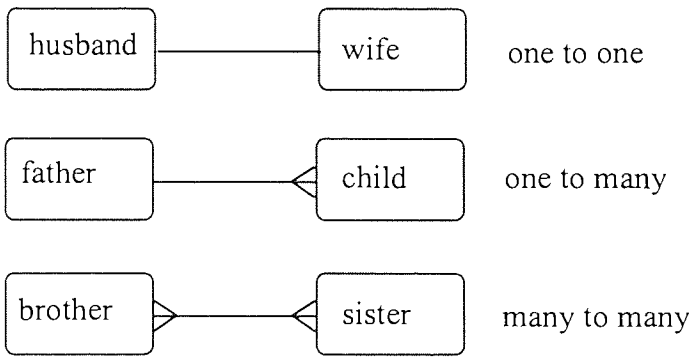

Figure 6. 1 : Cardinality of relationships with role names.

In figure 6.1 typical relationships are represented using the crows foot notation to represent the entity set which can contribute more than one instance to a single instance of a relationship.

The other important feature of relationships which needs to be brought out in any conceptual model is that of required membership. People can exist without being married or even without being siblings but they cannot exist without being a child of another person.

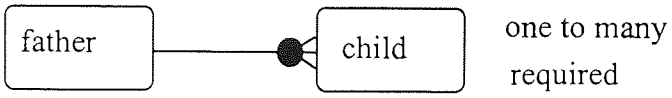

Figure 6. 2 : A required relationship

The example in figure 6.2 represents the fact that a person cannot exist without having had a father and this is shown by a black blob on the end of the relationship which is required. Many notations avoid a separate representation of required membership by placing lower bounds on the cardinality of the relationship. In the example in figure 6.3 a person can have zero or many (0:m) children but a child can have a minimum of two and a maximum of two parents (2:2). Other authors feel however that this is a qualitative constraint on the existence of an instance of an entity which requires distinct representation.

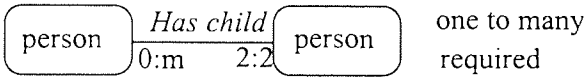

Figure 6. 3 : An alternative representation

Although it is possible when looking at a particular relationship to identify the semantics of the relationship by specifying the roles of the participating entities, it is more usual to indicate the entity sets from which each role is drawn and to give the name of the relationship on the link itself. Whilst relationships are not usually reflexive and a separate phrase is needed to indicate parent of, it is not usually necessary to show both directions on the diagram which would make it very cluttered as in figure 6.4

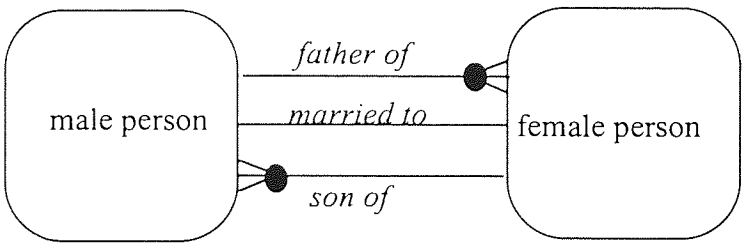

Figure 6. 4 : Several different relationships

6.2.2 The Binary Relationship model

A particular case of relationship modelling appears in the form of the binary

relationship model (Frost 1982) in which all information is modelled by relationships of different types between two objects. For this to be effective there is a need to raise to the status of objects, things which might normally be considered to be relationships.

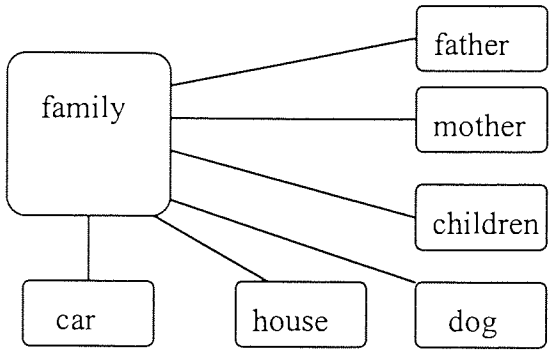

Figure 6. 5 : Binary relationship modelling

The problem of the binary relationship model is that it confuses implementation flexibility with the needs of conceptual modelling as illustrated in figure 6.5. To attempt to reduce communication to simple subject - verb - object sentences may be interesting but it makes describing reality very difficult.

6.2.3 Instance Connections (OOA)

Whereas in ER modelling and most conceptual models derived from it, a relationship is an important high level concept, in OOA (Coad 1991) one has to search to discover discussion of relationships. This is because the effect of creating specialist abstractions of specialisation and aggregation is to remove many of the fundamental relationships from the model. The relationships removed are in general very significant from an integrity maintenance point of view and are of great concern to implementers. The residual relationships are in general optional and do not carry inheritance or integrity responsibilities and hence can be treated more casually.

OOA identifies *Instance Connections* within the "defining attributes" chapter. To quote "Attributes depict Object state. Instance Connections add to that information, with required mappings needed by an Object to Fulfil its

responsibilities. Instance connections model association". The definition of *Instance Connections* in OOA is: "An Instance Connection is a model of problem domain mapping(s) that one Object needs with other Objects, in order to fulfil its responsibilities."

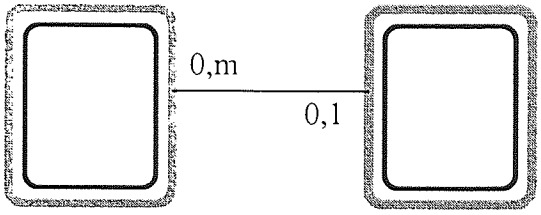

Figure 6. 6 : Instance connections in OOA

Note ,as shown in figure 6.6, that in the notation an Instance Connection exists between two Instances and not classes.

6.2.4 Links and Associations (OMT)

In OMT (Rumbaugh 1991) a link is a physical or conceptual connection between object instances. A link is an instance of an association. An association describes a group of links with common structure and common semantics. An association describes a set of potential links in the same way that a class describes a set of potential objects. Figure 6.7 illustrates links and associations in OMT.

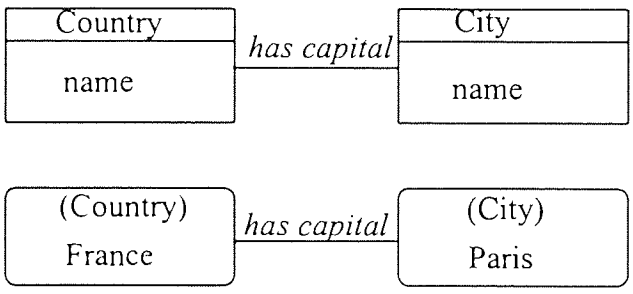

Figure 6. 7 : Links and associations in OMT

It is possible to represent "0 or more" and "0 or 1" by special symbols otherwise cardinality is represented explicitly "1 +".

Rumbaugh uses the term *Role* as did Chen (1976) to indicate the meaning of an object's participation in a link. This is shown in figure 6.8.

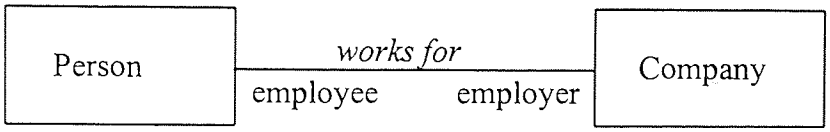

Figure 6. 8 : Association showing Roles

6.2.5 Aggregation

As has been seen, Smith and Smith (1977a) introduced aggregation as a modelling abstraction with a very general semantics. A major example of their aggregation is the record structure or the composition of an object by attributes. However aggregations between objects have yet to be discussed.

6.2.5.1 Aggregation association (OMT)

Rumbaugh (1991) also introduces aggregation (in the *Part_Of* sense) as a special case of an association but with richer semantics. In general if A is *Part_Of* B and B is *Part_Of* C then A is *Part_Of* C which will not be true of a typical relationship. In figure 6.9, for example, a chapter is part of a book. He also discusses the propagation of actions: thus deleting an object will typically delete those objects which are part of it, which is not true of most associations. In fact Rumbaugh suggests that the consideration of propagation is a tool for identifying aggregations.

Figure 6. 9 : A *Part_Of* association (OMT)

6.2.5.2 Whole Part Structure (OOA)

In OOA part of the process of identifying structures connecting objects is to consider Whole - Part Structures. Coad (1991) identifies three main variations in the Whole Part Structure:

- Assembly-Parts
- Container-Contents
- Collection-Members

as illustrated in figure 6.10.

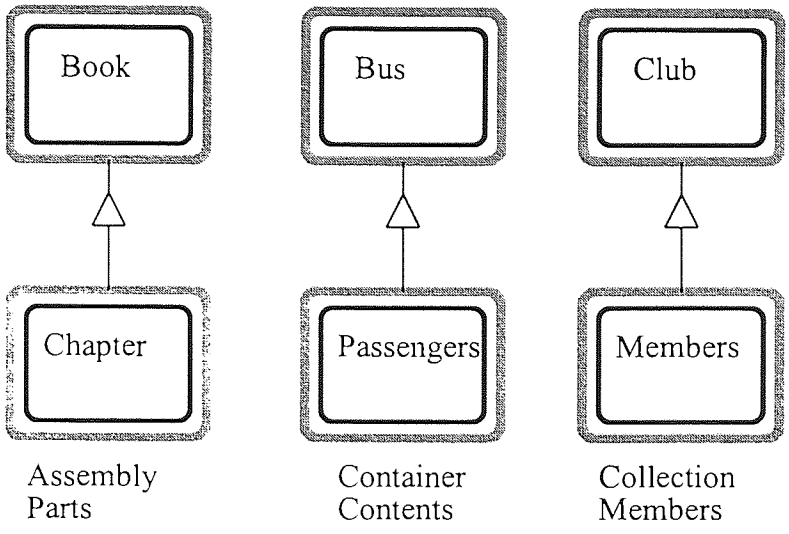

Figure 6. 10 : Whole Part Structures

Note that the Whole Part Structure is a association between instances and not classes. Although all three types of Whole-Part Structure have some similarity, that is a connection between an instance of one object class and several instances of another object class, the semantics are rather different as can be demonstrated by considering a deletion operation on the Whole. If a book is deleted are its chapters also deleted? - probably. If a bus is deleted then its passengers will be deleted - but not the people who are the passengers. This issue will be explored later; however here the suggestion is made that Whole - Part should probably be reserved for structures where the composition is permanent once created and not where it is essentially ephemeral as in Bus-Passengers; this latter certainly needs some attributes of dynamic modelling to adequately reflect the joint behaviour.

The modelling of the connection between attributes and objects as an aggregation is clearly an implementation oriented view of the problem and not a conceptual modelling one. To suggest that an Employee is an aggregation of attributes Age, Gender, Address etc. is to confuse the *Employee* as the intension of the Class Employee with the construct *Employee Record* which is the data object holding information about employees in the information system. This may be seen as an aggregation of fields but an *Employee* has many properties, a few of which are identified as attributes but it is not *composed* of those attributes.

6.2.6 Conclusion

In constructing data models the abstractions of specialisation and roles will not satisfy all the relationships to be found in an E-R model. In particular there remains the need for relationships at the instance - instance level. Such relationships represent a semantic connection between instances. Where there is no general semantics attributable to the relationship then the modeller and eventually the developer will have to provide the necessary semantic information and code to realise the relationship. A type of relationship which has been identified but for which the semantic implications are rather vague is the Whole - Part or the aggregation of objects.

6.3. ATTRIBUTES OR RELATIONSHIPS?

It is common for the analyst to have to exercise judgement as to whether a particular connection is a relationship or an attribute. An attribute is a property of an object which is variable between objects of the same class. However a relationship establishes a connection between two entities and may carry additional information such as the cardinality and optionality of the relationship.

From one point of view, age, position and department can all be seen as attributes of employee. However the modelling will usually distinguish between attributes and related entities. This distinction is not always evident and depending on circumstances different approaches would be equally valid. Current practice is heavily influenced by implementation issues, and the guidance in standard methodologies displays this pragmatic approach. The result is that a practitioner's E-R model is not a conceptual model of the real world in some abstract way, but rather a conceptual representation of the likely implementation, as illustrated in figure 6.11.

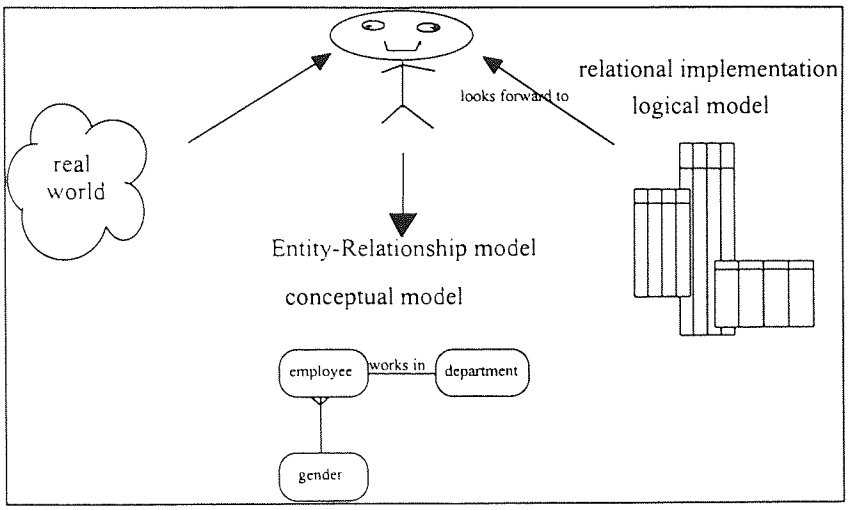

Figure 6. 11 : Conceptual modelling of the implementation

In the example shown in figure 6.12, a characteristic such as gender which would nearly always be treated as an attribute may for some purpose associated with the use of the data be raised to the status of an entity.

SSADM recognises this by distinguishing between classification entities and other parent child entities.

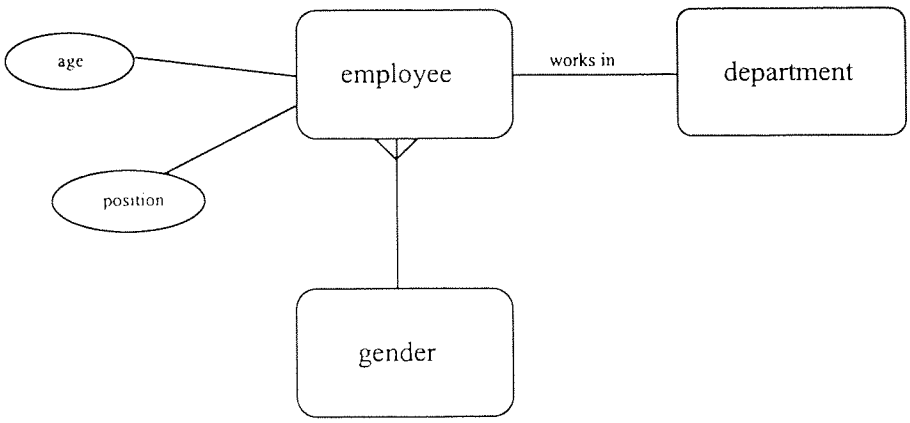

Figure 6. 12 : E-R model with a classification entity

The consequence for the understanding of elementary data is that most current modelling approaches are not semantically but rather implementation based.

6.3.1 Encapsulation and attributes

An abstraction central to the object oriented approach is that of encapsulation. The role of a data element cannot be identified by examining the representation of the element itself. *Brown* may be a colour and as such a recognisable attribute or it may be an identifier of an instance of person. There is a need to explore the nature of the objects under discussion as represented by a variety of conceptual modelling tools, and by the semantic structure of those statements referring to them. There are no easy solutions as to whether a concept is an object or an attribute and hence to understanding the nature of the relationship between two concepts. It is possible to propose some tests which may help in this identification.

6.3.2 Destruction

If a concept is destroyed in the physical world, or deleted from the information system, what is destroyed with it and what of the collection of concepts is now not within the Domain of Discourse?

Thus if the car modelled in figures 6.13 and 6.14 is destroyed, its colour *red* will disappear with it under most DoDs of interest, however its manufacturer *FORD* would probably continue to exist, but is it of interest? The conflict arises that although *FORD* looks like an attribute it will not cease to be of interest just because one Ford car is no longer within the DoD.

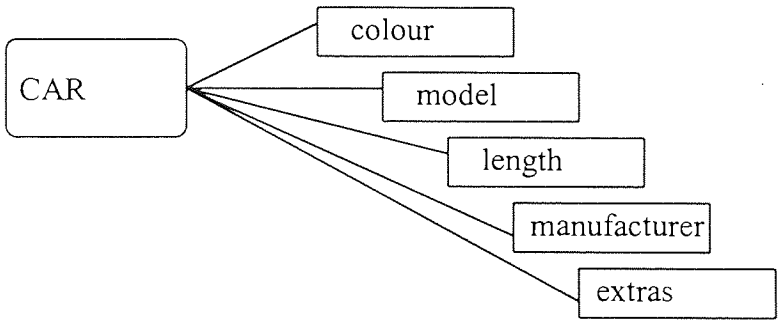

Figure 6. 13 : Description of a class

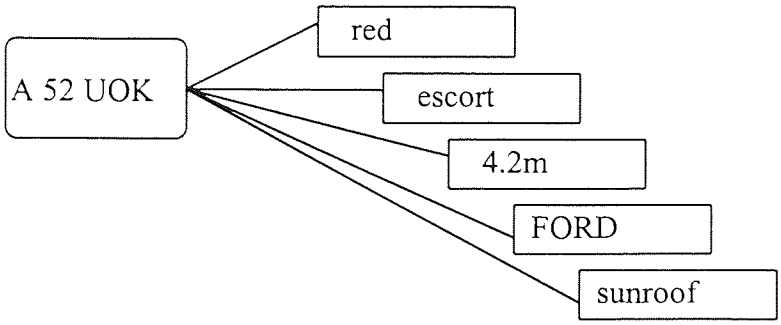

Figure 6. 14 : An instance of the class in figure 6.13

6.3.3 Destruction of an entire class of objects

If all cars made by Ford ceased to be within the Domain of Discourse (perhaps there is only an interest in servicing garden mowers) then would the manufacturer *FORD* remain of interest - probably not? In this case there is an example of an independent object which has conditional existence within the DoD.

It would appear that the problem with the distinction between attributes and instance connections cannot be resolved at a philosophical level (Eckert 1993) because it is not the property of the object possessing the attribute or relationship, but is embedded in the DoD. (See figure 6.15.)

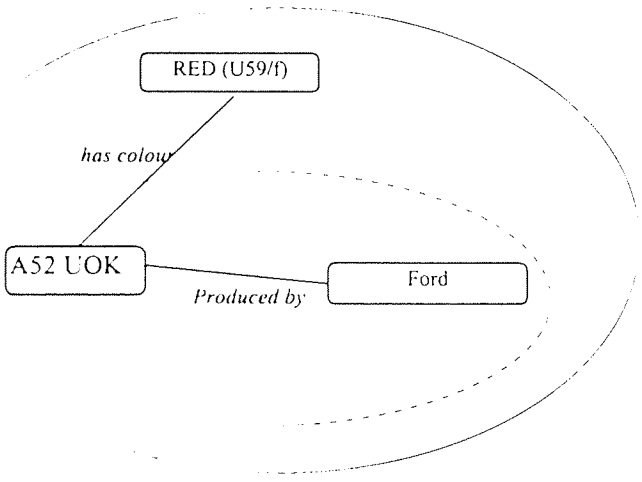

Figure 6. 15 : Two different DoDs

A car owner with a dented panel would be concerned about the colour of their car. Red is a colour and hence a value not an object but because there are many different shades of red it may be necessary to cite the batch number to identify the

particular shade of red which is required. However for the paint manufacturer, the batch is clearly an object: it is an instance of a class *Sunburst*. It would appear to violate principles of encapsulation if the existence of the manufacturer in the DoD changed the perception of the attribute. However it is clear that in the wider DoD the colour is an object and the attribute an instance connection. How can this be tested? It would appear that the existence of at least one attribute which is an instance connection is necessary for the class to be seen as an object class and not as a domain.

6.3.4 Conclusions

The distinction between values of domains and objects is not clear cut; hence it is not obvious whether a particular aspect of a model is best represented by an attribute with a domain or by a relationship with objects representing the elements in the domain. This is not only true in the modelling level but also in implementation. In many cases both will be represented by a logical collection which is itself implemented by a file or a table. If the sub system being discussed has methods for the insertion and deletion of objects into the collections then one is clearly dealing with objects. If such update functions are outside the DoD or absent altogether then one is looking at domains.

6.4. DOMAINS

The brief discussion of domains developed in Chapter 2 reflected the concerns of data processing and traditional database technology. However in developing the object data model to underlie a statistical summary model it became clear that this analysis was inadequate.

The fuzziness over treatment of domains results from implementation concerns. A binary relationship exists at two levels, that of the class and that of the instance.

Thus the general relationship:

```
Class has an attribute which maps to a domain
```

which is instantiated for a particular class e.g.

```
Employee has attribute Age with domain Integer
```

which can itself be instantiated for a particular instance as:

```
John has an Age of 20.
```

However the following is also a typical domain reference:

```
Employee works_for Department
John works_for Accounts
```

Users may not distinguish between these two cases. Nevertheless, it is possible to pose questions about the department for which an employee works. What is the average age of staff working for the department for which John works? It does not make sense to pose such a question about *20.*

Although the semantic differences are clear, developers working with a Object Oriented or relational technology will treat them identically.

It is important for the development of the Object Model that the different types of domain that may be encountered are considered in some detail.

6.4.1 Elementary domains

6.4.1.1 Primitive domains

Domains can also be classified by the nature of the process involved in validating that a value belongs to a given domain. For a number of special primitives (integer, real, string, date) an algorithm is required to examine the encoding and consider whether it is acceptable. For all other domains a check is made against a list to see of the value is present or not.

Most modelling systems and database management systems include a list of primitive domains to which it is not possible to add and delete members. Often these are incorrectly related to storage structures so `Int, Long, Double` become primitive types.

6.4.1.2 Object Domains

A common type of domain is one in which there is a reference to an instance of a specified class.

Thus the *Department* John *works_in* has to be a member of the class *Department*.

Many writers do not distinguish between the class and its members and this leads to confusion in explicitly defining domains. Following Wieringa (1991) it is appropriate to distinguish the intension of the class, the extension of the class and the existence set of class members. The domain value must be a reference to a member of the existence set. In many cases there may be a restriction placed on domain values e.g. "Employee with over two years service". Eckert (1993) argues that it is not appropriate to consider this a subclass of Employees because its membership is changing and does not depend on constant attributes of an Employee, rather this should be considered to be a collection of Employees constructed according to a given predicate. This leads to the proposal that the domain of any such attribute is in fact a collection or sub collection of the members of the specified class. In general there is interest mainly in collections which are the root collection for each class, i.e. the existence set of class members. Other collections are defined by placing selection rules on the membership of the existence set.

Thus when it is said that the domain of *works_for* is *Department* it means that the domain is the base collection of the class *Department* which contains all extant *Departments*.

6.4.1.3 Enumerations and Lists

The domain of many attributes is one member selected for a list of possible items. Thus Gender has the domain {Male, Female}. This cannot be considered to be a pointer to a member of a class because there is no natural class which has the two members *Male* and *Female*. As an implementation this might be treated as a class, creating a table *Gender* with the two elements *Male* and *Female* and even

including mappings from internal codes such as *m* and *f*. Such implementation issues distract from the fact that *Male* and *Female* have a meaning and are attributes applicable to instances of certain classes, not members of a class.

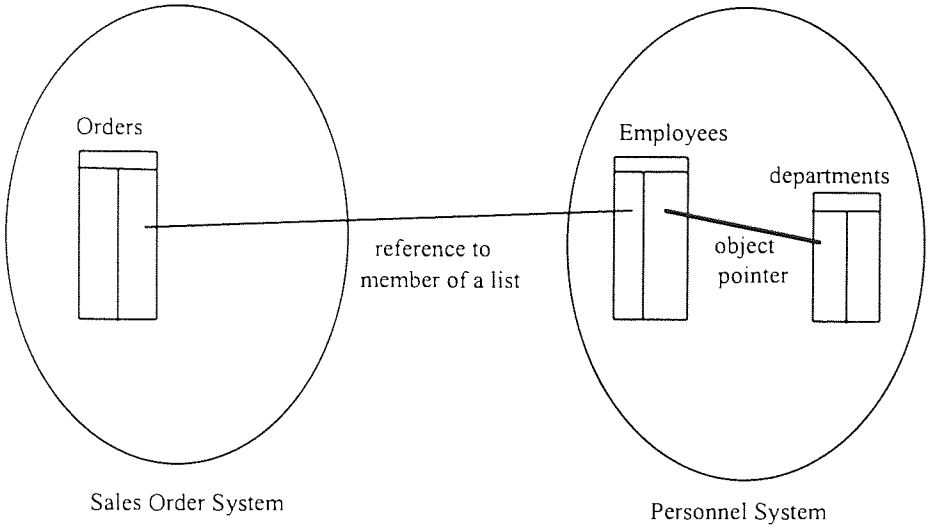

Figure 6. 16 : References and domains

It has been observed that whether the domain of a variable is seen as an object rather than an enumeration depends on the relevance of the object class to the Universe of Discourse. Thus for a sales order system (figure 6.16) the domain of the attribute *Salesman accepting Order* is the name of a salesman who at that time worked for the company. However the personnel management system would treat salesman as an instance of the class Employee.

6.4.2 Complex Domains

It is also necessary to allow for complex domains created by the Cartesian product of two or more other domain elements.

The classic example is an address

```
Address{
     Road {String}
     Town {String}
     County {String}
     }
```

The treatment of a complex domain like address is obscured by implementation

and data processing conventions. A classic implementation would be to create a class of addresses the attributes of which are Road, Town, County and then we could use the object_id of the address as a reference. This is a fix because Address is not a class of objects in the DoD. It might be in a Local Authority Planning department, or in the Council Tax office but not where it is the domain of Employee Lives_at.

It is thus necessary to permit domains which are aggregations of elements selected from other domains. How these domains are referred to and how values are assigned will depend on the syntax of the query language.

For example

```
person_id.Address
```

should return a structure (in C terms).

```
Person_id.Address.Road
```

should return the appropriate element. Although, depending on the scope of labelling of attributes so might

```
Person_id.Road
```

6.4.3 Domain Hierarchies

A domain may be derived by collecting elements of another domain. Thus the West Midlands Region is defined as the counties of Warwickshire, Worcestershire...etc. Many industrial and administrative units are described by hierarchical classifications.

Such a classification maps two or more elements of one domain to a single element of another. There is no change in the scope of the domain and any level in the hierarchy is a valid value for a domain reference.

Thus given the specification,

```
Customer has a location drawn from Geographic_Classification
```

the value could logically be "Aston Triangle", Birmingham, West Midlands, England, UK. Obviously in a given application consistent data will be collected by using one level of the classification e.g. customers may be recorded at the Post Code level.

A classification domain is itself a mapping onto another domain in the same dimension. The consistency of dimension over the classification is implicit in the location example, and suggests that classification domains are conceptually the same as the change of units of measurement within a given scale e.g. centimetres to meters or even centimetres to inches. The target domain could be any of the other domain types discussed.

6.5. CONCLUSION

Despite the relative paucity of discussion of domains in the literature it is clear that domains constitute a distinct and complex area of modelling. In developing the object model it is clear that there are many types of domain.

A broad classification is between domains whose elements are defined outside of the DoD and those domains whose elements are defined within the DoD. In the former category is found such primitive domains as Integers, Reals, Character strings, Dates and many enumerations, Days of the week, Countries etc. In the second category is found Part Nos., Customers, Employees.

In both cases there is an encoding which maps the real concept to a sequence of symbols stored in the information system. Such encodings may be formalised in classifications (such as the SIC) or by standard usage (Mon., Tu. ...) or be explicitly defined within the IS such as object-identifiers.

Any Object Model needs to take explicit account of domains and needs to provide methods of defining creating and referencing domains. The integration of domains into a single model and their role in the development of the statistical summary table model will be the focus of attention of later chapters.

Chapter 7

The CORD Model

7.1. INTRODUCTION

As the previous chapters have shown, there has been a very large amount of research aimed at developing appropriate conceptual modelling frameworks; it is thus with reluctance that extensions to existing models have been proposed. However, as is also evident from the variety of offerings, there is a lack of agreement as to the most appropriate features of a conceptual model and it may be argued that a range of conceptual models is required because of the different purposes to which they are to be put. In this research the aim of creating a conceptual model which will interface with the specification of Management Information Systems such as Executive Information Systems is sufficient to require specific attention to two areas where traditional conceptual models have been shown to be weak.

Firstly, because the main aim of management reporting of a process is the monitoring of change, it is considered that the modelling of the dynamics of a system is important and that the tools suggested in Chapter 4 are not sufficiently user friendly to be practical. The model proposed in this chapter will be based on the contribution to modelling of object behaviour developed in Chapter 5.

Secondly, because it is important to consider the nature of data to be summarised it is necessary to address the issues of domain definition in a rigorous manner. The traditional treatment of domains by current conceptual models is inadequate, so the proposed model will build on the domain definition developed in Chapter 6.

The basic object model discussed in Chapter 3 will be augmented with a more rigorous treatment of domain definition and dynamics of objects. Little of what is proposed is completely new and emerges out of the recent research (Wieringa 1991) as well as traditional models (Kent 1986). However it can be argued that taken together this constitutes a new approach to conceptual modelling.

The model proposed in this chapter is the CORD model named after its main

elements: Collections, Objects, Roles and Domains.

7.2. IMPORTANT ELEMENTS IN THE MODELLING SCHEMA

It is necessary to develop a schema to represent instances of the CORD model. Since a central theme of this work is that the schema should be consulted in the definition and execution of EIS applications, then the schema will serve not only as the external definition of the model but has been developed to be readable by object management applications.

In the discussion that follows the various elements of CORD are introduced in a systematic way, but, as many definitions are recursive, some early reference to concepts elaborated in more detail later is unavoidable.

7.2.1 Instantiation

Any object in the CORD model can be viewed as a class and instances of it created. Thus the object has two sets of properties: those which it possesses as an instance of a super-class and those which are defined for its instances. Every object has a unique identifier (character string) generated for it. An object can also be known either by one of its attributes or by a system-given name. Normal practice is to ensure that this name is unique within the class; thus any object is identifiable by a path created by cascading object's names.

```
A.B.C;
```

This implies that C is an instance of B which is an instance of A. For example

```
Employee.Andrew;
```

states that Andrew is an instance of Employee. If Andrew does not exist then an instance of Employee will be created with the name Andrew. The dot notation is conventional but does not deal conveniently with lists of instances. Thus to say that B, C and D are all instances of A one would have to write

```
A.B ; A.C ; A.D ;
```

An alternative is available using "[]" (square brackets) to denote a group of instances.

```
A[B; C; D;]
Employee [Andrew; Bill; Colin;]
```

This specifies that B, C and D are instances of A and that Andrew, Bill and Colin are instances of Employee. In the schema an object which is an instance of a class has a IsInstanceOf property with respect to its class. This is just one of many properties of objects which are usually set at creation time by the schema but which could be set by assignment. Thus the dot or square bracket notation is (approximately) equivalent to

```
Object.Andrew (IsInstanceOf: Employee;)
```

As all objects are equal it is only the analyst who knows whether it is justifiable to create an instance of a class. For example there may be a Domain of Discourse for which the following conveys valid information

```
Person.Andrew.Anthony
```

7.2.2 Ownership

A fundamental concept in the CORD model is that of *ownership*. Ownership is a cascade delete property which holds for a number of relationships within the CORD model.

Thus attributes are *OwnedBy* objects, actions are *OwnedBy* objects. Instances are *OwnedBy* their class and roles are *OwnedBy* the role-player. The *OwnedBy* concept is very close to the Container-Contents Whole Part association of OMT (Rumbaugh 1991). Where the *OwnedBy* property holds, concepts are usually required to have unique names within their ownership group rather than their class. This corresponds to normal real world practice: dogs typically have unique names (concurrently) within a family but only registered pedigree dogs would have unique names within the breed. A convenient consequence is that this

enables many objects to have attributes with the same name.

This makes the path identification of an object less certain, thus:

```
Attribute.Age
```

will probably not be unique as many different object classes may have an attribute called Age. To represent that one or more objects are *OwnedBy* another they are enclosed in "{}" (curly brackets) after the Owner. So, in the schema for the CORD model which follows ownership is shown by {}.

Thus

```
A {B}
```

implies that the object B is owned by object A. If B has no distinct object class specified then it will be assumed to be an instance of A; if a class is specified then B will be an instance of that class C which is *OwnedBy* object A.

```
A { C.B;}

Employee {Attribute.Age; }
```

This means that Age is an instance of the Attribute class which is *OwnedBy* the class Employee. Or in other words Age is an attribute of Employee. As with instantiation there is a property of an object which corresponds to the *OwnedBy* relationship and the previous expression could be written

```
Object.Attribute.Age (Owner: Employee;)
```

This general concept of ownership was introduced to standardise the handling of a number of dependent relationships in a semantically natural way.

7.3. OBJECTS AND OBJECT CLASSES

A class is shown as a hard labelled box with or without a heading panel. Unlike OMT there is no need to specifically represent instances. A conceptual model is about intention and this is adequately represented by the class symbol (figure 7.1). There is no statement implicit in the CORD model about the storage of instance

data corresponding to classes.

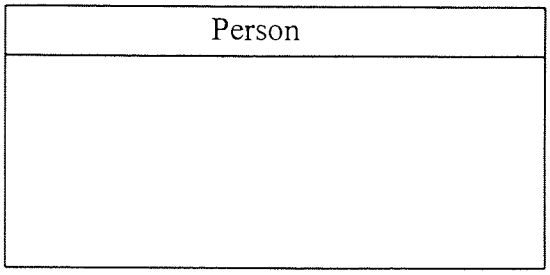

Figure 7. 1 : An Object Class

7.3.1 Generalisation and Specialisation

In any object model there will be a need to specify an inheritance relationship between two classes. In CORD this is implemented with the subtype property.

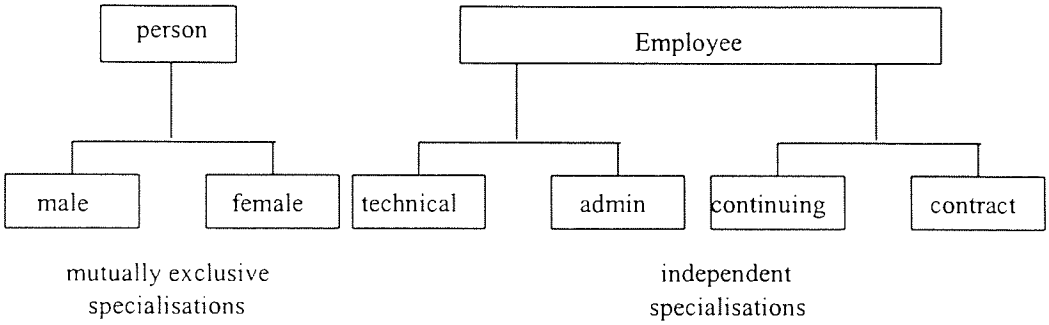

Figure 7. 2 : Specialisations

Specialisations can be shown graphically by trees (figure 7.2). However in the schema, as with other object models, this is stated explicitly. No special notation or function has been provided to distinguish specialisations from categories.

In the schema an object which is a specialisation has an *IsASubtypeOf* property.

```
OBJECT.Male (IsASubtypeOf: Person;)
```

7.3.2 Attributes

An attribute of an object is a function which returns values from a specified domain for instances of a type of the object. Rather than clutter the graphical model with the representation of attributes they will be listed in the CORD model

schema.

An attribute may have several characteristics which are represented here in the (partial) schema definition for attribute:

```
OBJECT.Attribute
{
 Attribute.Domain;
 Attribute.MaxCardinality(Default: 1);
 Attribute.MinCardinality(Default: 0);
 Attribute.Default;
}
```

The most significant element of an attribute definition is its Domain; the domain of the domain attribute of an attribute could be almost any collection of objects in the DoD or one of a few predefined primitive domains. The definition of domains is dealt with later in this chapter.

The cardinalities determine how many values from the domain can be associated with the object through this attribute. If the value of an attribute would return NULL and a Default is specified then the Default is returned instead. Thus the default for an attribute is that it may have one or zero values. If an attribute would have been defined in SQL as non null then the MinCardinality is 1.

An attribute can have a sequence order. This was introduced to enable traditional record structures to be used. Thus instead of referring to:

```
Person.Andrew (Name: Andy; Age: 25; Gender: Male)
```

it would be possible to write

```
Person.Andrew(Andy, 25, Male)
```

This can only be done if there is an implicit ordering of the attributes of a class. In many implementations, as attributes map to columns in a table, there is an automatic implicit ordering; however this is clearly a feature of a particular implementation which is not automatically available in a semantic data model.

```
Attribute(IsOrderedBy: SysOrder;)
```

When members of a class have the IsOrderedBy property set then that ordering will be applied by default on retrieval and as new instances are added the order attribute is automatically incremented. SysOrder is a default system property which is used to record the order, in this case, of attributes.

An object has those attributes defined for its class and all supertypes of its class. Where an object inherits a name more than once in the class hierarchy it is the first (lowest) occurrence which defines the attribute. This is not a case of refining the higher definition but possibly substituting a completely different attribute i.e. overloading. The existence of multiple inheritance is not precluded from the model and is specifically dealt with in the definition of Roles but the model includes no statement about the resolution of name clashes in the inheritance hierarchy.

7.3.2.1 Properties

In discussing the definition of classes and objects reference has been made to special types of attributes called Properties. As the CORD model is intended to be very flexible and with the minimum of predetermined structures a mechanism was required for defining meta-level properties. Thus attributes, domains and collections are defined within the model. Properties are used to define the behaviour of classes and objects and are to be recognised by high level application software operating at the ODBMS level, for example. End user applications on the other hand will typically only access objects and attributes of user defined classes.

A complete list of Properties follows.

```
IsInstanceOf
IsKnownBy
IsOrderedBy
IsPartOf
IsRoleOf
Owner
PartOf
RoleOf
Scope
```

```
SubType
SubSet
SysName
SysOrder
SysStates
```

Many of these go together in pairs. Thus IsKnownBy sets the attribute of instances of a class which will be used for identifying objects. By default IsKnownBy refers to the special property SysName which holds the names of objects which do not specify an attribute to identify them. SysOrder has a similar relationship with IsOrderedBy except the default is to have no ordering.

Because of their system role Properties have to have global names and cannot be overwritten so IsInstanceOf cannot be redefined no matter what object it is applied to. The Property Scope which can take the value "Global" is used to specify when naming is global to the class and not unique within ownership groups. In general where it is necessary to specify the domain of a Property this will be valid wherever it is used. However RoleOf and PartOf need to behave differently. Where Employee is a role played by a Person it will be necessary to be able to create instances of Employee which are roles played by instances from the domain of all Persons. Thus the IsRoleOf property of a class defines the domain applicable to the RoleOf attribute of members of the class. A similar relationship applies to IsPartOf and PartOf.

```
Employee(IsRoleOf: Person; IsPartOf: Company;)

Employee [Col(RoleOf: Colin; PartOf: AlphaCo;)
          Elly(RoleOf: Ellen; PartOf: BetaCo;)]
```

The above schema fragment defines the class Employee as being a role played by people (IsRoleOf : Person) and hence instantiates Col and Elly as roles played by Colin and Ellen respectively. To be accepted as a valid definition, Colin and Ellen would have to be already defined as being People.

7.3.2.2 Assignment

Instances of a object can have values for their attributes.

```
Object.Employee
{
 IsASubtypeOf: Person;
 Attribute.Salary (domain: Number;)
}
```

Note that the semicolon is used as a delimiter. The above means that Employee is an instance of Object and a subtype of Person. Instances of Employee have an attribute called Salary which has a domain of Number.

Objects have properties or attributes which are determined by their class. These attributes have values and values are assigned by the assignment operator: (colon). The relationship between attributes and the instance of which they are attributes is represented by normal parentheses "()". This enables an intuitive representation of record structures to be included in the schema.

Thus

```
Object.Employee
{
 IsASubtypeOf: Person;
 Attribute.Salary (domain: Number;)
}
Employee.Brown (Age: 23; Gender: Female; Salary: 1249)
```

states that instances of the class Employee can have an attribute Salary, and that Brown is an instance of Employee. Brown has inherited attributes Age and Gender from Person and has the value 23 for its Age, a Gender of Female and a Salary of 1249. Age, Gender, and Salary are sufficiently identified because the model in general expects a token which is either an attribute of Employee or the value of an attribute of Brown. As the token is assigned to (followed by:) then it is expecting an attribute of the Class of which Brown is an instance. The conventional "," (comma) separator for elements of a record has not been used because that is reserved for separating elements of lists. If employee Brown has several children they could be assigned as

```
Employee.Brown (Child: Sally, Bill, Zoë ;)
```

Thus in the CORD schema ";" (semicolon) separates attributes and "," (comma)

separates list values.

OwnedBy is just one of many properties which can be set by assignment:

```
Attribute.Salary (OwnedBy: Employee;)
```

This sets the value of the Attribute OwnedBy for the instance of Attribute called Salary to be Employee.

7.4. COLLECTIONS

The first elements in the name of the CORD model are Collections. Collections have no major role in the conceptual model because they are essentially a tool for managing data. However in the discussion of domains in Chapter 6 the point was made that domains are often collections.

Collections are generally of three types, defined in two ways:

Explicitly defined collections contain lists of the ObjectIds of their members. *Implicitly* defined collections consist of queries specified against a class of objects.

Homogeneous collections contain instances of a single class. In the implementation there is an implicit collection associated with each object class which is the collection of instances of the object. In the ODMG (1993) definition this is taken further and the existence of such a collection (called an extent) is used to specify the location of the data associated with a class and distinguishes abstract from concrete classes. As CORD is a purely conceptual model no specification for storage is included.

Hierarchical collections will contain references to instances from the same IsA hierarchy.

Heterogeneous collections can contain references to objects from many distinct classes.

When a collection is deleted the objects referenced are unaffected so the relationship between a collection and its members is a Part Of one.

In formulating the CORD model it was contemplated that some domain elements were defined as collections but this was rejected in favour of treating domains as high level distinctive objects. Collections are used to map members of classes to domains where the classes are otherwise relevant to the DoD. Where a class would solely exist to support a domain definition and where a collection would have been used to collect instances of that class a domain object is created instead.

7.4.1 Subset

It may be necessary to specify restrictions on domains or on collections. For example adults may be Persons whose age is greater than 18, or a sub collection may be those Employees who have worked for the company more that 5 years. A collection may be defined as a subset of another collection or a domain as a subset of a domain.

7.5. ROLES

An object may play a role:

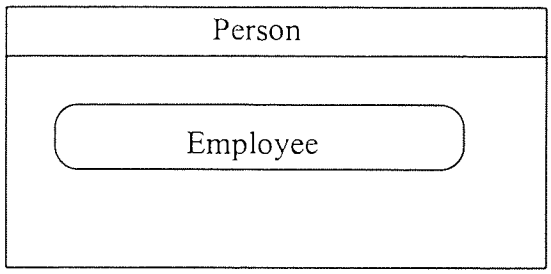

Figure 7. 3 : Role

shown in figure 7.3 and which would appear in the schema as:

```
OBJECT.Employee (RoleOf: Person;)
```

Given the definition of role from Chapter 6, a Role will usually have two parents and this has led to some problems in the specification of roles in the CORD

schema.

```
Employee(IsRoleOf: Person; IsPartOf: Company;)
       { Attribute.Department (Department;)
         Attribute.Children (Employee;6;0;)
         Attribute.JobTitle(Job_Type;)}
```

Although an Employee inherits attributes by being a Person they also have characteristics due to the Company they work for. However the relationship with Company is not an inheritance one but rather a question of belonging to; the usual phrase *OwnedBy* is appropriate not to the employee per se but to the employment. It would not be appropriate to inherit characteristics of the Company in the same way as characteristics of a Person can be inherited by an employee. Thus if the age of an employee was requested the expected information that they were 25 years old would be acceptable, to be told that the age of their employment with the company was 5 years would be surprising, but to be told that the company was 100 years old would seem bizarre. To resolve this a role has two different types of parent: where inheritance is logical in that an employee IsA person then IsRoleOf is used, otherwise IsPartOf specifies the participation in a role.

7.6. DOMAINS

The definition of a domain has been discussed above and is:

```
domain = simple domain | complex domain
simple domain
     =    (primitive domain | collection | list)
          primitive domain = (String | Numeric | Date)
          collection =
               (class [, selection rule]) | {object_id}
          list = {String} | {Numeric} | {Date}
complex domain
     =    (domain, domain...)
```

When an attribute has a value assigned to it the way the character string representing the value is interpreted depends upon the domain of the attribute.

```
Employee
{
 Attribute.Address {domain: String;}
 Attribute.Department {domain: Department;}
```

```
}

Employee.Brown
(
 Address: 12 High Street;
 Department: Accounts;
)
```

In assigning a value to Department a value consistent with the domain called Department will be expected. If Department is a class, a list or a collection, then the ObjectId of the member of the class (list or collection) with the name "Accounts" will be assigned to the attribute Department. If no such member could be found then an error occurs. If the attribute Department had a domain of type String then the string "Accounts" would be assigned.

The structure of the domains within the model is given by:

```
Domain{attribute.Class(Concept.Class;);
       attribute.Type(Domain_type;)}
       [Integer(Type: Primitive;)
        String(Type: Primitive;)
        Blob(Type: Primitive;)
        Date(Type: Primitive;)
        Boolean(Type: Primitive;)
        Domain_type(Type: List;)
        [Primitive; List; Classification;
                  Aggregation; ClassList;]]
        Path(Type: Primitive;)
       ]
```

7.6.1 Complex Domains

The CORD model supports complex domain creation as well as complex links between objects. Complex domains are distinct from complex objects which are created by aggregations of objects.

In CORD, complex domains are constructed using two main constructors, neither of which are available under the relational model but are available in various forms under the object model.

7.6.1.1 Aggregations of domains

An attribute can have as its domain a record structure. Thus Address can have

several elements. It is still recognised as an element in its own right, so Address can be referenced and listed but the lower elements can also be referenced

```
Class.Person (IsKnownBy: Name;)
    {
      Attribute.Name (String;)
      Attribute.Age (Integer;)
      Attribute.Sex (Gender;)
      Attribute.Status (Marital_Status;)
      Attribute.Address (Address;)
    }


   Domain.Address(Type: Aggregation;);
   {
     Part.County (County;)
     Part.Postcode (String;)
     Part.Street (String;)
     Part.Town (String;)
   }
```

In this fragment of code, the attribute of Person called Address has as its domain Address which is a domain of type Aggregation. It is possible to refer to the Street of a particular Person as follows

```
John>Address>Street
```

In implementing domain aggregations there is no need to create a class or a table corresponding to Address as an Address is not a distinct object which can be referenced other than as an attribute of a class. This clarification that aggregations of domains remain as domains and are not some sort of second class object is a contribution of the CORD approach. It is an implementation issue as to how the instance data is stored. In the case of an underlying relational model a separate table complete with posted identifiers would be used; in the binary data model underlying the Object Prototyper (see next chapter) no distinct storage structure is required.

7.6.1.2 Multiple values

The second technique for creating complex objects is the relaxation of the requirement that attributes may take only single values.

As can be seen from the model, CORD supports attributes with a cardinality of 0 (optional), 1 (required) or more. The attributes of an attribute:

```
MaxCardinality(integer;)
MinCardinality(integer;)
```

can be set to enforce most common cardinality requirements. In defining an attribute of a class the constraints default to (0 : 1) i.e. a single valued optional attribute, but can be defined to be different at creation time or at any later stage.

```
Employee(IsRoleOf: Person; IsPartOf: Company;)
        { Attribute.Department (Department;)
          Attribute.Children (Person;6;0;)}
```

In this fragment an Employee is defined as having up to six Children each of which is a Person. This approach is distinct from the ODMG (1993) approach in which all multiple valued attributes are defined as sets, bags or lists of elements from the given domain.

```
Attribute.Children setof(Person;)
```

The CORD approach is seen as more natural in that users would not typically think "I have a set of children" and in particular "I have a set of one children" would seem strange. Where it is required to reference a set then a collection is the appropriate construct.

```
Object.CricketTeam
     {
       attribute Name (String;)
       attribute.Captain (Person;)
       attribute Players (Collection;)
     }
Object.Collection.OurTeam {John; Bill; Sidney;}
Object.CricketTeam.CatandFiddle
       (Name: "Cat And Fiddle";
        Captain: John;
        Team: OurTeam;)
```

Multiple values and aggregates can be combined in any way in defining the domains of attributes of objects.

7.7. OTHER RELATIONSHIPS

Two objects may be associated; that is, have some shared behaviour. In its most general sense, this relationship between entities is the fundamental construct of all object modelling. In a semantically rich environment such as CORD many relationships have been specialised, leaving only residual relationships between kernel entities to be modelled by the relationship concept. It has been argued earlier that the Role construct will be the appropriate one for many such relationships. However, as a discussion example, consider the desire to model the relationship between a borrower and a book in the modelling of a lending library.

There are two ways of representing this relationship within a schema which correspond to the various implementations of the same relationship in a relational data model - a distinct table for relationships or posted identifiers.

```
Object.Relationship.Borrows
{
 Attribute.Borrower (domain: Person;);
 Attribute.Book (domain: Book;);
}
```

Alternatively

```
Object.Person
{
 Attribute.Borrows (domain: Book;);
...
}

Object.Book
{
 Attribute.Borrower(domain: Person;);
...
}
```

Most of the literature is vague about the nature of relationships. Clearly in a conceptual model such as the E-R model, relationships are an independent type of object with connections to the parent objects. This is very much the situation in a relational implementation of a many-many relationship in which a table containing posted identifiers is created to effect the relationship and this table will also hold any other attributes of the relationship such as start date.

In any object model there is a problem which is connected with encapsulation and visibility. If a relationship is a separate class (figure 7.4) then can the parent classes view the attributes of the relationship, can they change or update values? These problems have not been addressed in the literature.

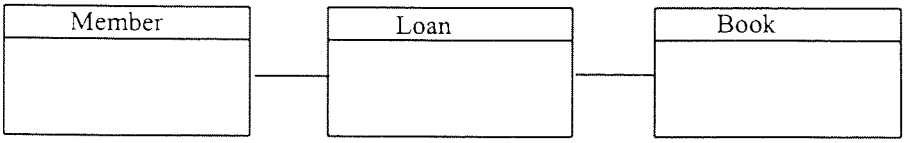

Figure 7. 4 : Distinct class model

An alternative model is that Loan is a hybrid object which inherits some characteristics of each parent yet is a different class of object hence "mule" (figure 7.5). This is distinct from what AlShawi (1991) called semantic convergence, whereby a new class of object is created which shares all characteristics of both parents and holds an ISA relationship with both.

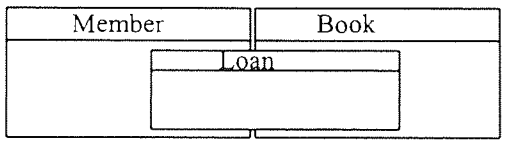

Figure 7. 5 : Mule model

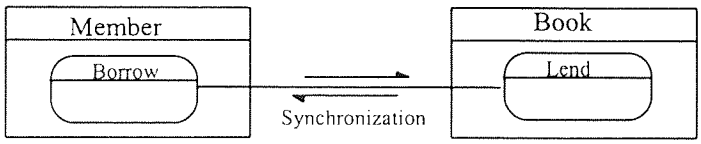

Figure 7. 6 : Synchronisation model

A third approach is to treat Loan as two distinct objects which are somehow kept synchronised (figure 7.6); this is the model underlying many transaction systems, the transaction being the unit of synchronisation. This may solve some issues of visibility and suggests implementation approaches but violates the instinct of the user: that the borrowing of a book and its lending are the same thing or in fact two different views of the same thing.

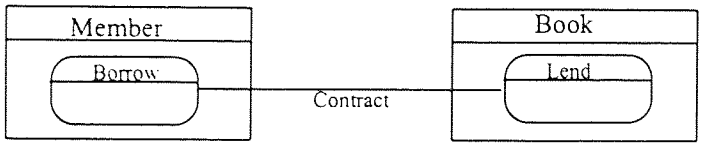

Figure 7. 7 : Partnership model

A fourth approach is that the activity represented by the relationship is an enterprise set up by the two parents regulated by a contract, thus there is only one object spawned jointly by the partners but this is governed by a contract which establishes the relationship of each partner to the joint enterprise (figure 7.7). This model seems to reflect the nature of a relationship and to have conceptual appeal. Thus in the CORD model it has been decided to choose to view a relationship as a single object which is a single role played by both partners. Issues of visibility and scope are governed by the contract but typically either partner can view the joint activity, the joint activity has no access to private details of each partner and either partner can effect changes in the joint activity. This will be referred to as the partnership model of relationships. Of course this is extensible to partnerships of three or more. This approach is consistent with the ODMG (1993) approach however as CORD is richer than the ODMG model fewer relationships will need to be treated as *Relationships*!

7.8. LIFE HISTORIES AND ACTIONS

Any object or role can have a life history. A life history is a set of actions:

```
Object.Employee
{
 action.Recruit {}
 action.Promote {}
 action.Discharge{}
}
```

An action has two parts, a condition which must be true before it can be performed and an executable part.

```
Object.Action
{
 attribute.Condition ;
 attribute.Do ;
}
```

When a object is instructed to perform an Action, if the Action is defined for that object, the Action described by Do is performed if the Condition specified is true.

Any object that suffers actions has a SysStatus which is the last action suffered.

Example

```
OBJECT.Action.Promote
{
 OwnedBy: Employee;
 Condition: if $today - StartDate > 2 ;
 Do: "Grade: Next(Grade);";
}
```

This states that Promote is an instance of Action which is OwnedBy the class Employee. If an instance of Employee is sent the message Promote then the Condition will be checked for that instance i.e. is today more than two years greater than the Employee's StartDate. If so the function Next is applied to the Employee's Grade and Grade takes the result. SysStatus will take the value *Promote.*

If Brown is an instance of Employee:

then

```
Brown > Promote;
```

will cause the object Brown to suffer the Action Promote.

The necessary language for specifying the Do parts of actions has not been developed. For the subsequent Statistical Summary Tables only the concept of SysStatus is really necessary.

7.8.1 Overloading

As discussed in section 5.8.1 different actions may have the same name this is a form of overloading of the name of the action. However an action is required to have a unique name within its owning class. Thus many classes could have an

action Delete defined for them. In obeying an action the class is checked to see if an action of that name is defined for it; if not each supertype is checked. If the action is not defined then it fails.

7.9. CORD

The Collections Objects, Roles and Domains model (CORD) was implemented as the base model using the Object Prototyper. This model featured a rich range of high level constructs. The main instantiation hierarchy of CORD is shown below, the detailed CORD schema is shown in Appendix 1

```
------------------------------------------------------------
-- Partial Model Cord
------------------------------------------------------------
  Model.Cord
  {
    Concept (Scope : global;)
    [
      Class;
      Collection;
      Domain;
      [
        Blob (Primitive;)
        Boolean (Primitive;)
        Date (Primitive;)
        Domain_Type (List;)
        [
          Aggregation;
          List;
          Primitive;
          SubSet;
        ]
        Gender (List;)
        Integer (Primitive;)
        Path (Primitive;)
        String (Primitive;)
      ]
      Link;
      [
        Action (IsOrderedBy : Link.Attribute.Order;)
        Attribute (IsOrderedBy : Link.Property.SysOrder;)
        Classification;
        Function;
        Part;
        Property (Scope : global;)
        [
          IsIndexed;
          IsInstanceOf (Concept;)
          IsKnownBy (Attribute; ;SysName;)
```

```
            IsOrderedBy (Attribute;)
            IsOwnedBy;
            IsPartOf (Class;)
            IsRoleOf (Class;)
            Owner (Concept; ;Model.Cord;)
            PartOf (;IsPartOf;)
            RoleOf (;IsRoleOf;)
            Scope (String;)
            SubType;
            SysName;
            SysOrder;
          ]
          Relationship;
        ]
        Model;
        [
          Cord;
        ]
      ]
    }
--------------------------------------------------------------
--End of Model Cord
--------------------------------------------------------------
```

7.10. CONCLUSION

CORD, a rich object model, has been defined reflecting the key modelling characteristics identified in the literature. The general concept of an Object supported by the two main abstractions of Instantiation (Class) and Ownership have been supplemented by explicit specification of

- Subtype
- Part
- Role
- Relationship
- Domain
- Collection

and a comprehensive model and schema have been defined. The validity of this model will be tested firstly by implementing it with storage and query facilities to confirm that it functions as a rich object data model adequate to represent real world objects and their behaviour. Later the model will be tested to see if it is general enough to hold the definition of elements of information systems: in particular, summary statistical tables for EIS type reporting.

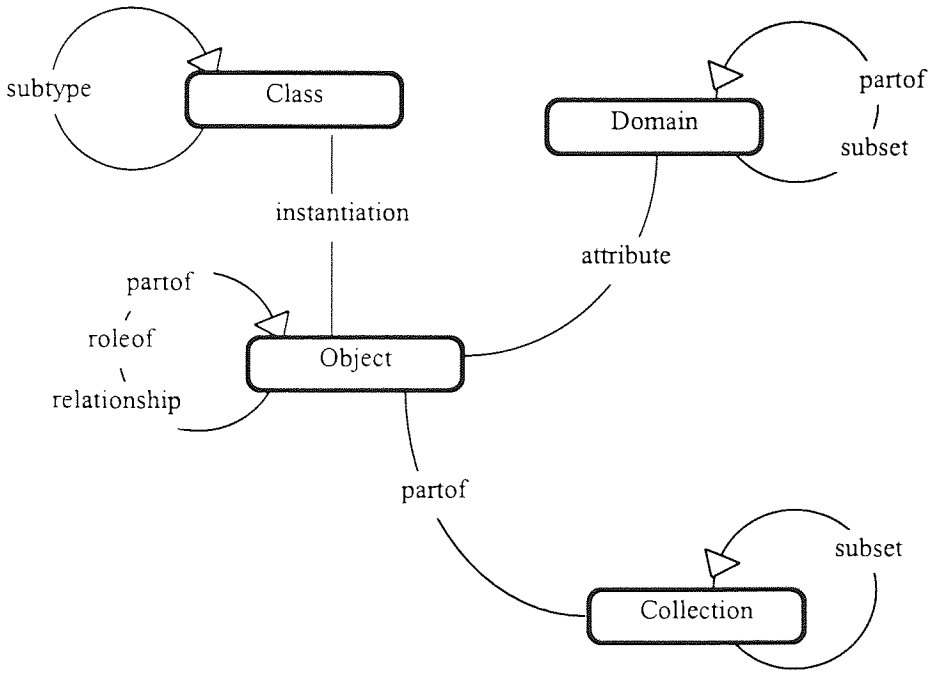

Figure 7. 8 : Main elements and relationships in the CORD model

Figure 7.8 summarises the main components and relationships within the CORD model.

The full CORD model and specification of most of the example fragments used in this chapter can be found in the appendices.

Chapter 8

The Object Prototyper

8.1. INTRODUCTION

Based on the discussion in the last three chapters a data model manager was designed to hold the proposed data model. A flexible model manager called the Object Prototyper has been constructed, based on the binary data model, which enables data models to be entered, edited and output. The development of the Object Prototyper used in this research was carried out over several years and in two distinct versions. Whilst the original concern was the storage and retrieval of a flexibly defined data model, the development used C++ and a home constructed storage manager. At a later stage it was felt that a significant understanding of the data model manager requirements had been achieved and that the user interface was the critical phase in the development. The data manager was then rewritten, still using a binary data model, in Microsoft Access, and Visual Basic was used for application development.

8.2. THE SEMANTIC DATA MANAGER

The data model management system has two major components, a simple database for storing the data model and a windowed interface for accessing it. To give maximum flexibility the data storage component is based on a Semantic Data Base (Rishe 1989) which is used to store triples of information of the sort:

```
Employee HasAttribute Age;
```

8.2.1 Semantic Data Manager version 1

The original semantic data manager used in this project was written in C++ and used a single B+ tree structure to hold the data for each semantic database. The use of the semantic data model and the B+ tree meant that all information concerning a single data model was stored in the same file. However, as the project evolved and C++ was replaced by Visual Basic as the development environment, the structure which had been developed and tested in C++ was

transferred to MS Access.

The original storage manager used a B+ tree storage structure to hold triples of data. To facilitate searching every triple was stored in two forms:

A B C and B C A.

Rishe (1989) identifies eight questions which one may wish to pose to a semantic data base. Using his notation it is possible to identify:

1.	ABC	Does the triple ABC exist? Returns True or False,
2.	AB?	Returns all triples which commence with AB
3.	A??	Returns all triples commencing with A
4.	???	Unusual request answered by the entire contents of the database.
5.	?BC	All triples terminated by BC
6.	?B?	All triples with the middle link field equal to B
7.	A?C	All triples which begin with A and end with C
8.	??C	All triples terminating with C

By storing not only the natural expression but also the rotated expression BCA, it was possible to satisfy queries 1, 2, 3, 4, 5, 6 but not queries 7 and 8. Thus queries which could not be directly answered are "What attributes of A have value C?" and "What objects have as a value C?". This posed no problem in developing the Object Prototyper. In practice in a non semantic data application such as the Object Prototyper , C cannot be interpreted without reference to the domain of B and questions 7 and 8 of Rishe's potential queries would be meaningless.

The fuller discussion by Rishe of the Semantic Data Model (SDM) includes issues of searching within ranges. For example, "Find the all employees whose ages lie between 45 and 55". The main purpose of using SDM was to support the Object Prototyper and store data models and as such range queries were not expected to be relevant.

8.2.2 Semantic Data Manager version 2

The second version of the data manager used the facility of Visual Basic to

connect to MS-Access databases and the triples were stored in an Access database.

The original SDM had a flexible approach to the length of fields, thus triples A:B:C could be of any length as long as the length was less than a block of storage. In converting to MS Access a fixed length field structure was introduced, where each column was large enough to hold object-ids and typical names.

At this stage in the research, the structure of the basic triple had been clarified. In general the three parts formed a sentence of the type

```
"The value of the characteristic B of object A is C"
```

The term *characteristic* was used instead of attribute as attribute has a specific meaning in the data modelling and database field. In fact it was the intention in developing an object modelling environment from scratch to be able to define a range of relationships between objects all of which would be represented by the collection of triples. Thus the semantics of a triple is more generally that:

```
"Object A has relationship B with object C"
```

It also became clear, once the need to store cycled triples was eliminated by the ability to use MS Access indexes, that A and B would only ever contain Object Identifiers. C on the other hand could contain an object identifier where C is an object, or a primitive (text string, value, date etc.) where C represents such a value. In the data modelling part of the development C was most often a character string representing the external name of an object.

```
objectA isnamed "customer"
```

or a small integer representing ordering or cardinality.

```
ObjectA hasMaxCardinality "6"
ObjectB isordered "2"
```

Thus by making an arbitrary constraint on the size of default names of objects the field size for field C could be fixed. Given the decision to use date-time to generate object identifiers which were 13 characters long, the basic field size of all three fields of a triple was set to 20 characters.

The primary table holding triple data became

```
define table objects(
(obid char(20),
linkid char(20),
pred char(30));
```

The whole triple constitutes the key so that no duplicate triples could be entered.

In the implementation it was necessary to allow for the fact that some values may be longer that 20 characters. One table held such triples, another held any long fields too long to fit in the third column of the main table. Where an overflow occurred a reference to another object in the text table was used. This gave an economical way of managing the few long strings that were found to occur. Instead of storing the triples in two forms as in the C++ version the MS-Access version was able to rely on SQL expressions to retrieve data in any of the Rishe 8 forms (Rishe 1989), and where performance was an issue indexes could be specified for the more standard searches.

8.2.3 Object identifiers

The main structures stored in the SDM turned out to be object identifiers. To create suitably unique object identifiers the date-time function (Now()) was used. Now returned a full date and time to the nearest second.

```
97/01/01 10:30:41
```

In version 1 this (or rather the equivalent Turbo C++ function) was adequate to generate unique identifiers. However the faster processor used for the version 2 development caused more than one identifier to be generated in the same second. In the second version a counter was added which added a serial number to identical identifiers thus guaranteeing uniqueness. The non numeric characters were stripped out of the date-time string to create the identifier.

```
97/01/01 10:30:41 became 970101103041
```

Rather than storing long integers as identifiers which would have improved

performance, the Object Prototyper used character strings of numbers so that the database was easy to inspect and debug. Moreover some object identifiers had to be known to the application software. Thus if a new object was created it had to be specified as an instance of a class:

```
"attribute is an instance of the class link"
```

and the triple

```
attribute:isinstanceof:link
```

had to appear in the model database somewhere. In practice this was stored as

```
@attribute, @isinstanceof, @link
```

where `@attribute`, `@isinstanceof` and `@link` are the object identifiers of the Attribute class the `IsInstanceOf` attribute and the Link class. Because the application software needed to know the identifiers of some of the base objects, the use of prespecified identifiers of the form of a string of text preceded by the identifier character "@" was introduced. In general to facilitate debugging when a new object was to be stored an attempt was made to use its name to generate a unique identifier. The numeric string generated from the date-time function was only used when a more readable unique identifier was not available. The actual ID was created by stripping out the non numeric characters and adding an "@". None of this was strictly relevant to the logic of the Object Prototyper, but was used to facilitate debugging without a significant reduction in performance.

8.2.4 Long text fields

When a string longer than the maximum twenty characters was to be stored a second table was used and a reference made to the appropriate row.

```
Create Table Text (
textid char(20),
text char (200))
```

To indicate a redirection the objectid (in this case a reference to an object in Text) was prefaced by a ">"

8.2.5 Names

In specifying objects externally, a name is required. Names frequently have to be unique within a given scope. It was decided that names could be any sequence of printable characters with single embedded spaces or underscores. After a little experimenting with the typical names being used in early data models a default maximum size of 30 for names was adopted.

8.2.6 Field sizes

The minimum field length for the object identifier fields was 12 (date-time) plus an extra digit to distinguish duplicate date-times plus "@" to identify identifiers and ">" for redirection i.e. 15 characters for the third column and 14 for the first two columns of a triple.

Because searching to match a name is an important aspect of loading a data model or parsing a query, an implementation decision was made that object names should not be redirected, so column three of the SDM table was to be large enough to hold names without redirection. The following default restrictions were built into the Object Prototyper:

1. Maximum digital identifier: 14 characters long
2. Maximum text based identifier: 20 characters
3. Maximum name for an object: 30 characters

These sizes meant that columns one and two had a width of 20 whilst column 3 had a width of 30. For added flexibility, when a new prototype is opened the maxima for text based identifiers and object names are taken from the column widths of the database table.

8.3. THE OBJECT PROTOTYPER

The Prototyper was developed in two phases, the first being in C++, the second being written in Visual Basic.

Obid	Linkid	pred
@children	@owner	@employee
@children->count(em	@isinstanceof	@tattribute
@children->count(em	@name	Children->Count(Employee)
@children->count(em	@owner	@empt1
@children->count(em	@tapath	>@1711961144260
@children->count(em	@tatype	summary
@cla:member	@isinstanceof	@class
@cla:member	@name	Member
@cla:member	@subtype	@person
@0412960957230	@name	Sex
@0412960957230	@owner	@ev14
@0412960957230	@tatype	category
@0412960957230	@vaorder	1
@0412960957230	@vaparent	@0412960957221
@0412960957230	@vasource	@tat:sex

TextId	Text
@1711961144260	@children->@fun:count(@employee)

Table 8. 1 : Samples of triples as stored in the database.

8.3.1 Object Prototyper version 1

The first version of the Object Prototyper was developed in Borland Turbo C++ and ran on a PC under Windows 3.1x. Data storage was in a SDM based on the Rishe (1989) Semantic Data Manager. The basic data model had a minimum of concepts "built in" enabling the maximum flexibility in developing data models.

8.3.2 Object Prototyper version 2

The first version, being constructed in C++, was relatively hard to maintain, and although the user interface was built to Windows standards it proved hard to implement some features that were considered desirable. In particular a drag and drop approach to the layout of table views was wanted. It proved relatively easy to write the Prototyper in Visual Basic and hence to take advantage of the better

user interface construction tools available and to use a standard relational data base for the data manager. The detailed functionality of the Prototyper is discussed after the basic data model has been introduced.

8.4. THE BASIC DATA MODEL

Reflecting the basic structure of semantic analysis, that there are referends and predicates, the top level structure distinguishes between Objects and Links. It was decided to enable models to be loaded which would gather together some aspects of the data model. The main model which included the main structure of the data modelling schema was called "CORD". A separate model was to be defined for the Summary Table Model (COST) and one or more models containing test data were expected to be loaded. When a particular model was the focus of attention then the Domain of Discourse was defined by those models "included" with the active model.

- All concepts are instances of Concept, the top level least specific class.
- Any concept may have links to other concepts. The model does not limit the type of link which can be defined but all new links are instances of Link. A fundamental link is Attribute.
- A structure has been developed for printing out and reading in data models.

8.4.1 The model schema

The schema is designed to be compatible with the CORD conceptual model. A concept can have three types of dependency. Following from Smith and Smith (1977a) Aggregation and Generalisation are identified as the two main abstraction mechanisms for building data models. Thus any concept can be defined in terms of its components (the aggregation abstraction) and its instances (the generalisation abstraction). Different types of parentheses are used to gather together objects which bear this relationship to a concept.

```
Concept
      { aggregation components }
      [ instantiations ]
Person
```

```
{ name; age; address; }
[ John; Sue; William; ]
```

This defines an object Person which is made up of objects *name*, *age* and *address*. John, Sue and William are instances of Person.

A third relationship, needed to create data models, is the mapping of an element of a domain to an object: the assignment operation. This will be shown with simple parentheses which reflects the common representation of record structures.

```
@John(name: Paul ; age: 25)
```

This states that the object identified by `@John` has the attribute *name* assigned the value "Paul" and the attribute *age* assigned the value "25". As the model is an object model, the object John will still be identified by the character string "@John" but if:

```
Print(@John.name)
```

was executed the result would be the character string "Paul".

This is slightly counter intuitive with real world objects like people but reflects OO conventions. A process for bridging this gap has been applied in the Object Prototyper and will be discussed later.

8.4.2 Basic elements of the model

The fundamental constructs discussed above are adequate to define any subsequent data model. However a data model manager which is to behave with some understanding of the model will need to know which relationships are attributes and many other features of the model. This is the reason why some data modelling constructs have been given prespecified object identifiers.

8.4.2.1 Instantiation

An important example is the case of instantiation. It is frequently necessary to retrieve all objects in a particular class. This involves the execution of the query

```
?:ISANINSTANCEOF:ClassA
```

In searching the SDM this involves a SQL expression like

```
Select OBID from objects where linkid = "@isaninstanceof"
and pred = ClassAId
```

8.4.2.2 Names

Other basic concepts which have been introduced are essentially for convenience including the naming of objects. An object needs a name by which it can be referenced in an external format. Most retrieval operations will produce a collection of object identifiers and an external listing of the collection will be, in general, meaningless. Some objects will have names as part of their attributes (people, companies, products etc.) but others will have to be given a name on creation (classes, attributes etc.). This is resolved by associating with every class of concepts an attribute KnownBy which would specify an attribute of the object to be used as a name. Where this is a natural attribute of the object, that will be used. Where a natural attribute does not exist, then the attribute SysName is used to hold the object's name.

8.4.2.3 Order

In order that values can be assigned within records, it is necessary that attributes have some ordering. The SDM does not preserve the input sequence of attribute definition or of any definitions so an automatic sequencing is necessary. A similar problem could be expected to occur in defining actions or life histories. This was treated in the same way as object names. An attribute OrderedBy has been defined which holds, for a class, the name of the variable which will hold its default order. Where no natural variable has been defined then the attribute SysOrder is used.

8.4.2.4 Ownership and Models

It has become a central feature of the model that every object has to be owned by something. In general objects are owned by a Model. Thus a Model becomes a

means of collecting together a number of interrelated objects. Not all objects are owned by their model directly. All link objects which are existence dependent on another object are owned by one end of the link. Thus all attributes of a class are owned by that class. The existence of relatively independent models results in the possibility of naming confusion.

Scope of Models

A model can specify that other models are known to it and that the resolution of paths in loading the model can include names of objects in the included models. There is a base model called CORD and all other models loaded will specify that CORD is included in their scope. As a model cannot be understood without reference to its included models, and these would not be resolved in the first scan through a model, models have to be introduced to CORD before they can be loaded.

```
Model.CORD
 { Concept.Model[Cost
    (includes: CORD;)]}
```

The above fragment defines a new element of model CORD which is a Model called Cost. This model specifies that CORD is included in its scope so that in reading it in references to elements of the CORD model will be correctly resolved.

Ownership

Ownership is implemented with two attributes in a similar manner to Order and Name. An individual object can specify its owner

"Attribute age is owned by class employee"

would appear in the database as the triple

```
@age: @owner: @employee
```

There should be a cascade delete process which will eliminate all objects related to a particular model from the database. All objects owned by the model should

be deleted and recursively any objects owned by them. Any instances of objects owned (recursively) by the model should be deleted irrespective of who owns them.

In the CORD schema it will be unusual to see explicit reference to the Property *Owner* as ownership is usually implicit in the creation of an object. Thus:

```
class.person
{ attribute.age (integer;)}
```

specifies that the class person has an attribute called age which has an integer domain. The implementation of this element of a data model will create an instance of link attribute called *age* which will be owned by the class *person* and an instance of an attribute *domain* which is owned by the attribute age and takes the value integer.

8.4.2.5 Basic model

The core concepts which were pre-loaded into a new blank database were:

```
------------------------------------------------------------
--Model Cord_0
------------------------------------------------------------
  Model.Cord
  {
    Concept (Scope : global;)
    [
      Domain;
      Link;
      [
        Attribute (IsOrderedBy : Link.Property.SysOrder;)
        Property (Scope : global;)
        {
          Attribute.D1domain;
          Attribute.D2domain;
          Attribute.Default;
        }
        [
          IsInstanceOf (Concept;)
          IsKnownBy (;;SysName;)
          IsOrderedBy (Attribute;)
          Owner (Concept; ;Model.Cord;)
          Scope;
          SysName;
          SysOrder;
        ]
      ]
      Model;
      [
```

```
        Cord;
      ]
    ]
  }
------------------------------------------------------------
--End of Model Cord
------------------------------------------------------------
```

The pre-loaded elements of the data model were augmented by the loading of the rest of the CORD model. This resulted in the models shown in Appendix 1. The base model was not loaded by reading in a CORD schema but by direct insertion of triples into the database. The functions which achieved this are shown in Appendix 4.

8.5. THE OBJECT PROTOTYPER VERSION 2

The second version of the Prototyper is discussed here in more detail. A typical windows interface has been provided with a mixture of menus and buttons to initiate functions.

The main screen menu provides access to

1. Prototype - to create and open prototypes,
2. Load - to import data models,
3. OQL - a query facility,
4. Browse - a complex browser showing the attributes and values associated with any objects,
5. Object Tree - a diagram showing the hierarchy of instantiations,
6. Print - a procedure for printing data models.

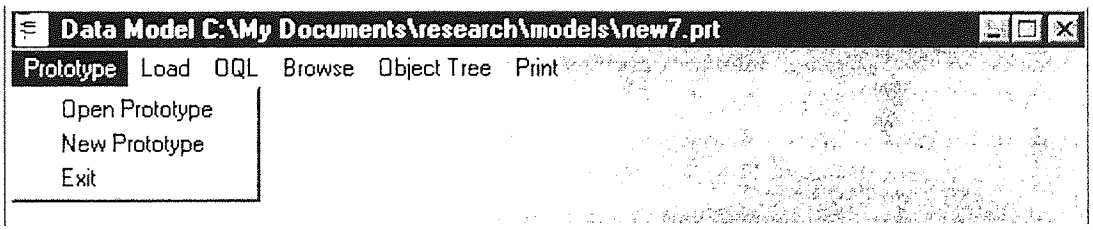

Figure 8. 1 : Main Screen Menu

8.5.1 The Prototype sub menu

Like the standard File menu on windows applications, the Prototype sub menu (Figure 8.1) enables the opening of existing prototypes, the creation of a new

(nearly empty) prototype and the exiting from the Prototyper.

8.5.2 The Loader

The data model Loader has been constructed to read in data models which are stored in text files. The structure of the model language will be discussed below in some detail. Figure 8.2 shows the Loader form. The Loader operates in a three pass manner. In the first pass it creates all objects, in the second pass it assigns any properties and in the third pass it assigns any other attributes. The three pass process enables reference to be made to objects not defined earlier in the model schema, and for the main characteristics of the objects to be set through assignment of property values.

Thus if employees had a Department attribute, and departments had an attribute Manager of domain Employee there would have to be a forward reference to an object not yet defined, as in the following fragment from the Emp model:.

```
OBJECT.Class.Employee (SubType: Person;)
      {
        Attribute.Age (Integer;)
        Attribute.Sex (gender;)
        Attribute.Status (MaritalStatus;)
        Attribute.Department (Department;)
        Attribute.County (County;)
        Attribute.Address (address;)
        Attribute.Children (Employee; 6; 0;)
      }
OBJECT.Class.Department
      {
        Attribute.Manager (Employee;)
        Attribute.Location (Region;)
      }
```

In the section of model shown, the domain of the attribute Department of Employee is not assigned on the first pass, neither is the domain of the attribute Manager of Department. Both are assigned on the third pass when the Loader has already created both classes. However the property of Employee *Subtype* is assigned the value Person on the second as such definers affect the nature of the objects created.

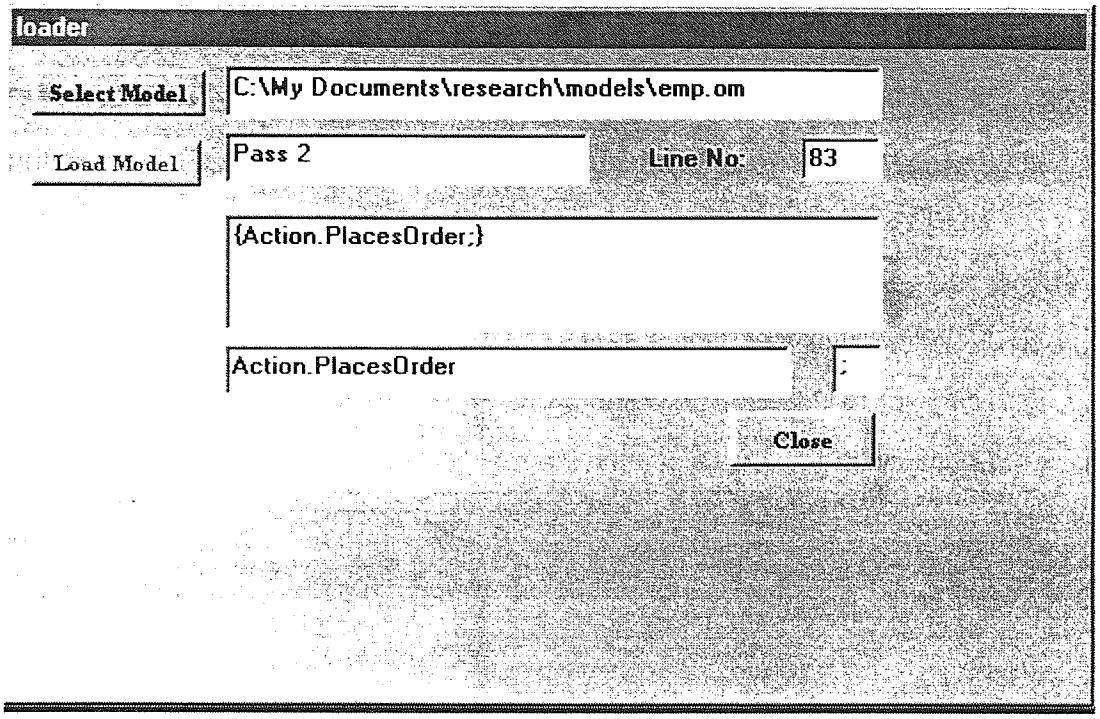

Figure 8. 2 : The Loader Screen

Whilst loading a number of status and trace messages are displayed so that in the event of failure it is possible see what has happened. This reflects the prototyping nature of the tool; an end-user tool would have substantially more validation of syntax and semantics and produce more useful error messages.

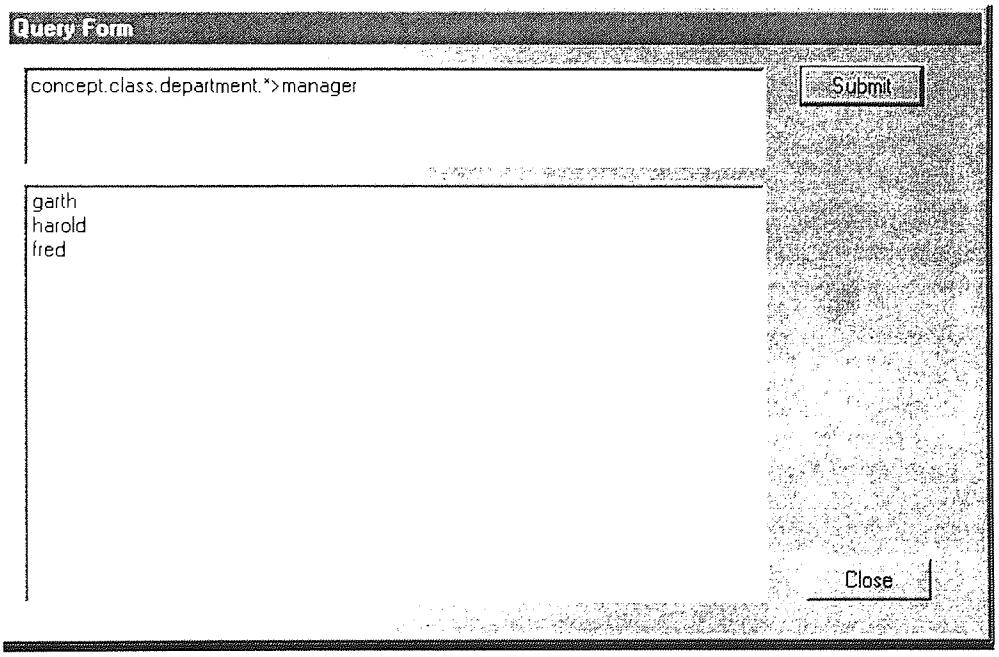

Figure 8. 3 : The OQL form

8.5.3 The OQL

The OQL sub menu enables the user to experiment with the object query language built into the Prototyper. Not that this is not the same as the OQL defined by the ODMG (1993) but the term is used here generically. Simple queries can be entered into the top box and executed. The main purpose in providing this facility was to enable path expressions required in the later stages of development to be tested. It also provided a faster way of examining some aspects of the data than the Browser. In the example in Figure 8.3 the managers of all departments are listed.

8.5.4 The Object Tree

Having loaded one or more models into the database two tools have been provided to enable the user to examine the resultant DoD.

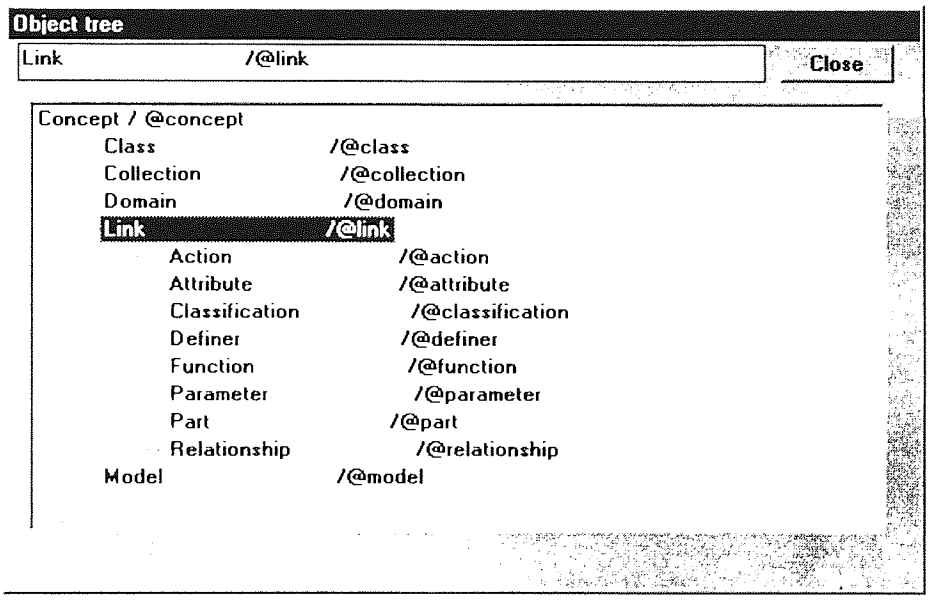

Figure 8. 4 : The Object Tree

The Object Tree shows the instance hierarchy and enables the user to expand or contract any level of the model. Figure 8.4 shows the Object Tree with some of the basic CORD objects displayed

8.5.5 The Browser

The Browser enables the characteristics of objects stored in the model to be examined in more detail.

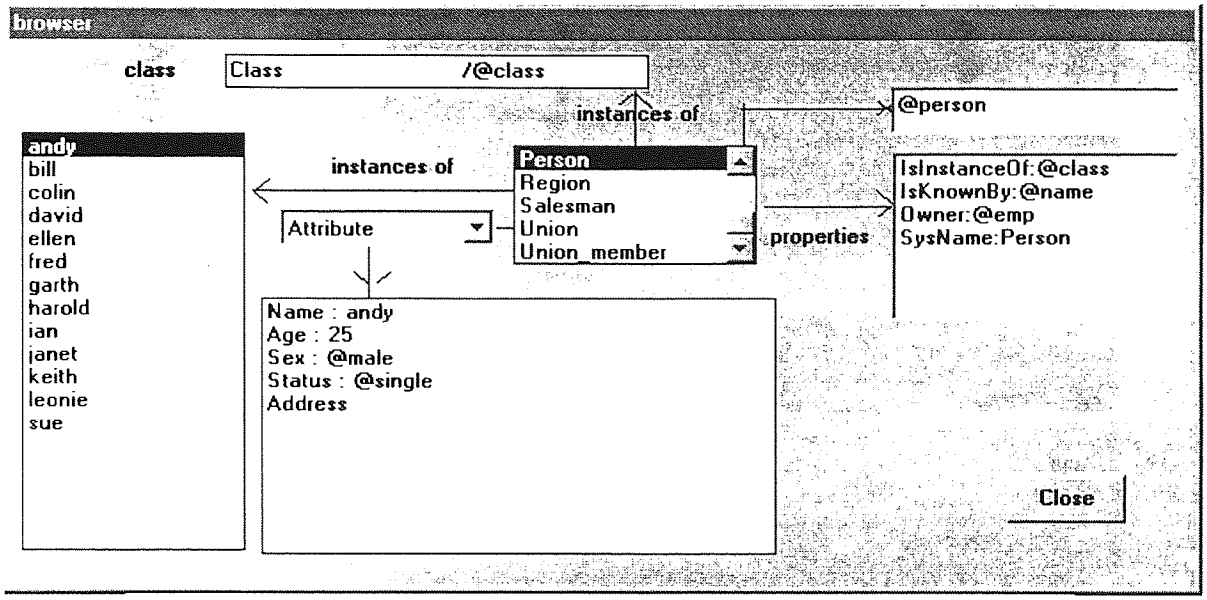

Figure 8. 5 : The Browser

The Browser enables the user to browse the instance tree and, for any object, to show five categories of information about it:

1. Its class,
2. Its system properties,
3. Any instances it possesses,
4. and for any Link those links of that type created for it,
5. To inspect for any instance the value of any link.

The various panels associated with this information are visible in figure 8.5

8.5.6 The Print form

The Print form, shown in figure 8.6, enables the user to display and to print to a text file any model stored in the database. The output structure is in the same form as the data model schema so that it can be read back in if required.

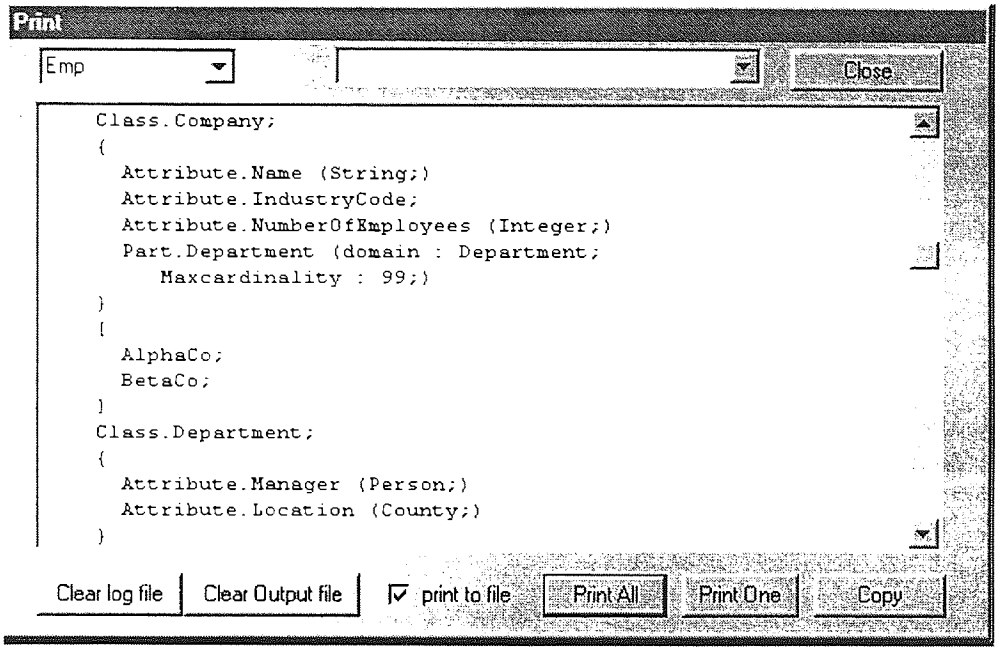

Figure 8. 6 : Print form

8.6. DISCUSSION OF THE OBJECT PROTOTYPER AND THE CORD SCHEMA

8.6.1 The CORD schema

A flexible object model has been developed which is free from many of the constraints of existing commercial and research models. The model has a flexible approach to the definition of links between objects, enabling a semantically richer schema to be developed. The model enables the semantics of aggregation to be expanded in a way unavailable in alternative models and also enables the implementation of the Role relationship.

8.6.2 The Semantic Data Manager

The semantic data manager permits the complex relationships specified in the data model to be stored and retrieved with SQL expressions. The approach means that every structure which the data model requires can be supported by the data manager, so the underlying data storage model does not inhibit higher level developments or experimentation.

8.6.3 The Object Prototyper

The Object Prototyper took a considerable time to build, but the investment paid off. The Prototyper makes it easy to define and load object models, to explore the model and to examine the low level structure of a stored data model. This was essential as new abstractions were being experimented with. Had the object model limited itself to classes and attributes then some simple enhancements of the relational model would have been sufficient conversely if only a simple prototyping tool had been developed then only a simple data model could have been defined and tested.

The other role of the Prototyper is to validate the object model. It proved possible to hold all elements of the object model in the SDM and then access them via the Object Prototyper. Higher level constructs have been defined in terms of lower level ones and in the end everything has been defined in terms of the main abstractions of aggregation and generalisation.

8.6.3.1 Generalisation

There are three ways in which the generalisation abstraction has been implemented in the final model:

1. Instantiation instance class relationship
2. Subtype class - class relationship
3. RoleOf instance - instance relationship

8.6.3.2 Aggregation

Aggregations have been implemented in a variety of ways:

1. Ownership - the concept of ownership which is the basic and most general aggregation concept is implicit to the model
2. Part - Part has been declared in two ways:
   between domains
   and between instances
3. Attributes and Properties are aggregations through ownership as are any Links.

Properties were introduced into the model as there was to be no implicit invisible structures and attributes of classes, which played a particular role in specifying the nature of objects in the final version.

8.6.4 Conclusion

The CORD model has proved to be a consistent, rich data model. It has proved to have the capacity for defining its advanced structures within its schema.

A conceptual model could be tested by creating a number of theoretical requirements and by confirming that it meets these requirements, it can be shown to be applicable over a range of projects or it can be shown to be implementable by embedding it in a software tool. The first option has been hardly ever used for practical schemas only for mathematical models of conceptual schemas. As the intention was to test the model in the subsequent stage of developing a summary table model the second approach would also be used. It was for these reasons that the Object Prototyper was constructed. The Object Prototyper has proved to be a satisfactory tool for experimenting with data models for managing and editing data through its interface with the Semantic Data Manager. The Object Prototyper and the CORD model together have proved to be a sufficient basis for the development of the summary table model in the later stages of this research. As it was not relevant to the research to use the Object Prototyper as a general CASE tool there seemed no reason why it should be tested by a larger group of users.

Section B

From Data to Management Information

B.1. INTRODUCTION

The first stage of the thesis is now complete, the literature review on conceptual modelling has been completed and an object model has been developed. The next stage changes focus, away from the modelling of the problem domain to the modelling of management information. This section reviews the work so far and introduces some general issues about data and information before leading in to the examination of the nature of management information in general and statistical tables in particular.

B.2. THE CORD MODEL REVIEWED

In Section A of this thesis a semantically rich object model was defined which, while based on existing concepts, is original in several ways. Firstly it is defined in a boot strapped way which explicitly defines all constructs within the model, unlike most conventional models which take attributes, relationships, subtyping and many other primitive constructs. CORD is constructed from the primitive constructs of instantiation and ownership, from which the rest of the model is defined. CORD then defines not only the normal abstractions of Classes, Instances, Attributes and Subtypes but the less usual abstractions of Part of and Collections. The significance of CORDs contribution also arises from the prominence it gives to the definition of Domains and the use of Roles to bring much of the behaviour of objects into the object model and out of the realm of process definition.

That CORD 'works' has been demonstrated by building the Object Prototyper, a software tool enabling object models to be stored and queried. The Object Prototyper was conceived solely for this research and has not therefore been tested by a wider community. It is apparent that it could be useful as a pedagogic tool for students learning about conceptual object modelling, but the necessary work required to translate a research tool into a robust teaching instrument has not been

taken on board.

B.3. FROM DATA TO INFORMATION

Having constructed a robust conceptual model, this thesis next addresses the issue of management information.

Whilst in the earlier part of the report terms such as data and information have been undefined and their conventional meanings assumed, it is appropriate to clarify the distinction between them, because this explains the significance of the second section of this thesis.

Data is defined by Avison and Fitzgerald (1995) to be "unstructured facts" whilst *Information* "has a meaning and use to a particular recipient in a particular context". They give the example (modified here) of the holder of driving licence 7870199 having the date of birth 25/07/90, hence suggesting an infringement of the law that a holder of a driving licence should be over 17 years of age. The argument here is that the structuring of the facts represented by the strings 7870199 and 250790 turns data into information. This is oversimplistic as even this structured data is not information to someone who is concerned about the climate, say. Moreover the structuring of the data does not by itself constitute information without a knowledge of driving licence procedures, the current date and the law as it applies to motoring. Thus most of the information contained in relational database tables, or structured files, should be considered data. It is organised and accessible so that the transformation into information is considerably facilitated, but the organisation per-se makes only a small contribution to its information content. Information systems are much more than data storage and retrieval systems: they "collect, store, process, and report data from various sources to provide the information necessary for management decision making" (Hicks 1993). If the data stored in a database is to become information, then some applications are required to retrieve and present it as required by the user. Such applications may be purpose written programs, but more often than not will use flexible query languages and high level report

generators to effect the transformation.

The literature uses other terms to categorise the domain stretching from data to understanding. Intelligence, knowledge and wisdom are all in some way used to describe the area. Intelligence has two distinct meanings, which are both relevant to the understanding of Information Systems. Intelligence can refer to the collection and analysis of information, usually environmental, with a view to defining business strategy (Kochen 1989). It can also refer to a property assumed of a person who behaves in an intelligent manner (Sternberg 1982, p 225). In this context, it is often asserted that knowledge, reasoning and problem solving skills are attributes which contribute to intelligent behaviour. Sternberg's assertion that "Knowledge is accumulated as a function of experience" (Sternberg 1982 p 19) suggests that knowledge is more than just information as we have defined it above. This could seem distinct from Jardine's (1986) approach (based on Dretske) in which knowledge is the state of accepting the truth of a piece of information. However Jardine also goes on to say that "the communication of knowledge thus depends on having a shared conceptual scheme". Wisdom would appear to relate to judgement and qualify intelligent behaviour which shows evidence of taking a wide range of factors, particularly values, into account in making a decision.

This suggests the following taxonomy of the terms as used in the Information Systems area.

- Data is a discrete collection of symbols.
- Information is obtained from data when the context of the data is defined (Curtice and Jones's (1981), Contextual, Representational and Physical levels)
- An information system stores data and the context necessary to turn that data into information and makes the information available to the user.
- Intelligence (type1) is environmental information, collected and stored as data for strategic purposes
- Knowledge is a collection of relevant information together with an appropriate conceptual framework.

- Intelligent behaviour (Intelligence type 2) results from the use of appropriate knowledge, together with problem solving and reasoning skills.
- Wise behaviour (wisdom) is a form of intelligent behaviour which includes the application of values.

In part A, the foundations were laid for defining and storing the conceptual framework which is necessary for turning data not only into information but potentially into knowledge. In this part of the thesis standard techniques of transforming data into knowledge for the purpose of management decision making will be reviewed. In particular the focus will be on statistical summary tables. The aim is to derive a method for integrating the definition of statistical summary tables in the formalism of the CORD model and then testing whether this model is adequate. If it is, then this thesis will have contributed to the creation of knowledge from data.

B.4. SECTION B OVERVIEW

Chapter nine reviews some of the literature on the nature of *information systems* and the forms and reports which are familiar components of any information system. For an information system to support decision making, it must produce information structured and presented to meet the needs of the decision maker. The main technique for producing aggregate and summary information is that of statistical summaries. Chapter ten reviews the rich literature on *statistical summaries* which has traditionally been treated independently from the work on management information systems and conceptual modelling. In chapter eleven the second main synthesis contributed by the thesis appears, it is the *Collections of Objects Summary Table (COST)* model. For the first time a summary table model is proposed which is coherent with a conceptual modelling framework. Whilst chapter eleven examines the COST model as it deals with traditional summary table attributes, in chapter twelve the particular nature of summaries in the context of an object model are discussed. In chapters eleven and twelve the COST-Table logical model is discussed. However the presentation model also needs to be

developed and this is described in chapter thirteen. Throughout these three chapters the COST model is described by modelling it within the CORD model, thus testing the hypothesis of this work: that it is possible to integrate management information reporting with a suitable conceptual model. The results of *implementing the COST model* within the Object Prototyper are reviewed in chapter fourteen.

The contribution of the thesis is reviewed in chapter fifteen.

Chapter 9

Modelling the information system

9.1. INTRODUCTION

The earlier chapters in this thesis have followed the traditional approach of trying to model the real world and thus have concentrated on modelling constructs which reflect the nature and behaviour of objects within the environment of the organisation. This is distinct from objects within the organisation itself, and even within the information system being developed.

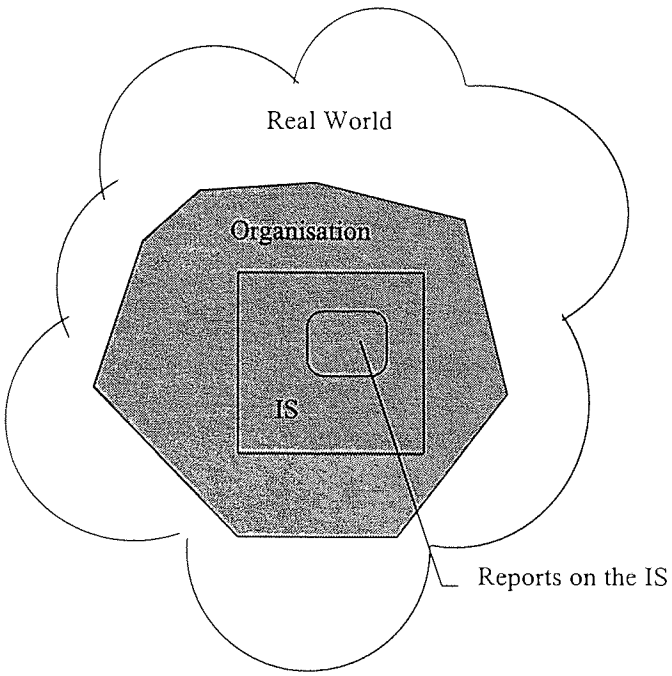

Figure 9. 1 : Nested levels of modelling

These are of course arbitrary boundaries and whilst customers may be valid external entities for an order processing system, derived information such as sales forecasts are also real things for managers developing market strategy, and are objects in the environment of a marketing information system. A rough distinction would appear to be that in modelling the real world the analyst is concerned with metadata about classes and the typical behaviour of members of classes. Such a system can be developed without the participation of a single real instance of many of the classes. Thus a processing system for the new national lottery can be designed and implemented without the participation of a single

punter, ticket or prize.

In developing an information system it is necessary to provide managers with information which is based on summaries and aggregates of instances and transactions from the real world. These reports can be planned in advance but they clearly contain derived information, reporting on activities within the information system itself.

As has been observed, the nature of boundaries depends on the perspective of the viewer and what is considered derived information from one point of view may be a real world object from another.

In trying to understand the nature of information in organisations Van Gigch and Le Moigne (1990) distinguish between circulating information and generic information :

> The organisation establishes "traces" of its own activities: a shipment, an order, an invoice, a bank draft, or a sales receipt are all deliberate and artificial traces, whose production are indispensable to the smooth enfolding of the organisation's activities. These traces are information which evolves or emerges, as the enterprise carries out its daily activities.
> *Generic information* is that which is "generated" by the organisation's own actions and which, initially, does not necessarily exist. In contradistinction, *circulating information* originates from outside the organisation. (Gigch 1990)

This chapter aims to examine what is known about organisations and information and hence aims to cast some light on the nature of information systems.

9.2. LEVELS OF ABSTRACTION IN MANAGEMENT

This section reviews levels of abstraction in organisations and their implications for information management. Much of it is based on the work of AlShawi (1991).

It would appear to be a truism that information used in organisations is defined in terms of the activity that uses it. The information usage of activities however would appear to be a function of the management level at which they take place. The nature of information and its relationship with organisational activities has been extensively commented upon, with the base assumption that abstract

aggregate information is required at high levels in the management hierarchy whilst detailed concrete information is required at low levels.

The relation between level of observed abstraction and perception has been suggested by the Structural Differential model of Korzybski (1933). The link between perception and position in the organisation hierarchy has been suggested by Jaques (1976, 1978). By examining the two links, a relation can be drawn between the ability of a user to deal with high levels of abstraction and the user's level in the hierarchy. It is clearly appropriate to examine the concepts of organisation hierarchy and level of hierarchy.

It is repeatedly stated in the literature that an organisation has three hierarchical levels: operational, tactical, and strategic. There seems however to be a general ignorance of the fact that the model of management based on these levels was based on a view of the range of types of decision made within a particular classification of organisation activities. As suggested by Anthony (1965), the approach was intended as a framework for planning and control systems. Yadav (1983) has pointed out the inadequacies of the three level approach.

> "Anthony's main contribution is the conceptual framework which recognises the fact that information needs are different at different levels of managerial activities. This framework provides insight, but is difficult to apply in determining information requirements. In the real situation, there are no clear-cut boundaries for different levels of activities. Furthermore, information requirements are affected by organisation structure and processing in addition to the levels of activities".

Cibora (1981), on the other hand, described organisation analysis and information systems design methods as being,

> "locked into the conception of the organisation and its information systems as a pyramid composed of three layers: operations management, control, and strategic planning."

Others would even go to the extent of describing the traditional three level approach as no more than,

> "useful for discussion purposes" (Shahabuddin 1987).

Nonetheless, reference is generally made to the three managerial levels in an

organisation irrespective of the context in which management levels are discussed. In AlShawi (1991), the legitimacy of utilising the three level management approach for the purpose of information systems analysis and database design was examined. It was established that management levels as defined by Anthony (1965) cannot provide a definitive basis for classifying information for the purpose of providing specific and clear-cut reference to the degree and scope of information relevance to a user. Therefore, an alternative mechanism for identifying hierarchy levels is required. Such a mechanism need not be concerned with the intricacies of structural relationships, but should be depended upon to provide valid grounds on which general reference to data abstractions in relation to structure hierarchies could be made.

Alternative methods to defining basic levels of hierarchic structure date back to Urwick's (1943) renowned work on the theory of organisations. Paterson (1972) and Beer (1979) addressed the issue of levels from the point of view of decision making and neurocybernetics, respectively. However, their work is mainly concerned with the management side of organisations, which makes it susceptible to inheriting certain shortcomings posed by authority channels, e.g., the constant update of such channels.

Jaques (1976, 1978) presented the theory of levels of abstraction as an explanation for the existence of the hierarchic structure of bureaucracies. The proposition stated that the stratification in structure represented by levels forming the hierarchic shape, as has always been recognised, can only be explained by the theory of the distribution discontinuity of human cognitive competence. The theory is backed by extensive research into social scientific issues in general, and management and cognitive psychology in particular. The suggested levels of abstraction present a dependable alternative to conventional approaches.

Rowbottom and Billis (1977, 1987, 1989) presented a work-level approach in which they suggested five levels in a structure hierarchy through which all organisation activities are carried out. They have also identified two additional

levels (sixth and seventh), which are usually found in major corporations such as departments of state, and large local authorities. This, however, does not exclude a smaller number of levels in small organisations. The approach provides sound foundations for identifying stable organisation levels. The stratification is one in which "the range of objectives to be achieved, on the one hand, and the range of environmental circumstances to be taken into account, on the other, broaden and change in quality at successive steps" (Rowbottom and Billis, 1989).

The work-level approach presents valid foundations for defining the kernel of structure hierarchies, and presents reliable guide-lines for the identification of levels in a structure hierarchy. It is based on issues more intrinsic to the organisation than its functions or line of authority which are used at times as the bases of structure. It was because of this that AlShawi (1991) in developing The Semantic Conceptual Organisational Model (SECOM) used the work-level approach as a guide for identifying the number of hierarchic levels in an organisation structure.

9.3. MANAGEMENT INFORMATION

Although the term Management Information Systems (MIS) has been in use for over 40 years (Ackoff 1967), the definition of the nature of Management Information has been left to common usage. The major works on the definition of data (Kent 1986, Codd 1979) have focused on record structures which are naturally associated with transactions or the artificial traces which are created within an information system in order to "memorise" the objects in the DoD and to keep the real world and the world of the information system synchronised. When an employee resigns, various pieces of paper are generated and transmitted between the employee and the organisation and within the organisation to record this fact. Van Gigch and Le Moigne (1990) discuss the role of the memorisation of information in the creation of the "intelligent organisation". Not only do these items circulate in the organisation but images of them circulate within the information system. Keeping the computerised part of the information system

synchronised with the manual part of the information system and both synchronised with the real world is one of the objectives of any information system.

9.4. REPORTS AND FORMS

Two externalisations of the information within an organisation are current in all organisational information systems. These are reports and forms. The specification and implementation of forms has developed within the area of Office Automation Systems (OAS) whilst within the MIS tradition, the main structure for externalising management information has been the report. Both forms and reports existed long before computerised information systems and hence have accepted structures which newer technology has had to reflect. However, little theoretical work on the nature of forms and reports has been evident until the latest generation of information systems needed more rigorous definitions to enable the automation of office procedures (work flow systems) and the support for the definition of reports (report generators).

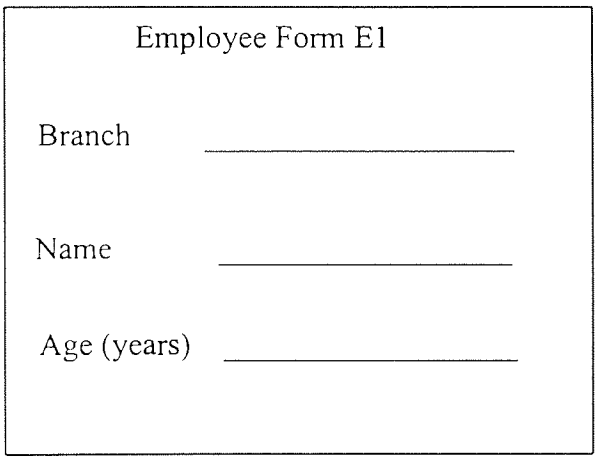

Figure 9. 2 : A screen form

Screen forms as typified by current screen form managers (MS Access, dBase IV) permit the placing of fixed text and data fields, and allow operations at the form level such as data entry and editing, at the record level such as the creation and deletion of records, and at the table level such as the retrieval of records (Query By Example).

Employee Analysis			
Branch	Gender	Number	Average age
Birmingham			
	Male	20	54.5
	Female	25	37.6
	Total	45	42.3
Coventry			
	Male	10	58.3
etc.			

Figure 9. 3 : Typical MIS report

Report structures as revealed in current report generators (SQL*Report, Crystal Reports) permit the layout of pages and columns to hold data and include the summarisation of data and the use of break levels to group records. Report generator programs have emerged to meet information needs but have been constrained by a technology which was based on the linear processing of files. This means that examination of the needs of Management Information Systems can only be partly conducted through examination of existing systems.

9.4.1 Architecture of forms and reports

Adiba and Collet (1988) point out that dynamic forms can be modelled in the same three level ANSI/SPARC architecture that has been used for database applications. There is a physical model which is concerned with the detailed storage of data, nearly always in the form of a relational data model, a conceptual level which they call an abstract form which models the semantics of the data and the external model which is concerned with the presentation of the data.

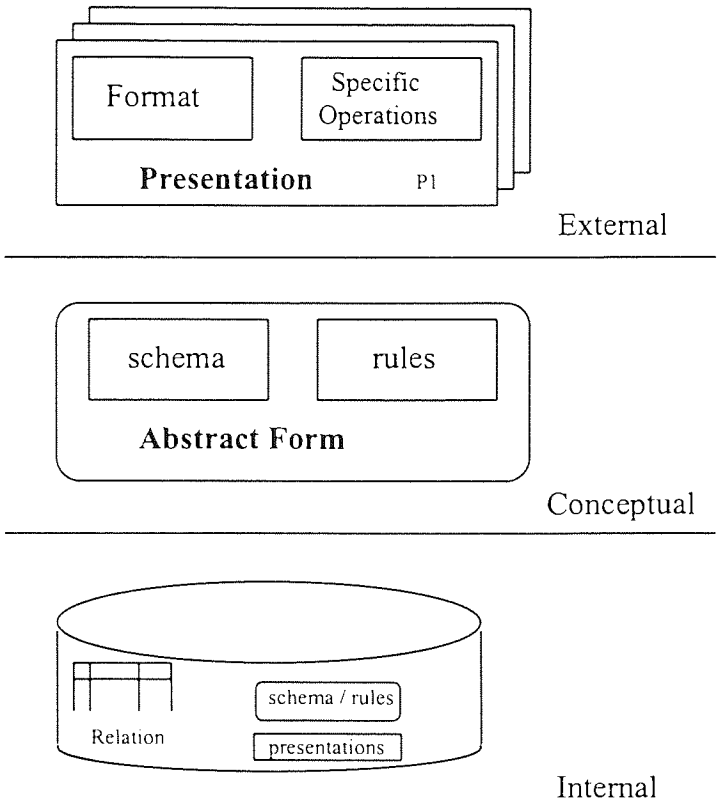

Figure 9. 4 : Model of form architecture after Adiba and Collet

9.4.2 Differences and similarities between forms and reports

Although at the margin many counter examples could be found, forms are typically used to collect data and to modify data. They are the physical equivalents within the information system to real world objects, and their movement and use will track real world events of interest to the information system. We are so used to forms that an "order form" has become equivalent to an "order" but it is only necessary to observe an open air market in action or to discuss the issues of contracts with a lawyer to confirm that the "order form" is only an attempt to track a transaction in the real world.

Reports on the other hand represent consolidated statements about the information system and hopefully about the real world. Even if the report cites individual objects or transactions it is nearly always an aggregate view. Thus "best salesman" or "worst payer" are defined in relation to the body of salesmen and customers even if they focus on individual instances.

Thus it is possible to suggest that the form and the report map to the different management levels as in the diagram.

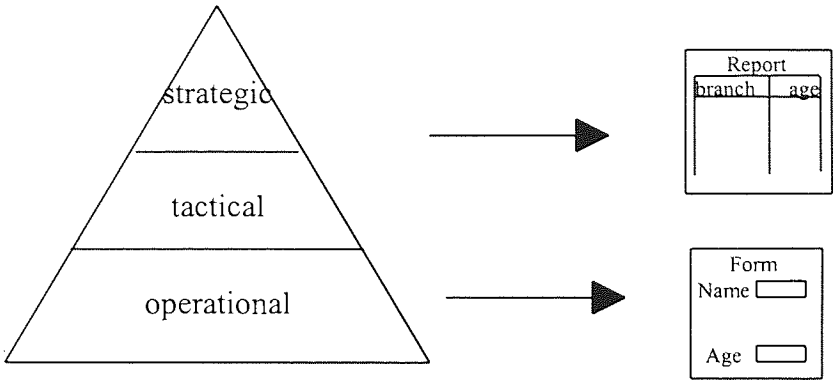

Figure 9. 5 : Mapping the organisation to the paradigms

9.5. ROLES OF FORMS AND REPORTS IN ORGANISATIONS

Both forms and reports have a conceptual information content and one or more presentations. There can be many valid versions of the same form or report and a form manager or report generator should allow a number of transformations or restructurings which maintain the information content. However, because forms are intrinsically of a lower dimensionality than reports, they are usually based on a single record and different versions will be achieved mainly by restriction of content or by changed layout. Restriction involves either suppressing some of the base data e.g. not displaying salaries on some employee forms, or by limiting the functionality. Thus some forms may have no edit access. Layout changes will vary from simply changing type styles and layout to changing the language of the fixed text on a form to reformatting of date display into UK date format.

Reports on the other hand are intrinsically multidimensional, and there is thus always a tension in mapping them to a two dimensional medium which permits many different presentations of the same information. These alternative presentations are not neutral but will have a purpose. Thus a presentation aimed to highlight potential managers might be different to one concerned with productivity bonuses even if the logical information content is the same.

Employee Analysis			
Gender	Branch	Number	Average age
Male			
	Birmingham	20	54.5
	Coventry	10	58.3
Female			
	Birmingham	25	37.6
etc.			

Figure 9. 6 : Alternative Presentation of Report

9.6. FORMS, REPORTS AND DATA

It is clear that a report or form consists of two main elements: a structure and a set of data, and whereas in the case of a form its instantiation or actualisation consists of the filling of the blanks in a selected presentation by the current data, in the case of a report there is likely to be a certain amount of calculation (sometimes very elaborate calculation) and transformation of the original data through a selected presentation to produce an instance.

Until recently the main constructs of a modern information system would appear to be forms, reports and data tables. All of these predate computerised information systems and would appear to be as old as the organisation and recording of data and administrative processes. To what extent the conventional structures they have adopted are implicit in the nature of things or a function of the tools and techniques available for recording information is not clear. What is apparent, is that despite the considerable advance in many aspects of data processing these fundamental paradigms have remained unchallenged. It is possible that a stable system of concepts has been constructed which are mutually supportive so that no great change could take place in any one element without changes in the other two.

The technology is now in place for liberating Information Systems from the straightjacket of the table, form and report. Object database technology and hypermedia offer an alternative higher dimensional, richer set of tools for managing and viewing data and diagrams, graphs and charts are becoming increasingly popular as the technology for their production has improved. Whether these will result in a paradigm shift is to be seen. However this research whilst adopting the richer modelling possibilities of the object model, is less adventurous as far as the statistical report is concerned.

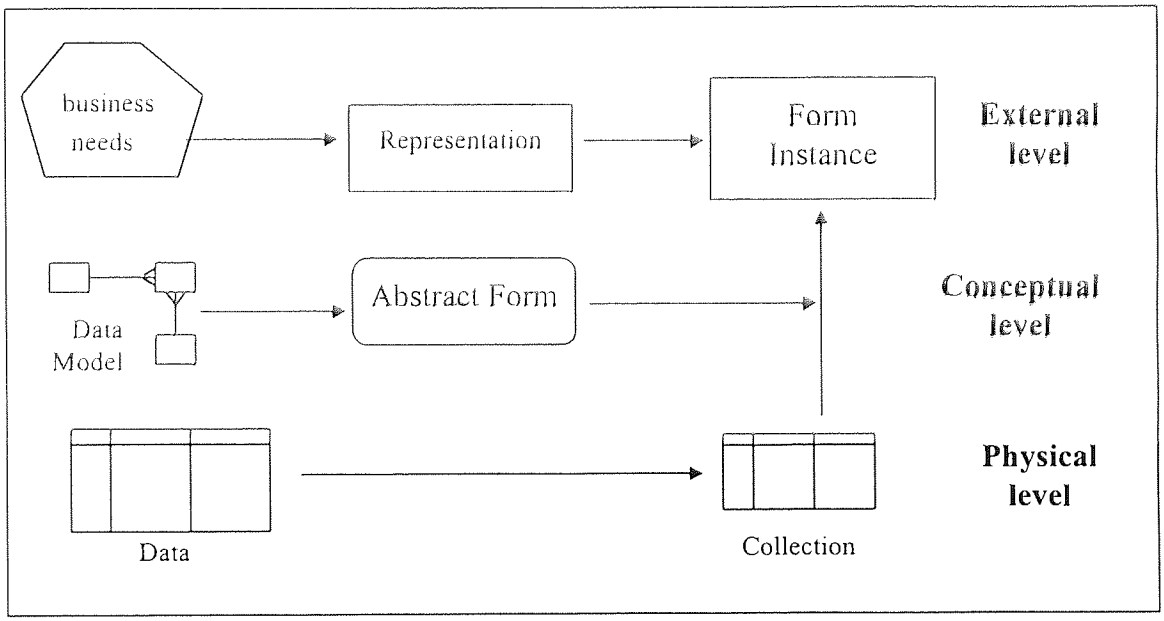

Figure 9. 7: Instantiation of forms

9.7. CONCLUSION

The interface between users and an information system is dominated by two main constructs: the form, central to office automation and the proceduralisation of events in the real world, and the report, historically the main tool for the creation of management information systems.

Whereas forms are relatively simple in structure but essentially dynamic in use, reports can be extremely complex but are generally static.

Despite the long history of report construction and generation there has been little work on the conceptual framework for reports: rather a plethora of report

generators and report generation languages.

The specification of "management information" has been developed outside of the MIS research community, mainly within the statistical community, where there has been a longer tradition of formalisation of complex information structures.

To develop a conceptual framework for discussing report construction it will be necessary to examine the literature on statistical summaries and statistical summary tables. This will be addressed in the next chapter.

Chapter 10

Statistical Summaries

10.1. INTRODUCTION

Most of the research on what MIS users call reports has been carried out within the statistical database community. The reason for this is evident: whereas report generators exist which satisfy most MIS users needs, the problems associated with the management of large statistical databases demand support well beyond the traditional report generator. Such databases as the UK Census returns, by virtue of their size and complex structure, require a distinct set of tools if users are to be given access to the information contained in them. Researchers in this area have generally accepted the need for, and existence of, statistical summaries such as the analysis of population by location, gender and age and have attempted to model such summary tables with a view to developing applications which can give the required access. The data sources are in general seen not as the raw instances or transactions which are being summarised, but summaries or abstracts based on the analysis and aggregation of survey and questionnaire results. Because of the complexity of the data sources, an issue which has not been so important in the discussion of MIS reports, that of the metadata which describes the structures being managed, is much more important and has received a lot of attention. Although the problems of statistical databases would seem far removed from the every day world of management decision making, the volume of data being collected by modern transaction processing systems in many organisations will exceed the information collected by public statistical agencies. The type of analysis and presentation of aggregate business data in an MIS is no different to that used by national statistical offices and the problems of information about data are also very relevant to the decision maker trying to filter, analyse and understand the information which is available to them.

In this chapter, the state of the art in statistical database modelling and the management of summary tables will be reviewed to consider what elements from the research area are relevant and portable to a business decision making

environment. A suitable starting point is the model developed by Su (1983) "SAM*: A Semantic Association Model for corporate and scientific-statistical databases" which not only is one of the earliest comprehensive models in the literature but also one of the few which claimed relevance to the MIS field. Other contributions, particularly in the manipulation of statistical summary tables, which have complemented Su's work will also be reviewed.

10.2. SAM*

SAM* is a semantic association model developed by Su (1983). SAM is built up from seven basic constructs.

10.2.1 Membership Association

An attribute type is defined as a set of similar atomic concepts. Such a set is denoted by **M** in the graphical version of the SAM model. For example, a range of data representing the age of employees could be considered a membership set, the atomic concepts being the integers {20...65}.

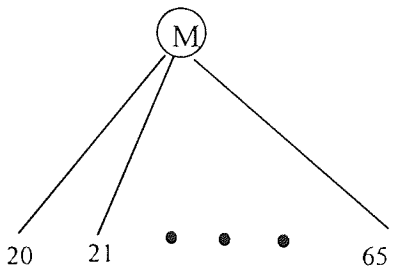

Figure 10. 1 : Membership association

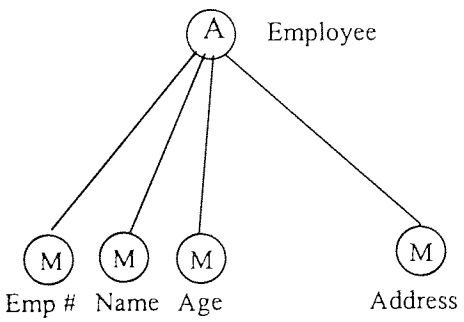

Figure 10. 2 : Entity Types

10.2.2 Aggregation Association

An Aggregation is a named group of concepts, labelled A. Where the concepts are themselves attribute types an aggregation is called an entity type. An occurrence of that type is a member of the Cartesian product over the attribute types. For example an Employee consists of the record information Emp#, Name, Age etc. and this is represented in figure 10.2.

10.2.3 Generalisation Association

A *Generic* association is a group of concepts which defines a more general concept. This becomes a generic type and the union of the members are known as generic occurrences. Such nodes are labelled with a **G**. For example if the company has two types of projects domestic and foreign then Project is a generalisation of Domestic and Foreign Projects

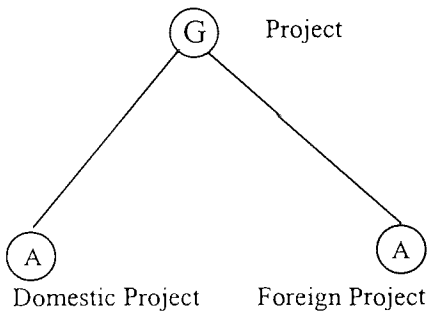

Figure 10. 3 : Generic Types

The model permits the definition of the way in which the generic relation is composed of its member sets. Set Exclusion means that concepts cannot be members of more than one type forming a generic type. Set Equality means that concepts must be in each participating type, Set Intersection allows for multiple membership and Subset Constraint means that members of one set must be members of an other participating set.

10.2.4 Interaction Association

Where independent entity types participate in a connected event there exists a relationship occurrence and the set of such occurrences is a *Relationship* type.

Concepts formed by this association are labelled with **I** nodes. The staffing of projects is a relationship between a project and an employee as shown in figure 10.4.

10.2.5 Composition Association

Where distinct concepts are assembled to create a new whole there exists a Composition type denoted by **C**. A monthly report is assembled by bringing together the Project Progress assessment and the Financial Position statement.

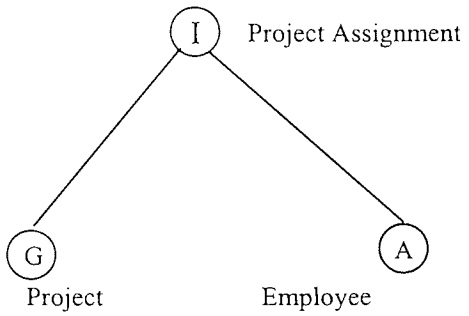

Figure 10. 4 : Relationship types

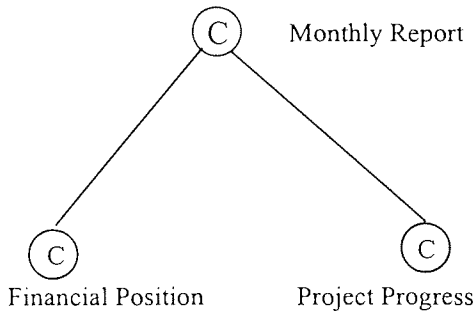

Figure 10. 5 : Composition Types

Composition can collect not only different types together but also instances of the same type to create a class of objects. A portfolio of projects is a collection of individual projects.

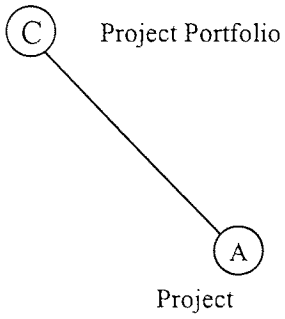

Figure 10. 6 : Composition to create a class of objects

10.2.6 Crossproduct Association

A type created by taking the crossproduct of the occurrences of two or more component types is called a crossproduct type, denoted by **X**. A crossproduct of a group of employees can be obtained by breaking down the data by age and sex.

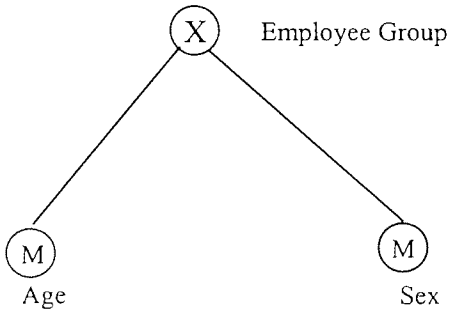

Figure 10. 7 : Crossproduct

10.2.7 Summarisation Association

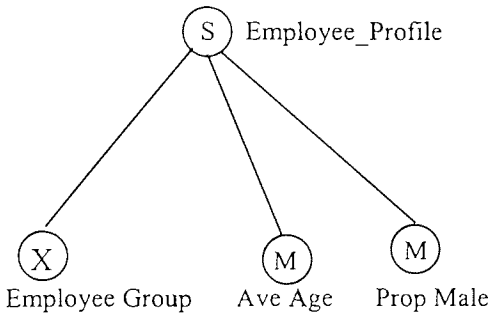

Figure 10. 8 : A summary type

A summary type is a concept based on a set of objects and a group of attributes which summarise the set. These summary attribute values are measured data on

which statistical analyses are usually performed. An **S** is used to denote summary types.

10.2.8 Conclusion

Su (1983) has developed a set of modelling constructs which go beyond the usual set used to model the real world and include some constructs specifically designed to model the information in the information system.

To identify the contribution of Su's work it is possible to compare the seven associations he has defined with the modelling constructs developed in earlier chapters

SAM* association	equivalent constructs
Membership Association attribute type	Domain
Aggregation Association entity type	Attributes Aggregation (Smith 1977a)
Generalisation Association generic type	Generalisation (Smith 1977a) supertype/subtype
Interaction Association Relationship Type	Relationship (ER model)
Composition Association composition type	is-part-of (aggregation)
Crossproduct Association Crossproduct Type.	?
Summarisation Association summary type	?

Table 10. 1 : SAM* concepts compared with traditional data modelling concepts

It is thus apparent that apart from giving a homogeneous structure to the graphical representation of the data model, SAM* adds only two main concepts and these are the crossproduct and summarisation associations. However it may also be argued that the composition association is used in a different way to the is-part-of abstraction in the description of data concepts.

Both the crossproduct and summarisation associations of Su are in fact standard statistical constructs which Su has succeeded in integrating into a wider data modelling framework. Su is not the only writer to have addressed the issues of crossproduct and summarisation which are generally referred to as summary tables with the crossproduct generating the table structure and the summarisations filling the body of the table.

The theory of statistical summaries, such as it is, will be discussed next and then the work of other authors who have contributed to this area of research.

10.3. THE THEORY OF STATISTICAL SUMMARISATION

Although there is a huge volume of literature on both the theory and practice of statistical method which includes the calculation of statistical summary measures (Yule 1957), there appears to be no data on the semantics of aggregation of information. Much of the literature confounds presentation and structure with information content. Hence there is a large literature on the modelling of statistical summary tables which will be relevant when we need to discuss the externalisation of summaries. The following discussion and review will attempt to build up a model of the summary process from first principles.

10.3.1 Instances and collections

In the CORD model of the real world a distinction has been made between entities or objects and their instances. This distinction between the intension of a concept and its extension is essential to data modelling but frequently glossed over or inadequately distinguished such that analysts are frequently confused as to whether a particular construct refers to properties of the whole class (including all future and past instances) or to the extant members of the class within the current Domain of Discourse.

Quoting Eckert (1994b) "Collections denote groups of objects currently present while types give the behaviour of these objects". A consequence of this is that

collections vary in time whereas the data model defines a set of possible states, and hence delimits possible realities. It does not represent the current situation of the organisation. The instantiated data model is the current representation of the organisation and as such is the focus of interest of managers. However, as has been discussed above, the set of collections of all instances stored in the data model is of little interest to anyone. Some sets of instances may be relevant at the operational level to operators and their supervisors e.g. the current contents of the warehouse. However in general it will be summaries of these collections which will be used to support management decision making.

It is useful to model the simplest of such summaries, the count of the number of instances in a collection and the sum of the values of an attribute.

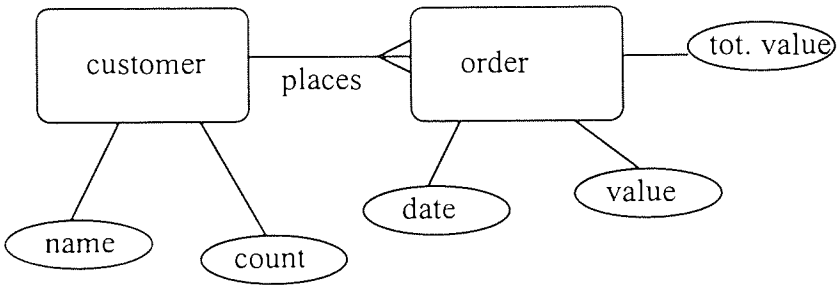

Figure 10. 9: Confusing data model

The example given shows the confusion that arises when one tries to model summary or aggregate values with a normal data model. Here *count* is intended to represent the number of customers and *tot. value* the total value of orders placed. *Name* and *date* are attributes of instances of the classes represented by *customer* and *order* whereas *count* and *tot value* are attributes of collections and will be inevitably temporally defined. For example "the number of customers at the 1st April 1998, total value of orders placed during the first quarter 1998".

It is apparent that there is a need as recognised by Su (1983) to distinguish between the class or type, an instance of the class or type and a collection of instances of the class or type. Just as normal modelling permits the specification of the attributes of members of the class, so it is necessary to specify attributes of collections. There may of course be many different collections constructed

according to a variety of selection rules from the set of all instances of the class. The largest set of current instances will be called a base collection when it is appropriate to distinguish it from other collections, and all other current collections will be sub-collections of this base collection.

There are two other classes of collection which will not be discussed at length here but are worth identifying to distinguish them from the current collection. One group is that of current collections temporally defined such as "total value of orders placed in 1993". The other is that of collections defined historically, "all past customers". Such collections will be based on the same definition process as current collections but with either non current time definitions or a range of time.

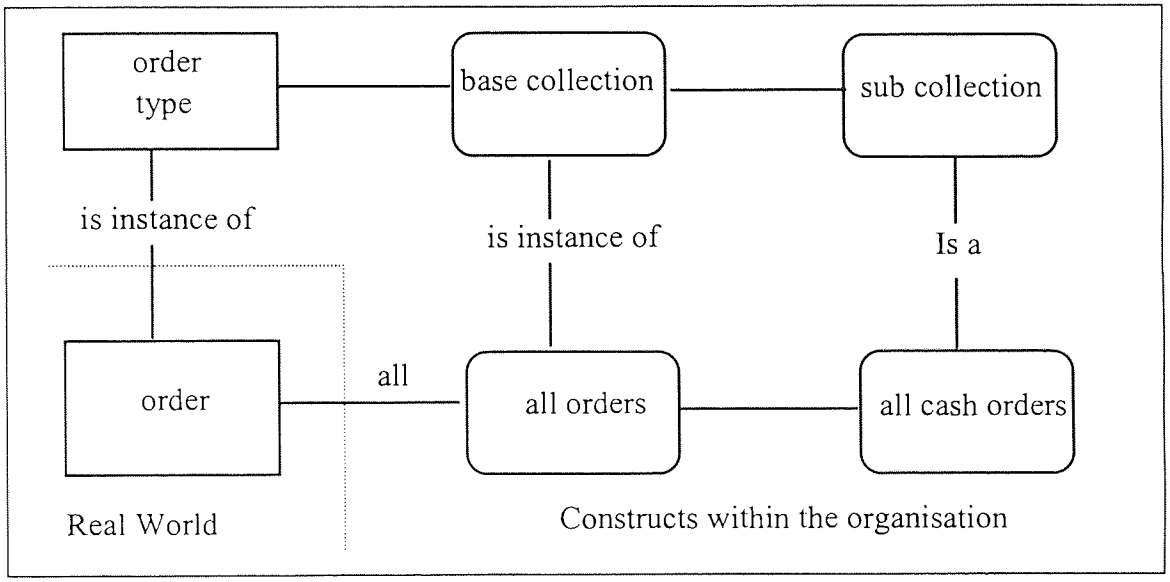

Figure 10. 10: Relationships between classes, collections and instances.

In some cases a collection may be a significant object within the DoD, such as cancelled orders, however they will usually be defined intensionally as a sub type. Very often collections are only objects within the organisation or within the information system of the organisation. As the intention here is to build a conceptual model of the information system this distinction has not been made; some elements of the conceptual model will be implemented within the information system whilst others will remain outside within the manual system or the ad-hoc computerised sector.

10.3.2 Attributes of collections

Like any object, instances of collections can have attributes and be related to instances of other collections.

Attributes of a collection fall into two classes: those associated with the definition of the collection itself, and those which are a function of the attributes of the instances collected.

10.3.2.1 Attributes associated with the definition of collections

The attributes of the collection process will include the date and the selection process which derived the collection members. Thus a collection may have a description "All orders paid for in cash during the first quarter of 1994 as enumerated on April 12th 1994". It is possible to identify several attributes which apply to all members of the collection, "payment in cash", "during first quarter 1994". The date on which the collection was enumerated is relevant if errors or delays in reporting orders exist, in which case collections could differ depending on the information available when they were elaborated. Typically these attributes are true of every member of the collection. However it is possible to envisage attributes which are more true of the collection than true of each member, for example, "Five largest cash orders", "A random sample of orders for auditing purposes".

Attributes of the instances constituting the collection can be broken down into two distinct groups, the first being the identity of set members, e.g. a list of Order Ids. The other is the attributes of those instances, this is the result of a typical data base query e.g. "SELECT value FROM order WHERE payment_mode = 'cash' AND date = Q11994;".

10.3.2.2 Attributes as summary measures

The second group of attributes are summary measures describing the collection as a whole, that is, aggregates of the instances.

In most information systems there are three basic types of variable (Lawson & Golder 1991, Lakhal & Cicchetti 1990) Nominal, Ordinal and Numerical. Attributes will in general be of one of these types. Most DBMSs will specify attributes in terms of their physical storage rather than the underlying data type. Thus Oracle basic data types are *date*, *number* and *text*. As this discussion is concerned with conceptual models neither the external representation nor the physical storage are relevant. However it is important to note that information systems frequently hold information which is unstructured text *memo* fields, and will more and more be called upon to store sounds, images and video clips.

Summarising Nominal data

A nominal attribute, frequently called a category attribute, classifies an instance and is an element of a domain (enumeration) such as:

```
Method of payment = {cash | credit card | prepayment | on
account}
Order received = {post | telephone | fax | salesman}
```

Nominal data is usually summarised by the construction of a frequency distribution, counting the number of occurrences of each domain element.

Summarising Ordinal Data

Ordinal data can be found in organisational information systems in two forms: the first where a set of instances are given a ranked ordering (with or without occasional ties), and the second where each instance is given a score drawn from an ordinal scale ("like a lot", "like a little", "don't care" etc.). Both types can be summarised by partitioning the ranking into two or more groups and counting the number of instances falling in each group. Where the user chooses to ignore the basic nature of the data, then arithmetic summaries are often calculated but the validity of these is questionable. Their use reflects a lack of familiarity with the appropriate technique, and often a lack of suitable tools to carry out the summarisation process.

Summarising Numerical data

Although the attribute in the real world may be real such as length, the attribute as recorded in the information system will be nearly always integer or fixed decimal. There is a whole body of statistical methods for the summarisation of numerical variables (Yule 1957) and despite the range of statistical measures available, the main ones in use are the sum and the arithmetic mean. Occasionally percentiles are used, such as the median and the 90th percentile, when an upper limit on a range of a variable is required. A further technique used to display the variability of a numerical variable is the collection into an ordered set of classes and the construction of a frequency distribution (age 10-20; 20-30; etc.).

In this work the main interest is the standard reporting used routinely by managers. Only summaries achieved by summation or averaging or by regrouping will be considered. The whole range of other summary and statistical techniques such as regression, correlation, timeseries analysis, will remain for others to consider in detail. As the core operations in these techniques are the summation of squares and crossproducts they should also be ameanable to the same treatment as more simple summaries.

10.3.3 Domain hierarchies

Summarisation can occur in the instance dimension as suggested for each variable type above. This results in the same variables and domain elements but a lesser number of instances. In the case of the arithmetic mean one value summarises any number of instances. In the case of Nominal variables there are as many values in the summary as distinct domain elements.

Alternatively, summarisation can be achieved by the restriction of the range of the domain. This is what happens when a numerical variable is grouped into classes. Firstly there is a domain restriction applied and then a summarisation over instances. To achieve such summaries a domain hierarchy is established which maps several domain elements of the original variable to a single domain element

of the summary variable. Although such regrouping may be completely arbitrary there is usually a semantic content to the regrouping process.

10.3.3.1 Classifications

In many fields the use of domain hierarchies is well formalised into standard classifications. Such classifications like the European classification of activities (NACE 1975) identify attributes of individual instances, and usually provide a standard coding for these attributes and a hierarchy defining how higher level concepts are made up from basic ones. Thus the example drawn from the NACE shows the domain of the attribute of a enterprise - its industrial activity - and shows higher level groups of activity into industries and sectors.

3	Metal Manufacture; Mechanical, Electrical and Instrument Engineering
31	Manufacture of Metal Articles (except for mechanical, electrical and instrument engineering and vehicles)
311.	Foundries
311.1	Ferrous metal foundries
311.2	Non-ferrous metal foundries
312.	Forging; drop forging, closed die-forging, pressing and stamping
312.1	Forging; drop forging, closed die-forging
.11	Forging
.12	Drop-forging, closed die forging
312.2	Stamping, pressing (drop stamping)
313.	Secondary transformation, treatment and coating of metals
313.1	Manufacture of articles on metal turning or forming machines (screws, bolts and nuts)
.11	Manufacture of articles on turning machines or lathe, including the manufacture of turned screws

Whilst many domain hierarchies may be specific to a particular analysis and problem, many are formalised classifications which enable comparisons to be made between collections of objects over time or from different sources and are thus very important for many MIS activities from exception reporting to strategic planning.

Sato (1989) discusses in detail the different types of category hierarchies and classification hierarchies showing that these hierarchies are represented in

semantic models by "part of ", "a kind of" and " is - specialisation of" relationships. Su (1983) does not make these distinctions and uses his Generalisation Association (G) to describe classification hierarchies.

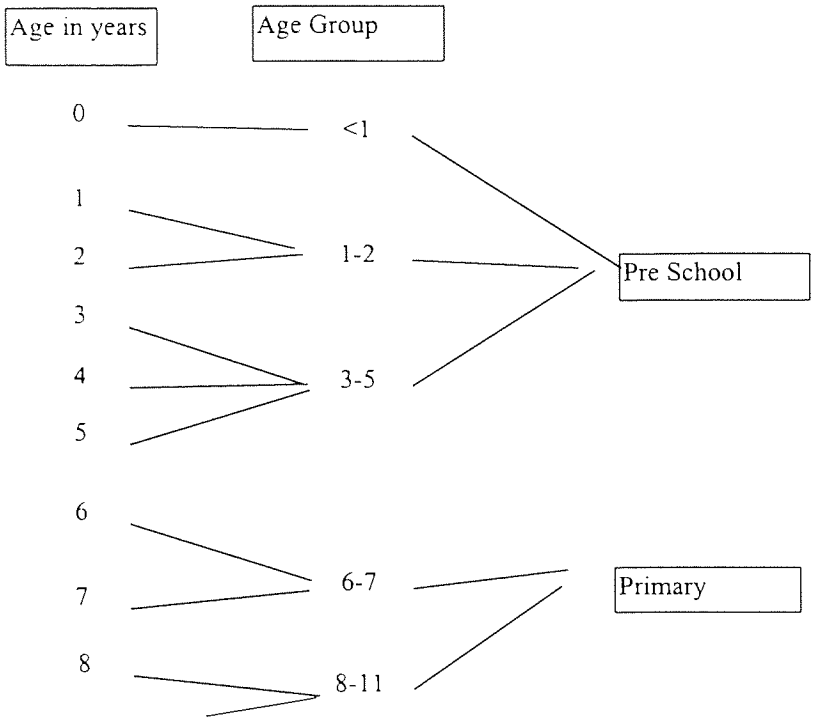

Figure 10. 11 : Domain Hierarchy

10.3.4 Summarisation of data

It has been argued that the creation of a management report (known in this context as a statistical summary table) is based on a small number of functions defined over a collection of objects. The result of applying these functions is to produce a new object with attributes all of which are aggregates of the attributes of the instances making up the collection, plus attributes which are part of the identification and selection process which generated the collection. To distinguish this new object from other objects in the model which relate to single concepts in the real world it is called a Statistical Object (SO) by Rafanelli & Shoshani (1990). The distinction between a *statistical* database and a *normal* database is that normal databases traditionally hold instance data and generate aggregations only for reports which are outside the data base, whereas statistical databases retain aggregated data. Thus in a corporate context any decision support system

or Executive Information System which retains aggregated data will have the same features and face the same problems as a statistical database. In a dynamic database such as that belonging to an organisation's transaction processing system, there is a consistency problem arising from the retention of summaries. Currently this is being addressed by the creation of data-warehouses.

10.4. STATISTICAL META DATA.

One of the fundamental distinctions between a statistical object and a normal object is the role of its qualifying attributes.

An employee record may show:

```
Employee Number     123
Age                 23
Sex                 Male
Blood Group         A-
```

If the blood group information is omitted, the record is still referring to the same person: it is just that there is less known about them. On the other hand with a statistical object, say the number of employees with different blood groups

```
Blood Group   A-
Sex           Female
Count         25
```

Then the loss of one item of information about the object renders it as having no information content. If the Blood Group or the Sex attribute is omitted then it is not known what these 25 employees have in common. If the Count is omitted then it is not known if there are any female employees with Blood group A, so the information is useless.

Because of the fundamental importance of data in identifying the statistical object in this way it is usual to refer to as "Metadata" that data which defines the object. The importance of metadata in the management and use of statistical information has been stressed by many authors (for example, McCarthy 1982).

10.5. STORM

Rafanelli and Shoshani (1990) identify weaknesses in earlier proposed models for statistical summaries and suggest their own model. Recognising the wide range of summary data that may be produced by and stored in a statistical database (tables, matrices, time series etc.), they consider a statistical database to be a collection of *statistical objects* and propose STORM (STatistical Object Representation Model) as an appropriate modelling tool to describe such objects.

The basic concepts underlying all statistical objects which they identify are:

Summary attributes - attributes that describe the quantitative data being measured or summarised e.g. "Population".

Category attributes - these are attributes that characterise the summary attributes, for example "Gender" may categorise "Employees".

Multidimensionality - a multidimensional space defined by the category attributes associated with a single summary measure e.g. "Employees" may be characterised by "Division", "Gender" and "Year".

Classification hierarchies - a classification relationship often exists between categories. e.g. "Departments" can be classified into "Divisions".

10.5.1 Representation of statistical objects

Rafanelli and Shoshani (1990) discuss current methods of representing Statistical Objects (SOs) in deriving proposals for the graph based representation of the STORM model.

The practice of showing a table in its two dimensional representation clearly presents many problems.

- It confuses an instance with a type (if the cells of the table are filled).
- It confuses a particular 2 dimensional presentation with the general description of a multi dimensional structure.

- It fails to identify different levels of metadata. In particular, levels of classifications are not adequately distinguished.

They also criticise the use of a graphical representation as in SAM* because it typically mixes categories and category instances. Like the tabular representation, it rapidly becomes large and difficult to fit on single sheets of paper and as a modelling tool has a vital communication and presentation role. This is a non-trivial criticism.

The conclusion of the authors is to distinguish the meta-level (intensional representation) from category instances hence restricting the depth of a graphical representation.

10.5.2 The STORM model

A Statistical Object is defined by the quadruple:

<N,C,S,*f*> where:

N is the name of the SO which gives the SO an identity and may well identify its purpose e.g. "Age of Employees".

C is a finite set of category attributes.

S is a single summary attribute associated with the SO.

f is a function which maps from the Cartesian product of the category attribute values to the summary attribute values of the SO.

The notation N (C1,...,Cn: S) is proposed for an SO thus the following could describe a particular statistical object:

```
Employees (Division, Gender : Average Age)
```

In most cases the mapping function *f* is implicit and will not need to be defined further.

Because there is a single summary attribute in a STORM model the graphical representation will take the form of a directed tree with the summary attribute (S) at the root and various aggregation nodes (A) bringing together the constituent

category attributes (C) below the root.

10.6. MEFISTO

That there is no clear conclusion to the issue of statistical modelling is indicated by the appearance of yet another proposed model: Mefisto by Rafanelli and Ricci in 1993.

Whereas SAM* attempted to integrate statistical and conventional data modelling into a single framework, STORM concerned itself with the modelling of the statistical object and, without taking a position on traditional data models, identified the need to distinguish between the metadata level and the category instance level or domain level in modelling statistical objects. In Mefisto the focus is on the manipulation of statistical entities (identical with statistical objects in STORM).

The authors identify the following operations which can be performed on a statistical entity. The following discussion will use the statistical entity:

```
Employees (Division, Gender,: Average Age)
```

as an example.

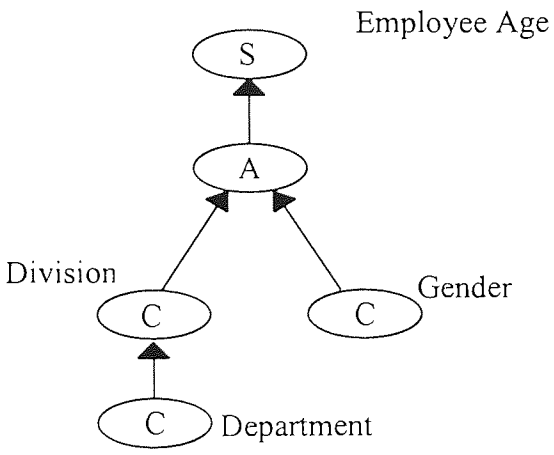

Figure 10. 12: A STORM Model (Rafanelli & Shoshani 1990)

10.6.1 Summarisation

Given a statistical entity it may be wished to reduce its dimensionality by collapsing over one of the category attributes:

```
Employees_DG (Division, Gender, Average Age)
 → Employees_D (Division, Average Age)
```

This process they have called summarisation. The exact nature of the process will depend on the type of summary attribute (it may not be possible in some cases).

10.6.2 Classification

Where a hierarchical classification exists it may be required to merge certain categories to achieve the next level up in a hierarchy. Thus:

```
Employees (Department, Gender,: Average Age)
→ Employees (Division, Gender,: Average Age)
```

The same issues of aggregation arise as in the previous operation.

10.6.3 Restriction

When the need is to select out only a subset of the categories for reporting this is a restriction:

```
Employees (Division, Gender,: Average Age)
 → Employees (Division, Gender: female,: Average Age)
```

As no new data is generated, no issues of aggregation arise.

10.6.4 Enlargement

Where a statistical entity is the "union" of two compatible entities then the process is called enlargement (the inverse of restriction):

```
Employees (Division, Gender: male,: Average Age)
 + Employees (Division, Gender: female,: Average Age)
 → Employees (Division, Gender,: Average Age)
```

It is obvious that this can only produce a valid statistical entity when the

components are parts of an aggregation hierarchy or elements from a common domain.

10.6.5 Extension

It may be required to raise the visibility of an item of metadata common to the whole of the entity by turning it into a category attribute. This process is called extension:

```
Employees (Department, Gender: Average Age)
 → Employees (Year: 1994, Department, Gender: Average Age)
```

No new information is generated but a previously implicit feature of the statistical entity (that it described employees in 1994) is now made explicit.

10.6.6 Renaming

A category attribute can be given a new name:

```
Employees (Department, Sex,: Average Age)
 → Employees (Department, Gender,: Average Age)
```

10.6.7 Conclusions

Whereas the operations defined in Mefisto are a contribution to the definition of the conceptual model by the specification of valid operations, it is not evident that a consistent conceptual level has been adopted. In particular Extension is not a valid conceptual operation being mainly a presentation issue as is Renaming. The other operations however are essentially functions of category hierarchies, except the Enlargement operation which does however have to be consistent with existing hierarchical structures.

10.7. MAPPING TO THE RELATIONAL MODEL

Many authors, Lakhal et al (1989), Ozsoyoglu et al (1989) in discussing statistical summaries have been concerned to integrate their work with the relational data base framework.

There are several reasons for this:

1) the raw data is likely to be stored in a RDBMS
2) the metadata is likely to be stored in a RDBMS
3) the instantiated summaries are likely to be stored in a RDBMS.

On the face of it none of these issues are of direct concern as an OO conceptual model has already been adopted. However an underlying architecture has been developed which may be applicable in an OO environment as well as a relational environment.

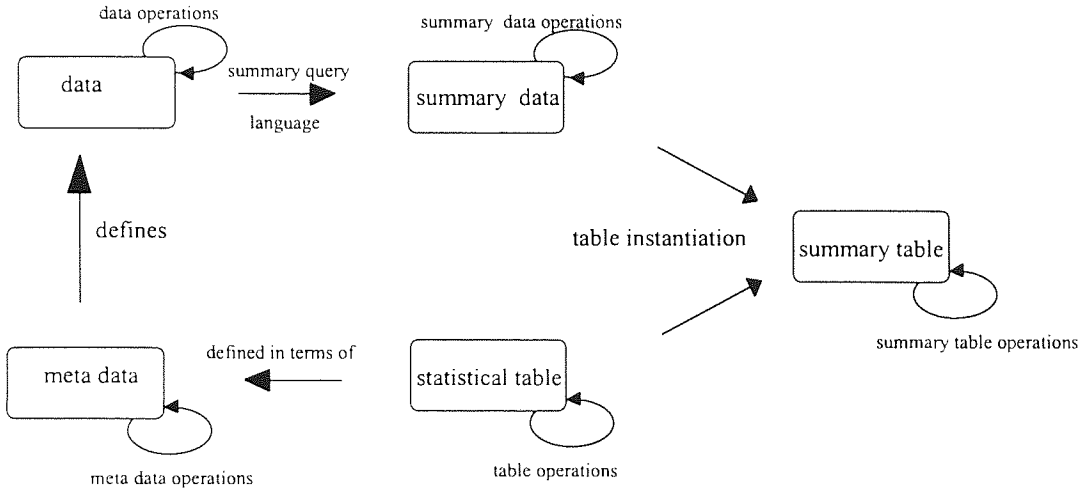

Figure 10. 13: Elements of a summary table management system

Combining the models of Rafanelli (1990) and Adiba & Collet (1988) it is possible to define two distinct environments. There is the statistical data space and the statistical entity. For each of these environments there will be a distinct conceptual model and a set of valid operations which can be performed on objects in that space and there will be operators for transforming data from one space to another.

10.7.1 The statistical relational model

Ghosh (1988) has argued for an elaboration of the relational model as a framework for managing statistical data.

A statistical data set can be mapped to a relational table as in table 10.2.

Employee	Male	Female
Full Time	50	5
Part Time	5	50

→

Contract	Gender	Count
Full Time	Male	50
Full Time	Female	5
Part Time	Male	10
Part Time	Female	50

Table 10. 2 : Mapping Summary table to Relational table

Ghosh proposes a set of operators which extend the relational algebra of Codd (1979) in three ways:

1) an extension of the relational algebra on columns.
2) an extension of numerical algebra on rows
3) a set of operations between multiple tables.

Select-power (x) -aggregate : This operation extends the normal select-group-by clause allowing the sum of powers of selected numerical variables to be computed within the Cartesian product of the selected categories.

Select-product-shift (x) -aggregate : This operation calculates crossproducts for pairs of selected numerical variables which differ by a given number of columns (shift). This is essential for determining correlations between variables.

Aggregate-distribution : Sums frequencies within cells defined by category variables, which would appear identical with *Select-power (1) -aggregate* with the ability to generate a new attribute to hold frequencies if this does not exist.

Join-power (x) -aggregate : Offers the same functionality as performing a relational join on two tables and then applying Select-power (x) -aggregate.

Relational-set-recursive-function : This is a specialised function for doing time series analysis.

It is not evident that this is a particularly economical way of proceeding. The

main contribution is a function which creates summary tables with aggregates of the base numerical variable. The other functionality can be achieved by using ordinary relational query language to produce an intermediate table with squares, crossproducts etc. and then using the aggregate function to sum the values.

10.7.2 Operations on summary tables

Ozsoyoglu et al. (1989) describe a database query language for summary table-by-example query processing. They discuss the extended relational algebra needed to manage complex data and two summary table operators needed for the query processing.

Pack : The pack operator converts from fully normalised structures to repeating groups. Thus if the relation:

```
ASSIGNMENT (Employee, Project)
```

represented the many-many relationship between employees and projects, then the effect of *packing* over project would produce a relation with one tuple for each employee and a repeated group of projects.

```
Pack (ASSIGNMENT, Project) → ASSIGNMENT* (Employee,
*Project)
```

Unpack : The unpack operator reverses the process and normalises a relation with repeating groups.

Aggregate formation : Computes aggregates for mutually exclusive groups based on the category attributes.

Aggregation by template : Generates aggregates based on a grouping supplied in another relation.

Directional Join : Produces empty cells in circumstances where cells would not otherwise be produced, this is similar in logic to an outer join.

Construct : Generates a single column relation corresponding to the category

instances either by simple enumeration, e.g. (Male, Female) or by defining a for-loop e.g. Integer (20,60,5) for integers from 20 to 60 in steps of 5.

Relation Formation : This function converts a table into the equivalent relation by creating a tuple for each cell with attributes appropriate to its position in the table.

Decomposition : This function converts a complex summary table into primitive (simple) summary tables.

10.8. STAR

Lakhal and Cicchetti (1990) also attempted to integrate the data level and the table level functions into a single system STAR (STatistical And Relational database model). They identify the following main operations on statistical summaries:

Composition / decomposition Simple summaries can be aggregated to produce more complex summaries and complex summaries may be partitioned into more elementary summaries.

Transposition Summaries may need to be reorganised, that is without changing their information content but only aspects of presentation. The authors point out that such reorganisation may have an information content when used to facilitate comparison between different summaries.

Derivation New statistical information may be derived from existing summary tables by standard calculation over ranges of category variables.

In the work of Lakhal and Cicchetti as in the work of Ozsoyoglu et al (1989), a confusion arises between presentation and conceptualisation. To develop an adequate model it is necessary to distinguish between these two levels and to identify which operators are appropriate to which level of analysis.

10.8.1 Simple tables and complex tables

Lakhal and Cicchetti (1990) follow Ozsoyoglu et al (1989) in distinguishing

between simple and complex statistical tables.

A simple statistical table is one in which every cell is characterised by attribute values drawn from the same domains:

	Employee Analysis		
Branch	Gender	Number	Average Age
Birmingham			
	Male	20	54.5
	Female	25	37.6
	Total	45	42.3
Coventry			
	Male	10	58.3
etc.			

Table 10. 3 : A statistical table

In the table reproduced from Chapter 5, the cells have attributes drawn from the domains "Branch" and "Gender" so this would appear to be a simple table. In some of the definitions used, only one summary attribute would be allowed in a simple table so Number of employees and Average Age would each need a simple table to describe them. Moreover the Branch totals are not characterised by Gender so could be considered part of another simple(r) table.

Where two or more simple tables are brought together, a complex table is created. In general there will be some common category attributes, otherwise there is merely a collection of tables with a common theme.

10.8.2 Presentation of tables

The presentation of a statistical table is the problem of mapping a multidimensional object to a set of two dimensional surfaces (paper or VDU screens). It may be that visualisation techniques combined with virtual reality can

raise the dimensionality of the interface, and even the use of colours and symbols can impose more than two dimensions of information on a two dimensional surface. However in general there will be a mapping to multiple two dimensional surfaces. In general there will also be many mappings possible, some of which are essentially rotations in the presentation space; thus an alternative presentation of the employee report might have Gender as a column heading and, Branch, Grade and Average Age as row headings. Table 10.4 gives a more typical statistical table:

Employee Analysis				Gender		Total
Region	Branch	Grade	Age Group	Male	Female	
W Midland	Birmingham	Manual	up to 25			
			25 to 45			
			over 45			
		Clerical	up to 25			
			25 to 45			
			over 45			
		Management	up to 25			

Table 10. 4: Employee Analysis: a 4 dimensional statistical table

Employees				
Employee No	Gender	Age	Branch	Grade
0001	M	24	Birmingham	Manual
0002	etc.			

Table 10. 5 : Underlying data table

In this example, if the regions have many branches a new page may be started for each Region, or staff Grade could be promoted to the top level and the report divided into sections for each Grade. There are thus very many presentations which contain essentially the same data.

Any fully functional summary table system will require operators to specify the presentation, and an interactive interface will require functions for representing the data, equivalent to driving through the data hyperspace. At this stage only the definition of the conceptual table schema and operators for manipulating it are of interest.

10.8.3 Operations on Statistical Tables

Lakhal et al (1989) define a set of commands for creating and manipulating complex statistical tables. They assume as a basis the Statistical Relational Table (SRT) of Ghosh (1988) and then define operators for creating and manipulating such tables from their relations.

Aggregate : This is the basic aggregation command creating a summary table by summing within cells.

Table : Transforms a SRT into a statistical table.

Relation : Creates a relation corresponding to a given statistical.

Concatenate : This enables two summary tables with a common dimension (row or column structure) to be composed into a single complex table.

	White	Black
Male	10	20
Female	5	6

	Employed	Unemployed
Male	5	25
Female	3	8

Table 10. 6 : Two compatible tables

	Ethnic Group		Economic Status	
	White	Black	Employed	Unemployed
Male	10	20	5	25
Female	5	6	3	8

Table 10. 7 : Concatenated table

The proposal by several authors of a concatenation type poses many problems. Whilst evidently it reflects common practice in composing complex tables from more elementary ones, there is no recognition of the role of metadata nor the generation of appropriate metadata for the resultant table. In the example it may be assumed that the counts refer to the same population but it is not necessary, and the definitions which were attached to the two separate tables cannot easily be associated with the one combined table.

Extract : This function extracts from a complex statistical table a specified sub table.

Permute : This operator reverses the row and column presentation of a table.

Displace : This operator changes the order of categories in either the row or column dimensions. Thus if age was broken down within gender, displace could be used to change it into gender within age.

Rotate : In general a category attribute can only appear in one dimension (either row wise or column wise). Rotate permits the movement of a category from one dimension to another.

They also define:

Cut : A select statement producing a subset of a complex summary table by specifying characteristics of the rows and columns desired.

Paste : A function for joining two complex summary tables with identical schemas by summing the equivalent numerical variables.

Aggregate : This operator is used to aggregate a summary table with respect to one of its category variables.

10.8.4 Discussion

STAR, although providing an extensive set of functions for manipulating complex

statistical tables, is very physically defined. Although the authors distinguish between the schema and its extension, the use of a single statistical relational table to hold the data and its metadata mean that these aspects are not adequately differentiated. The lack of a distinction between the instantiated schema and its presentation mean that there remain operators which are too closely dependent on a two dimensional presentation and could not cope with a multidimensional view of the data. The language is thus excessively complex.

10.9. SUMMARY STATISTICAL TABLES

The large amount of research into statistical tables has succeeded in clarifying a certain number of issues:

- The nature of statistical summary tables
- Complex and simple tables
- Operations creating tables
- Operations on summary tables

There is, however, a high degree of confusion between the different levels of abstraction, and a concentration on the definition of statistical tables, to the extent that the relationship between tables and the source data has not received much attention. Moreover, the statistical modelling has been carried out in isolation, and although there is reference to the relational model as the underlying storage manager, there is no attempt to situate the statistical modelling in a wider datamodelling context. There is however enough consensus for the identification of the major issues in the development of an object based statistical data manager.

Chapter 11

The Collections of Objects Summary Table Model

11.1. INTRODUCTION

In Chapter 10 a range of suggestions for the modelling of statistical summary tables was discussed. Noting the several failings of existing models, in that the conceptual levels and instantiations are inadequately distinguished and that there is no attempt to integrate the models in a more general modelling framework, this chapter develops an alternative model which is consistent with the CORD model developed to describe organisational data in Chapter 7.

The new model will be based on the distinction between classes, instances and collections established earlier and will add structures to represent table schemas and instantiated tables. The model is called the Collection of Objects Summary Table model (COST), highlighting the fact that it is intended to develop a model for summarising collections of objects.

As the main focus of this work is concerned with the conceptual modelling level, issues concerning the storage and retrieval of data have been ignored, assuming that whether by normalisation into a relational data base or more directly into an Object Oriented database system such issues can be managed. The same is true for the management of all data with respect to the models being proposed in this chapter. The Object Prototyper will be used to demonstrate the issues discussed, but is not intended to be a serious data management tool for the large quantities of data underlying any management report.

In order to describe statistical tables it will be necessary to specify a suitable schema notation and identify the operations required for schema manipulation. No particular functions will be necessary for the instantiation stage, although it will be necessary to discuss the compatibility issues which will arise from bringing together independently defined objects.

Following Lakhal et al (1989) a relational query language of the type discussed by Klug (1982) or any other language capable of generating summary descriptions for collections for the "summary query language" could be assumed. As there is no intention to distinguish simple from complex data structures in the way which is necessary in a relational model, then the "summary data operations" will be a superset of the "summary query language".

The main elements of any statistical summary model will be those identified by Rafanelli & Shoshani (1990) reviewed in the last chapter.

1) Category Variables
2) Summary Variables
3) Classification Hierarchies
4) Multidimensionality.

Before discussing each of these aspects of the COST model in detail it is necessary to identify the relationship of a COST to the other elements of the CORD model as presented in Chapter 7.

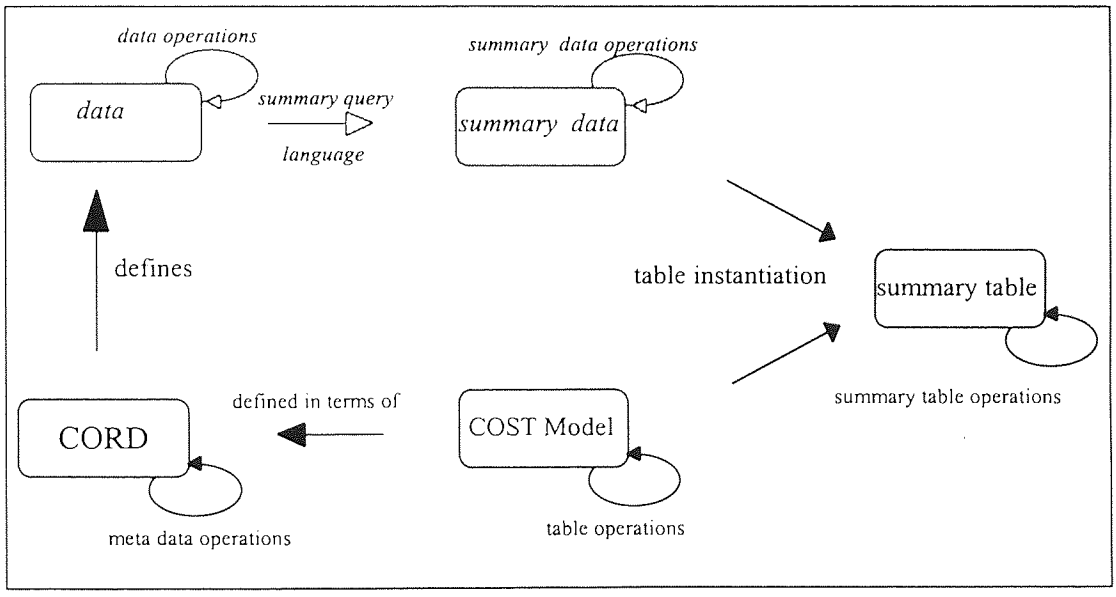

Figure 11. 1 The COST and CORD models in context

11.2. CORD AND COST

An instance of COST is a complex object which will bear a strong relationship to a particular object class. It is possible that a particular COST (say one which has Regions × Age as dimensions) could be used to summarise collections based on many classes; however, in line with the CORD developed in Chapter 7 and to enable adequate validation to take place, an instance of COST will be *owned by* a particular class and can be only used to describe collections of instances of that class. An instance of COST is made up of two main types of element - an abstract definition of the category and summary attributes making an instance of a *Table*, and one or more *Views* being externalisations of the Table. In this chapter the nature of the abstract tables will be discussed. The specification of Views will be discussed in Chapter 13.

In the COST model it is proposed that an instance of a Table should be *owned by* a class and be capable of defining summaries of collections of instances of that class or its descendants in any inheritance hierarchy. Should a class be deleted from the model then dependent Tables would be deleted by cascade. This will permit validation and integrity enforcement and seems intuitive. Users with many similar classes which are not part of an inheritance hierarchy could economise on the construction of Tables by "cutting and pasting" models, but as COST-Table models are very simple there appears few benefits from treating COST-Tables independently of a parent class.

It may be argued that this approach makes it impossible to use COST based summary tables to describe relationships which link instances of two or more distinct object classes. Within the CORD model, relationships are not very important modelling constructs; links between different objects are to be found in the OO constructs of hierarchy, parts of and complex objects, complemented by the new concept of role which instantiates multi object behaviour. It will be necessary to show how the COST model permits the summary of information about these constructs without needing to specify more than one owner class.

A partial definition of the COST-Table is:

```
Class.Table (IsOwnedBy : Class;)
    { Part.Table_Attribute;
          {
            Attribute.Type (Table_Attribute_Type;)
            Attribute.Path (Path;)
          }
    }
Domain.Table_Attribute_Type;
    [
      Category;
      Summary;
    ]
```

Note that a COST-Table can be created for any instance of Class and that it is made up of one or more Table Attributes. Each Table Attribute has two attributes, a Table Attribute Type and a Path. The Table Attribute Type has a list domain with two elements Category and Summary. The path is a primitive datatype, which when evaluated for an instance of the class yields a value which is consistent with the attribute type.

11.3. COLLECTIONS OF OBJECTS

The purpose of the next discussion is to see how a summary table model could handle all facets of the description of a collection of objects. The description falls into three parts: the metadata, that is, all facts which are essentially true for all instances in the collection by virtue of the class from which they are selected, and any common facts due to the selection rule used to filter out the collection members from the class membership, and the summary of the facts which are variable over elements of the collection.

Considering a collection of students, some characteristics will be true because they are students, others will be true because of the selection rule (if any) invoked to generate this collection: thus they may all be over 25 years of age. Finally, there are a range of different types of variable which distinguish the members of the collection, and which in aggregate distinguish (to a greater or lesser extent) this collection from another. The selection criteria and the member characteristics

must all be defined with reference to the domains of the variables; the class characteristics will consist of intensional definitions of the class and any specialisations belonging to this subclass.

Common attributes	The class from which members are drawn
	The selection rule
Other attributes	Individual attributes

Table 11. 1 : Characteristics of members of a collection

11.4. VARIABLES IN CORD

In the CORD model a variable describing an object is defined as a mapping from an instance of that object to a value of a specified domain. The domain part of CORD was developed in a detailed way because it was realised that variables and their domains would be central to the development of COST.

So for the summary structure itself we need to refer to the domain model defined in Chapter 7; the metadata may be tightly defined against the same domain model or may be in formalised textual descriptions of the class intension.

Part of the domain definition from the CORD model is repeated below:

```
Domain;
      {
        Attribute. Class (Class;)
        Attribute. Type (Domain_type;)
       }
      [
        Boolean (Type : primitive;)
        Date (Type : primitive;)
        String (Type : primitive;)
        Integer (Type : primitive;)
        Domain_type (Type : List;)
        [
          Aggregation;
          Classification;
          List;
          Primitive;
        ]
      ]
```

In fact the value of an attribute in the CORD model can be from three different sources. It can be from a defined domain i.e. an instance of the Domain class specified according to the schema above. It can also range over the instances of any class, e.g. any employee. It can also range over the members of any collection. Thus the definition of an extended domain from the CORD model is:

```
extended domain = (domain | class | collection)
domain          = (primitive | classification |
                   aggregation | list)
primitive       = (String | Integer | Date | Path)
```

The review of statistical summaries in Chapter 10 suggests that summaries are made up of category and summary attributes only. It is thus necessary to identify how the variables in the CORD model will map to the definition of category and summary attributes in the COST model.

11.5. CATEGORY VARIABLES

All approaches to modelling statistical tables define an attribute variable, but most are more concerned with defining the characteristics of category variables than with locating them within a general data model. It appears to be assumed that attributes of objects are simply either category variables or numerical.

In developing the COST model it is necessary to situate category variables within the overall data model. Any attribute which has a domain consisting of a moderate number of states can function as a category variable; this will include any attribute of List type (Gender, Marital Status), any attribute which has a set of objects as its domain (Customers, Regions, Employees) and any attribute which has a domain consisting of ordered values (choices, preferences). Thus the process of identifying category variables for a summary table is the process of selecting attributes of the class to function as category variables.

A category variable can be defined as one having the same domain as its parent attribute together with a schema for relating instances of the parent attribute to instances of the category variable, or it can be defined with reference not to the

domain but to the attribute itself. It is this latter alternative that has been chosen for the COST model. However not all attributes can function as category variables, in particular attributes with a high number of states (numerical variables, long strings) will produce very sparse summaries with no useful information content. In many cases this cannot be ascertained until the collection which corresponds to the category variable domain is elaborated. Such validation will not be included in the COST model. In a fully functional EIS based on this research, some plausibility checking might be useful to ensure that a proposed COST model appears to be instantiatable; however this is not included in the present work.

It is important to consider each domain type and ask if a suitable set of categories can be unambiguously generated from the definition proposed.

11.5.1 Primitive types

In general primitive types will not produce categories. Numeric data, character strings, and even dates will often produce too many values with too sparse a density of occurrence to be useful in defining category attributes. There may be valid variables with restricted domains within these types which could generate suitable categories, so this should not be excluded; however as little supportive information exists for such categories in the CORD model they will have their values as labels and respect some primitive ordering.

11.5.2 List

The list is the classic domain for a category variable; the KnownBy property for the list will enable the presentation model to infer labels for the various values and a default ordering.

11.5.3 Class and Collection

Where the domain of a variable is a collection, then the values are ObjectIds, and whenever the number of objects in the collection is modest, this makes a valid

category variable. The KnownBy property of the class of objects in the collection provides labels for each of the states. Conceptually classes cannot be domains and the default collection of all instances of a class should be the domain when a class is cited. However this is counter intuitive, so the CORD model allows class names to appear as domains, with the interpretation that the scope is over all instances of the class, i.e. the default collection of the class instances. In resolving a possibly ambiguous domain reference, the Object Prototyper first checks the class of domains, then the class of collections and then the class of classes.

11.5.4 Domains and category variables

It is clear that for simple attributes with list domains the mapping from the attribute to a summary table category attribute is very straightforward. This will in general be true for collection or class domains; however as has been observed the cardinality may be dynamic, and the appropriateness of an attribute with such a domain type for use as a category attribute cannot be guaranteed. For domains which are primitive however it is unlikely that they would function adequately as category type table attributes.

11.6. SUMMARY VARIABLES

As has been stated, the intention of a COST model is to define a summary model which can be instantiated with an appropriate collection to produce a summary table. The effect of defining a set of category variables means that on instantiation the collection will be divided into sub-collections, one for each element of the Cartesian product of the category attributes. Summary attributes of statistical tables are those which supply a single value for each cell of this Cartesian product. To produce a single value for each cell an aggregate function will have to be applied to the subset of values included in each cell.

Most work reviewed considered summary variables to be numerical variables for which single valued summaries could be defined. This is a very restrictive definition, as can be seen by exploring some examples.

Consider a collection of instances of Employee with the following CORD Model:

```
Employee {
 Attribute. Name (String );
 Attribute. Gender (Gender;)
 Attribute. Age (Number;);
 Attribute. Children (Child; M;);
 Attribute. Grade (Grades;)
 Attribute. Department (Department;)
 Attribute. Picture (Bitmap file;)
 }
```

Previous authors would have identified Gender and Grade as being typical Category attributes and only Age as being a typical summary attribute, but it is more complex than that as the following discussion shows.

11.6.1 Summarising non-numerical variables

It is relatively easy to propose summary measures for non numeric variables.

A nominal variable such as Department can be summarised by:

- extracting the mode and reporting on the most common department for the collection or sub collections defined by the category variables,
- expanding the Department into its domain elements and then presenting a summary such as count for each.

A binary variable such as Gender can be summarised just as a nominal variable:

- or by choosing one state and reporting on the number or proportion of the sub collection in that state.

An ordinal variable such as Grade can be summarised as a nominal variable:

- if only a small number of distinct states have been defined, or by returning the median or other percentile of the sub collection.

11.6.2 Summarising non-traditional variables

There is no restriction to summarising only traditional data types. Newer types such as images can be summarised if a suitable process can be defined. Just as the

police form composite identikit images of suspects from several witnesses' descriptions, so one could produce a composite image for each subcollection; this would be a summary and retain the image data type. Some other variables derived from images, "average inter-pupil gap / brow to nose distance", which could be of a numeric type, could also be defined as a summary. Several statistics could be extracted from unstructured texts from word counts to the number of spelling errors which may in some circumstances be meaningful summary information.

11.6.3 Summary variables in COST

Bearing in mind the above discussion, summary variables in the COST model are defined in the same way as category attributes. A summary variable has a pair of attributes: a type which is set to *Summary* and a path expression. This path expression must evaluate to a single value for a collection of objects. In general this means that at least one aggregate function must be included in the path.

```
Class.Person (IsKnownBy = Name;)
    {
     Table.ptl;
      {
        Table_Attribute.Status (Type : Category;
                                Path : Status;)
        Table_Attribute.Sex (Type : Category; Path : Sex;)
        Table_ Attribute.  Ave_ Age
                          (Type : Summary;
                           Path : age>average (););
      }
    }
```

In this element of the CORD model a table is defined which is owned by the Class Person. It is thus able to define tables for collections of persons. Three table attributes have been defined. Two category attributes:

- Status - corresponding to the attribute Status
- Sex - corresponding to the attribute Sex

and one summary attribute:

- Ave_Age - corresponding to the attribute age of person with an aggregate function *average* applied.

11.7. CATEGORY EXTENSION

As the CORD model allows complex objects and objects as user defined types, an object may have an object as an attribute. It is possible that a category variable may be dependent on such an attribute. If an Employee `Works For : A Department` then Department could be a category attribute. However any single valued attribute of Department could be used as a categorising variable. If Departments have Location as an attribute, then a report on Employees by Location of Department would be valid.

```
Class.Employee
    {
     Table.Et1;
      {
        Table_Attribute.Location
                  (Type : Category;
                   Path : Department>Location;)
       ......
      }
    }
```

In the definition of a category attribute for the COST model it is worth asking what sort of variables could validly be used as categorisers.

In practice any mapping from an instance of the collection being summarised to a single value from a restricted domain would be satisfactory (as long as it has some semantic justification). In many cases where the domain is a collection of objects, the collection has attributes which it may be valid to use as category attributes. Thus any attribute can be used as a category attribute for a table summarising members of a collection given that:

1. it is reachable from a member of the collection,
2. it has a single value,
3. it is either a list, collection, element of an aggregation or exceptionally a primitive value.

Thus an element of a table definition might include a category variable which is the County in which the Manager of the Department for which an Employee works, is located.

```
Class.Employee
        {
          Table.et1;
          {
            Table_Attribute.man_region
                  (Type : Category;
                   Path : Department>Manager>County;)
          }
        }
```

In conclusion, the nature of the path attribute as it defines a table attribute is becoming clearer. A category attribute may be derived by almost any path which at each step results in a single value for each member of the collection being summarised. The issue of multivalued paths will be discussed later in this chapter.

11.7.1 Aggregation

The CORD model supports the aggregation abstraction in two ways. Both are represented within the model by the *Part Of* relation. They are the aggregation of instances of classes through the *Part Of* link and the aggregation of domains through the aggregate domain_type. At this point in the discussion it is the domain version that is of interest; the part case will be discussed in the next chapter where structural relationships are explored.

An aggregate domain maps an instance of a class to a record structure consisting of two or more elements each with their own domain. By its nature an aggregation tends to produce a very large domain, frequently with no natural ordering or naming process. It is sufficiently unlikely to produce a useful category variable that it is reasonable to exclude variables with aggregations as their domains from being used as category variables. However any component of an aggregation which has itself a suitable domain could be a category variable. Thus given a typical definition of address:

```
Class.Person
    {
      Attribute.Name (String;)
      Attribute.Age (Integer;)
      Attribute.Sex (Gender;)
```

```
        Attribute.Status (Marital_Status;)
        Attribute.County (County;)
        Attribute.Address (Address;)
      }
Domain.Address (Type : Aggregation;)
      {
        Part.County (County;)
        Part.Postcode (String;)
        Part.Street (String;)
        Part.Town (String;)
      }
```

It is possible to define a category attribute called Location.

```
Class Person
      {
        Table.pt1;
        {
          Table_Attribute.Location (Type : Category;
                                    Path : Address>County;)
        }
      }
```

The category attribute Location is defined through the aggregation Address using the same notation ">" used for object paths and functions. This overloading of the symbol is feasible because either Address has an aggregate domain or an object domain so no confusion arises. The benefit for the user is that they do not need to know how Address is implemented. If as in some object relational systems Address would be a separate table with attributes Street, Town, County etc. and the Address attribute of Person would be a posted Identifier then the same path would identify County. There is a possible conflict with the use of function, which means that function names and attribute names must be mutually exclusive.

11.8. CLASSIFICATION HIERARCHIES

Whereas aggregate domains increase the cardinality of the category, attribute classifications reduce it.

The ability to regroup instances according to classification is a feature represented by many summary table models. In the context of statistical reporting, classifications and classification hierarchies are formal techniques which are well understood. In the context of the CORD model and applications of a MIS type,

the use of classifications becomes somewhat richer. Here it is necessary to discuss the nature of classifications in information systems, and to consider how classification hierarchies are to be included in the COST model.

11.8.1 Formal Classifications

By formal classifications are meant classifications elaborated for a purpose. Such classifications as Standard Regions, Standard Industrial Classification, Registrar General's Causes of Deaths are well established and have a simple many to one hierarchical structure. Where authors have addressed the issue, they have treated them as ISA-hierarchies and part-of hierarchies within the Domain of Discourse. However in the CORD model these are defined separately and rather than being objects are a separate type of relationship between domains.

```
Domain.Region (type : List;)
              [South_East; Midlands; North; South_West;]
       County (type :List;)
              [Essex; Warwickshire; Huntingdon;
               Lincoln; Staffordshire; Sussex; Northumberland;
               Cumbria; Devon; Cornwall; Lancashire;
               Shropshire; Oxfordshire;]
               {classification.Region
                   {Attribute.County (County;)
                    Attribute.Region (Region;)}
                   [Essex (Essex; South_East;);
                            ....]}
```

In this fragment of model, two domains County and Region are mapped to each other by a classification called Region which is owned by County.

11.8.2 Regrouping of ranges

A common special case of classification is the regrouping of a numerical variable into an ordered set of categories. A typical example is the recoding of age data into age groups.

```
Domain.age_group (type : list;)
                  [10-20; 21-30;]
        Age (Type : Subset ; Domain : Integer;)
            (constraint : "age > 16", "age < 65";)
            {classification.age_group
```

```
{attribute.lower (integer;);
 attribute.upper (integer;);
 attribute.age_group (age_group;);}
    [10-20 (10;20; 10-20;);
     21-30 (21;30; 21-30;);
     .....
     ] }]
```

This fragment of a schema defines two domains Age and Age_group. Age is a subset of the primitive Integer and is restricted to be within the range 16 to 65. Age-group is a classification or grouping of the domain of Age. Consideration was given to defining a special classification for regroupings as it would then have been simpler for the eventual processor to look for the class attributes. However, as this could have led to special cases for regrouping of dates or members from any ordered list, this alternative was not followed. This leaves the processor the more general task of interpreting classifications. Given that the classification has to be owned by the "many" domain and must map to another domain, it could be assumed with confidence that this interpretation would be reasonably straightforward.

11.8.3 CORD derived Classifications

Apart from formal classifications, many classification hierarchies arise naturally from the specification of the CORD Model and lie within the DoD.

Thus for a hierarchical organisation: Organisation consists of many Divisions which consist of many Departments which consist of many Sections. Then there exists a natural classification or aggregation hierarchy. Such a hierarchy will be characterised by pairs of classes, one of which has an attribute with the other class as its domain, or which have a part_of link.

```
Division
         {Part.Department;}
or
Class. Department
        {
         Attribute. Address (String;)
         Attribute. Division (Division;)
        }
```

If the COST model is to manage the natural semantics of business data it will have to be able to create classifications from explicit hierarchies and from implicit object cross-references. In implementation terms, specifying a path through a classification is no different from specifying category attributes which are derived from object links. It is the table designer who has to have the appropriate domain knowledge to specify such paths.

11.9. SET VALUED ATTRIBUTES

Like other object models and unlike the relational model, CORD permits attributes to take more than one value. In the previous discussion it has been assumed that the mapping from an attribute to a domain has been one to one or many to one but not one to many. It is necessary to consider the issue of both category variables and summary variables in the presence of set valued attributes.

11.9.1 Summary variables

Where the attribute is to be used in producing a summary attribute, it will always need to be aggregated to produce a single value. This will be as true if there is only one value for the attribute per member of the collection as if there are more than one value. It is thus possible to confound summaries in a natural way so long as the statistical logic is correct.

For example, a salesman may make many calls in a reporting month, and a summary may be required which aggregates salesmen giving as attributes the number of calls made, the total mileage travelled etc.

```
Salesman {
          Attribute.Name{}
          Attribute.Sales_Calls
            {domain : Sales_calls; MaxCardinality : 20;}
        }
domain.Sales_calls (type : Aggregation;)
                  {
                    Attribute.Mileage;
                    Attribute.Client;
                  }
```

A path like:

```
sales_calls>mileage>sum ()
```

Would yield a suitable summary statistic, giving the total mileage travelled for a collection of salesmen. However there remains a problem, if the number of calls made is considered.

```
sales_calls>count
```

Is it intended to count the number of salesmen or the number of calls? This problem resurfaces when the average mileage is asked for. Is this the average per call, or the average per salesman? Within the COST model this problem is referred to as fan out. Whenever a summary is to be applied to a set of values which is subject to fan out, then the scope over which the summary is to be calculated needs to be specified. This could be presented as an argument to the summary function.

```
Sales_calls>mileage>average (salesman)
Sales_calls>mileage>average (sales_calls)
```

These two paths can clearly produce very different results.

11.9.2 Category Variables

It is not evident how to treat multivalued category variables. They could be seen as aggregates with the Cartesian product as the domain, but frequently there is no fixed cardinality. If the number returned is generally small then one could create a category based on the count:

```
employee>children>Count(Children)
```

and any single valued function of the set returned could make a category variable. Thus the syntax applicable to defining summary variables could be used to define summaries to be used to generate categories. In this context the most popular summaries SUM and AVERAGE are unlikely to be useful but COUNT, MIN, MAX and COUNTSELECT are all potentially useful. Thus the path (not

implemented in the Object Prototyper):

```
employee>children (age>max (children))>sex
```

would give as a category the gender of the oldest child of each employee.

11.10. CONCLUSIONS

A partial model for COST defining the table schema is :

```
Model.Cost
  {
    Class.Table;
    {
      Part.Table_Attribute;
      {
        Attribute.Type (Table_Attribute_Type;)
        Attribute.Path (Path;)
      }
    }
    Domain.Table_Attribute_Type;
    [
      Category;
      Summary;
    ]
  }
```

An instantiation of a table for the employee class results in the following partial schema:

```
Class .Employee (IsRoleOf : Person;)
    {
      Attribute.Department (Department;)
      Attribute.Children (Employee;)
    }
    {
      Table.et1;
      {
        Table_Attribute.Region
                (Type : Category;
                 Path : Address>County>Region;)
        Table_Attribute.Ave_Age
                (Type : Summary;
                 Path : Age>Average;)
        Table_Attribute.Status
                (Type : Category; Path : Status;)
        Table_Attribute.Sex
                (Type : Category; Path : Sex;)
      }
    }
```

The example shows three category attributes and one summary attribute. The category attributes Status and Sex have simple paths. "Status" and "Sex" are each inherited from the parent role Person. The category attribute Region on the other hand has a more extended path through the aggregate domain Address, to the part County which as a classification has the domain Region. The summary attribute Ave_Age is derived by applying the summary function Average with the scope Employee. As there has been no fan out in this case the scope is the only possible application of the function.

The CORD data model has been shown to be adequate to describe statistical summary tables without specialised definitions or functions being necessary. This is a significant step forward, because although previous authors have found it possible to provide a physical representation for statistical summary tables using the relational model, no convergence of modelling schemas has been attempted. Although not apparent at the outset, the simplicity of the COST schema is a consequence of the semantic richness of the CORD model.

Chapter 12

Structural summaries

12.1. INTRODUCTION

In the previous chapter the COST model was introduced and the issues resulting from summarising attribute information discussed at length. However this is not all the relevant information about a collection of objects that could reasonably appear in statistical summaries. As COST is based on CORD, an object model rather than a relational model or Entity-Relationship model, it is necessary to explore the other information that is available about a collection of objects. This goes beyond simple attribute information and concerns in part information derived from the structure of an object, its relationship in the type hierarchy etc. This is called structural information.

The treatment in the previous chapter dealt with the summarisation of object attributes. However, there is a variety of ways in which information other than the immediate attributes of an object may be the means of categorisation or the subject of summarisation. Logically these extensions of the attribute base are very different, but they are collected together and discussed here.

A path structure has already been used to enable a category or summary attribute to extend a domain, but is this sufficient to deal with all hierarchies?

The inheritance hierarchy is one source which is an extension in the class domain; a PartOf hierarchy is an extension in the instance domain, a role hierarchy is an extension in the time domain and the use of attributes of attributes (explored in the last chapter) is an extension in the domain domain.

12.2. THE INHERITANCE HIERARCHY

12.2.1 Heterogeneous collections

The inheritance hierarchy is a major distinguishing feature of the object model and it is clear that the instances of any collection of objects can be described by their

pedigree. However Collections in COST will be homogeneous i.e. they contain only members of the same class. This may seem an arbitrary restriction, and an argument for summaries constructed to compare elements from heterogeneous collections could be made. Imagine that witnesses to a crime (or perhaps a very bad video recording) report that something left the scene in the dark at speed, but they were not able to categorically say whether it was a person or a motor vehicle. Thus a collection of suspect people and motor vehicles could be made and described by the many attributes which they share by having the same or closely related domains - height, weight, speed are possible examples. It would need a tortuous justification for summarising the members of the collection in a table which did not have the class distinction as a major dimension. Where the members of the collection do not have a common ancestral class which itself could form sensible collections, in the rare circumstances that such a comparison is necessary separate COST models could be defined, one for each class, and these could be combined into a common presentation by concatenating the views (see Chapter 13). The inability to produce a cross classification of a mixture of people and cars by height, weight and speed is seen as a sensible restriction.

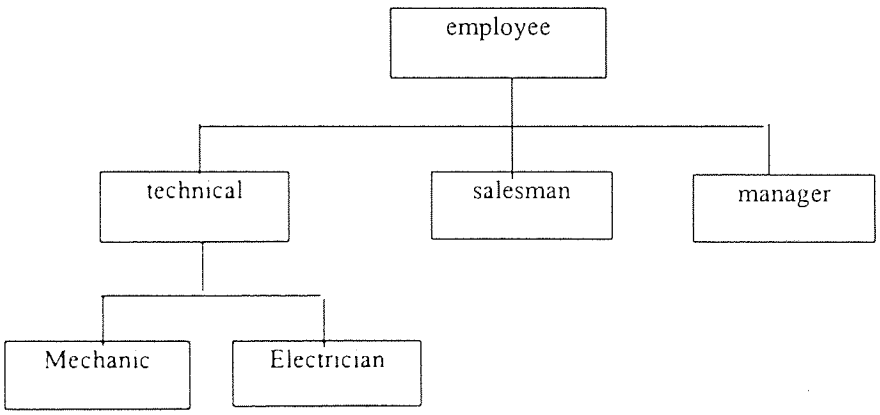

Figure 12. 1 : An inheritance hierarchy

12.2.2 Homogeneous collections

Following on from the previous discussion, it is only collections of objects which have a reasonably identifiable common ancestor that will be discussed further. Thus the collection of objects being summarised will lie in an inheritance

hierarchy being itself a specialisation of a superclass and possibly have instances which are themselves specialisations. A typical example of such a situation is the employment hierarchy is shown in Figure 12.1.

12.3. CATEGORISATION THROUGH SUBTYPES AND SUPERTYPES

Assuming that the hierarchy is static and does not include roles, then any instance in a collection of objects can be categorised by the lower levels of the hierarchy. Thus employee could be classified by role in the organisation (technical, salesman, manager) and technical staff could be classified by specialism. Both cases correspond to the situation of mutually exclusive subtyping (technical | salesman | manager) and (mechanic | electrician). There is a logical reference problem which arises, which in many implementations of an object model would not appear as a problem. If being a subtype is a property of the subclass itself, then the parent class need not be aware of the existence of its children. In this case, when creating a collection, the retrieval algorithm has to recursively search for all objects which are subtypes of the parent class. Alternatively at an implementation level this may be dealt with in a number of ways, one of which may be by prohibiting the instantiation of staff so that only instances of engineering or clerical or management staff can be created. Another might be by giving staff an attribute staff_category which can only take values "technical", "clerical", "management" and linking the creation of partial instantiation of the attributes relevant to the specific staff_category as an integrity constraint issue. As both COST and the CORD model on which it is based are intended to be conceptual models, it is inappropriate to assume any particular implementation for mutually exclusive subtypes. Instead a function is assumed which will construct a categorical variable from the object class membership information.

```
ObjectId>Exists(class1,class2,...)
```

This fragment will return the *name* of the class to which objectid belongs if the class is in the argument list. Otherwise it returns null. If the object is an instance

of more than one of the classes in the class list the result is unpredictable. By introducing the Exists() function it is possible to assume that a collection of objects could have subtype as a category variable so it would be possible to produce a table of technical staff broken down into Mechanic and Electrician and display summaries of any attribute of Employee for each sub-collection.

12.3.1 Attributes of subtypes

As has been discussed, the inheritance hierarchy can produce valid categorical variables for COST. It is useful here to consider to what extent attributes of subtypes may have a role in categorisation or summarisation.

By definition, an attribute of a subtype will not have defined values for all instances and could not sensibly be used to categorise the whole collection without the added value "Not Applicable". Further, the scope of naming means that two variables with the same names but associated with different subtypes need not have the same meaning nor the same domain. There is no obvious way in which such a variable can be considered to describe the collection as a whole. Thus it is reasonable to exclude the use of subtype specific attributes in COST.

12.3.2 Summary variables through subtypes

Although such subtyping would not usually provide a summary variable, by the process of section 11.6 a summary could be derived from it just as for any nominal variable. Thus a collection of technical staff could have as a summary variable the "percentage of Mechanics".

12.3.3 Table attributes through super-types

The higher levels cannot be used for categorisation, or for summary variables as by definition all members of the collection will belong to the same super class. Of course higher levels will provide variables by inheritance which could be used for categorisation at the lower level, and this has been included in the previous

chapter.

12.3.4 Summary

In inheritance hierarchies category and summary variables can be generated for the inherited variables in the usual way. No category or summary variables are derivable from the supertype and can only be generated from subtypes by the use of the Exists() function to create category information from the subtype, or by simply treating the subtype category as a list domain and applying a Count() function.

```
Table.engt1;
        {
          Table_Attribute.Ave._Age()
               (Type : Summary;
                Path : Age>Average();)
          Table_Attribute.TypeofEngineer
              (Type : Category;
               Path : Exists(Electrician, Mechanic);)
        }
```

The fragment of model above illustrates some of the issues discussed in this section.

Table 12.1 is presented to summarise the results of the above discussion on the use of sub/super type structures in the creation of summary variables.

Inheritance	Category	Summary
subtype - supertype	no	no
supertype - subtype	through Exists()	Count() applied to subclass

Table 12. 1 : Summary and Category variables and Inheritance

12.4. PARTOF HIERARCHY

The instances in the collection being summarised may well be members of a PartOf hierarchy. As has been discussed earlier, the ability to model complex objects is a feature of the real world which is found in most object models. This complexity is modelled in two ways: by the aggregation structure and by the

PartOf relationship. PartOf is very different from the inheritance hierarchy, because it represents a link between different types of objects which cannot inherit attributes from each other. It represents links at an instance level rather than a definition at a class level.

Thus in Figure 12.2 a book may be part of a series and may itself be made up of text and pictures. In a true PartOf hierarchy (rather than just a connection) the concept of cascade delete applies. If a book was to be destroyed - e.g. publication was to be abandoned, then the text and pictures that compose it would also be lost. This model is different from that say of a legal contract from the point of view of a lawyers office (Figure 12.3).

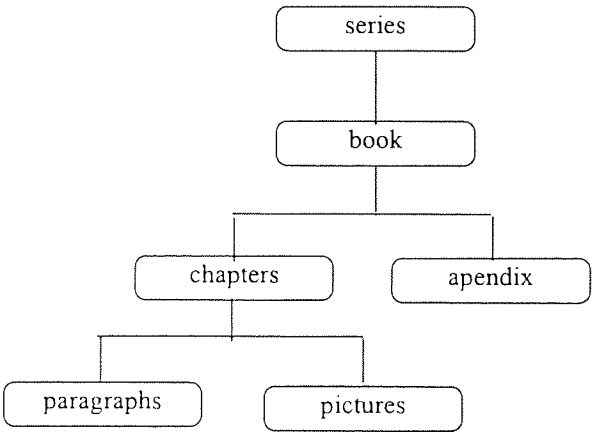

Figure 12. 2 : PartOf hierarchy

Figure 12. 3 : A relationship

In this case a contract is composed of a selection of clauses, and if the contract negotiation falls through and it is abandoned, the stock of clauses remains.

If a collection of books are being summarised then both summary variables and categorisers may be derived in the aggregate and dis-aggregate directions.

12.4.1 Aggregate direction

It is the nature of PartOf hierarchies that, unlike inheritance hierarchies, a collection of books would not all be members of the same series; thus attributes of the 'parent' will be interesting and relevant attributes of the 'child'. Moreover, as the child does not automatically inherit parent attributes in a PartOf hierarchy, it would be necessary to specifically reference any parent attributes that were required to make up a summary table.

A book could be classified by the type of series it was a member of, and such a category variable included in a table specification. It would also be possible to derive a summary variable from some property of the series such as the average number of books in the series; however as this information would span the elements being summarised, it would need to be interpreted with caution.

12.4.2 Disaggregate direction

A book could also be described by reference to its component parts. However, because the existence of parts is common to all parents but there are typically a variable number of parts, most useful summary information will be summary statistics, quantitatively describing the parts. Thus the number of chapters, the average length of a chapter, the number of pictures, would be typical summaries derived in the disaggregate direction.

```
Type : Summary;
Path : chapter>paragraph>text>length>Average;
```

Category information will typically be derived from the presence or absence of certain parts and will be derived by applying some transformation from summary to category variables. A book may be categorised by its length which in turn may be derived by counting the words in its parts.

12.4.3 Summary of PartOf table attributes

Where objects being summarised are elements in an aggregation then a number of

summary attributes can be generated. Attributes of the 'parent' can be used to create summary or category attributes. Sums or averages of attributes of parts of the aggregation can also be used.

```
Table.bt1 (Owned_By : Book;);
{
  Table_Attribute.Series_Type
     (Type : Category; Path : series>type)
  Table_Attribute.Page_count
     (Type : Summary; Path : chapter>pagecount>Sum())
  Table_Attribute.chapter_length
     (Type  : Category;
      Path : chapter>paragraph>text>Length>Length_group())
}
```

The fragment of model above, which summarises information about books, illustrates some of these possibilities. A collection of books can be categorised by the type of the series they are part of. They can also use the sum of the number of pages in their component chapters as a summary measure. Even non numerical attributes of component parts can be used as summaries, an example being the average length of the text making up the paragraphs composing the chapters of the book. In this case Length is assumed to be a function which returns the length of a text attribute.

Direction	Category	Summary
part-aggregate	Yes	Yes with caution
aggregate-part	Yes (conversion of summary)	Yes

Table 12. 2 : Summary and Category attributes from aggregations

12.5. ROLES

The object model derived in this thesis not only supports the traditional features of inheritance and aggregation, but, in order to deal more appropriately with the dynamics of the problem domain, roles have been introduced.

Consider an instance of a collection of union members. It can clearly be categorised by attributes of union member (e.g. membership category, union) and

may be described by suitable summary attributes (e.g. length of membership, subscription). In the upward direction, as Role_Of is a link at instances level, then it may be appropriate to describe a union member by attributes of their super-role: for example employee (salary, job title).

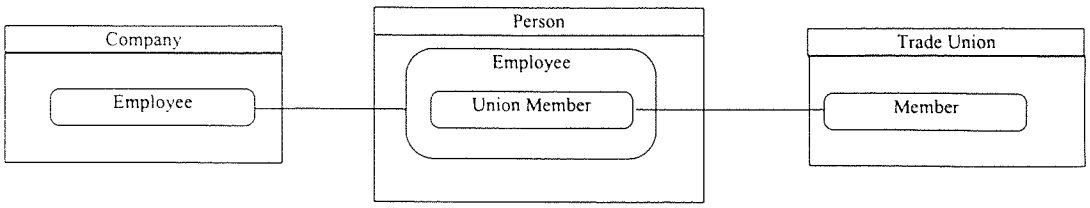

Figure 12. 4 :A role model

In general it will be possible to derive summary attributes on the super-role. In the sub-role direction it becomes difficult to derive either categorisers or summary variables, because of the one to many relationship between employee and union membership. This means that not all employees will be union members, and (depending on the semantics) an employee may be simultaneously a member of two unions. Where the semantics impose only an optional one to one relationship, then an obvious categoriser would be Union Member {true | false}. Where a 1 to many relationship is specified, then a summary variable would be Count of Union memberships.

It is apparent that Role_Of has properties similar to both inheritance and PartOf hierarchies, but has the distinguishing feature of having a time dependent existence. The inheritance path around a role will be considered first, and later the time domain issue.

12.5.1 Role parents

In this modelling it has been suggested that roles will generally have more than one parent. It is necessary to consider the impact of this on COST. A role instance can be described not only by its own attributes but also by those of its parents. Thus employees can be categorised by their gender (person parent attribute) and the industry in which they are employed (employer attribute). They

can have summary attributes also based on both person (average age) and employee (average size of company).

The reality of role modelling is made clear when one considers the basis of such summaries.

The average age of people in employment:

```
people>age>average
```

The average age of employees:

```
employee>person>age>average(employee)
```

The average size of employers:

```
company>size>average
```

The average size of companies for which people work:

```
company>employee>company>size>average(company)
```

These examples illustrate that the relation between roles and both 'parents' can lead to valid summary information.

12.5.2 Role children

Where roles are children of the object being summarised, they behave like PartOf and it is possible to count certain attributes.

12.5.3 Summary of Role table attributes

Three types of relationship involving Roles have been discussed. The relationship between a parent and a role and the relationship between a role and its two types of parent.

Direction	Category	Summary
parent - roles	Exists()	Count()
role - 'supertype'	Yes	Yes
role - 'aggregate'	Yes	Yes with caution

Table 12. 3 : Summary and Category attributes from roles

12.5.4 Other aspects of summaries based on Roles

12.5.4.1 Role status

An intrinsic attribute of any role is its status, and if this is not implemented by an explicit attribute, it is still available as a characteristic of a role, which could be used for creating a category attribute.

```
employee>status
```

12.5.4.2 Historical information

It is not the intention here to explore in any detail the nature of summarisation of historical databases. Assuming that all changes in a object are remembered, then in principle it is possible to ask for a report based on a previous point in time. An example would be a report on employment as at 1st Jan. 1979 compared with 1st Jan. 1989. In the absence of a true historical database for which the same queries can be issued, one at 1/1/1979 and the other at 1/1/1989, this problem is likely to be solved at the statistical table level, that is by attempting to retrieve a summary table produced on the 1/1/79 and comparing it with the same table produced on the 1/1/89. Again the issue of the storage and retrieval of statistical tables is outside the scope of this study, but it will be discussed in the context of the relationship between COST and presentation models in the next chapter.

A different aspect is *remembered information:* an aspect of database reporting which is clarified by the role concept. A CV typically includes a list of previous periods of employment. This could only be constructed with difficulty using

successive time slices as in a historical database. As each period of employment is a distinct object then it is a question of policy if non active roles are retained; if they are then the typical CV information : employer, start date, end date, final salary, are likely to be immediately available.

12.6. INFORMATION FROM SUBTYPES AND ROLES OF SUPER-TYPES

It is appropriate to explore the relevance of orthogonal subtypes or roles of a supertype to the construction of summary tables. Suppose that in the example in figure 12.1 any member of staff could be a safety officer, is it relevant to make information about safety officers available to a summary of technical staff? It will automatically be available to a summary of all staff and logically, being orthogonal, could qualify any member of technical staff. Just as an attribute can be composed to represent the class membership of subtypes, so an attribute (binary) could be composed to represent being a safety officer, and this qualifies all staff and is thus inheritable by technical staff. The caution exercised earlier applies here when considering extensions through the subtype say to "years as safety officer" and although it would be trivial to implement it is felt that the decision to model safety officer as a distinct subtype or role rather than an attribute of staff should be seen as a barrier to the chaining of attributes. If the object model allowed multiple inheritance, then the same discussion could be applied to Down and Up again chaining.

12.7. CONCLUSIONS

The final model for a COST-Table is very simple. An individual Table is owned by a class. A Table consists of a number of attributes. Each attribute is either a category or a summary variable and each is defined by a path expression which is applied to the collect of objects of the type for which the table has been defined.

The same principles which underlie the construction of paths for simple summaries based on attributes, also apply to summaries based on ISA and PartOf

paths. The applicability of such paths for summary and category table attributes has been summarised in the text of the chapter.

Chapter 13

The Presentation Model

13.1. INTRODUCTION

The COST model is a logical model describing a multidimensional statistical table. It does not, however, address issues concerning presentation of this data. Traditionally data such as this has been presented in the report format or the statistical table format, and in many applications it would now be displayed on a VDU screen. Higher dimensional display devices, Virtual Reality and multimedia may be expected to play a part in the display of management information in the future. The presentation model for COST is at least one level of abstraction away from physical devices, but here the concern is to map the data to multiple two dimensional displays, be they screens or paper. To maintain the abstraction, the units of such a display which are visible at once will be called *views*. A *view* will typically be a page or a screen of information.

13.2. MAPPING INTO TWO DIMENSIONS

The problems concerned with mapping multi-dimensional data sets into multiple two dimensional displays are well known but appear to have received very little serious discussion. Nevertheless it is useful to rehearse the issues and problems here.

13.3. DIMENSIONALITY OF A DATASET

A summary table defined by 5 categories is in essence a 5 dimensional matrix of data with the summary attributes in the cells. The generic term for such an abstract collection of data is a hypercube.

The following discussion will focus on the high dimensional data sets found in management analyses: 10 or 20 dimensions would not be uncommon. However for representational ease the diagram and following discussion refer to a 3 dimensional data set.

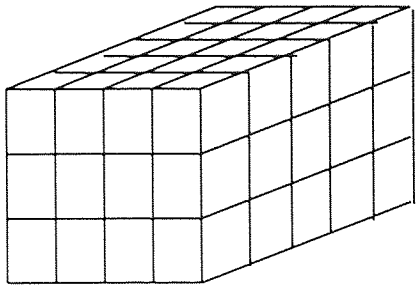

Figure 13.1 : A 3 dimensional dataset

A three dimensional dataset with 5, 4, 3 categories on its dimensions is shown in Figure 13.1. This can be presented in two dimensional views by creating 5 of 3*4 slices (or 3 of 4*5 slices, or 4 of 3*5 slices). Although this projects all the data into two dimensions there are significant losses. It becomes difficult to make comparisons between figures not presented in the same slice. In general, a projection will therefore choose to keep those variables which have the most interest in one slice, distributing the other variables over the slices. Within a slice comparisons are easy, with a rowwise or columnwise comparison displaying the variation due to a particular variable (holding all other variables constant).

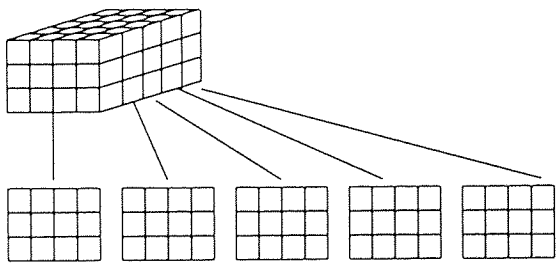

Figure 13.2 : Flattening a hypercube into views

It is not necessary to have only one two dimension matrix visible at once; a view may well display several slices through the data.

Where the size of the slices and the number of slices are small it is possible to represent all of a data matrix in a single view. Even in this case there are losses. It is easier to compare variations within slices than between them, and the reader will tend to assume that the difference between objects summarised within a slice is less than the difference between slices.

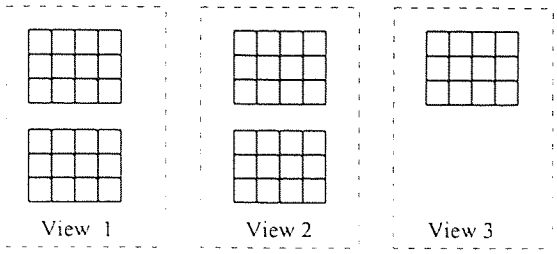

Figure 13.3 Collecting slices into views

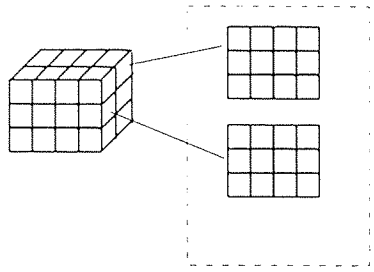

Figure 13.4 : All dimensions of a dataset in a view

It is also common practice to nest slices so that visual examination of the information is easier. This tends to reverse the previous position: comparison is now easier between categories in the third dimension.

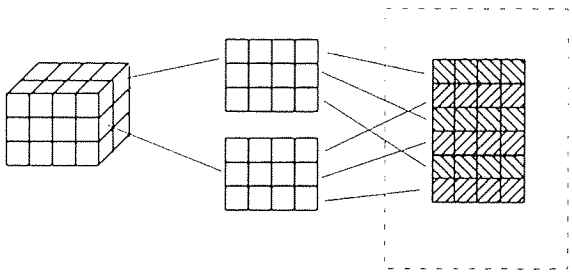

Figure 13.5 : Nesting dimensions in a view

13.4. NESTING OF DIMENSIONS IN TABLES

The main method of projecting data sets into lower dimensions is the nesting of dimensions. This is very common in the classical report format and the statistical table. Breaking down figures by gender within age group ("broken down by Sex and Age") has long been the mark of statistical tabulations. Such a nesting of dimensions preserves full information content of the source data matrix if there is a factorial approach such that every combination of categories is represented. Thus a 3*4*5 matrix can appear in two dimensions as 12*5 table, 3 * 20 table or a 15 * 4 table. A common diagrammatic representation of any axis is a tree

structure (Rafanelli and Shoshani 1990).

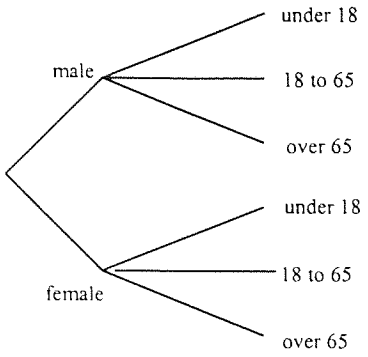

Figure 13.6 : A category tree

13.4.1 Ordering of categories

For the first time in the modelling ordering becomes important. Many categories will have a natural ordering, as does age in Figure 13.6. Others have no natural order but may have a conventional ordering such as causes of death, usually backed up by a numerical coding scheme. The general default is to use alphabetical order. It is important to use a conventional ordering wherever possible, and a standard ordering within any presentation, as this reduces the cognitive load on the reader.

13.4.2 Specification of presentation

A convenient way to specify a presentation would be to use such a category tree as proposed in Figure 13.6. A complete presentation would need three such trees. One would map to the row or vertical dimension of a view, one would map to the column or horizontal dimension of a view, and the third would define the ordering of views making up the complete representation of a data set. Where the ordering of categories is implicit or has been defined separately, it is more compact to simply specify the hierarchy of variables in a simple phrase. For example, it would normally be clear what is meant by a table showing "Age within Gender".

Thus the table in Figure 13.7 could be described as being "Age within Contract Type by Gender". If this is merely a subset representing one department of a

division then the full specification would be "Age within Contract Type by Gender by Department within Division".

		Gender	
Employment	Age	Male	Female
Continuing	18-30		
	30-50		
	50+		
Contract	18-30		
	30-50		
	50+		

Figure 13.7 : A view of a three dimensional slice of a dataset

13.5. PARTIAL REPRESENTATIONS

To fully present a data set demands a factorial approach. That is, the Cartesian product of the nested variables defines a single dimension. In many cases a partial presentation is adopted.

		Division	
Employment		Plastics	Rubber
Continuing	18-30		
	30-50		
	50+		
	Male		
	Female		
Contract	18-30		
	30-50		
	50+		
	Male		
	Female		

Figure 13.8 : A view of a partial four dimensional slice of a dataset.

In Figure 13.8 although four dimensions are represented, Age, Gender, Contract Type and Division, the full detail of the age breakdown by gender is not available, so this is a partial representation.

13.6. COMPLEX TABLES

A similar presentation to that in Figure 13.8 could have been arrived at by combining two factorial presentations from two different data sets, one based on Contract Type by Division by Age the other based on Contract Type by Gender by Division. Thus this could be a full representation of two three dimensional datasets. Combinations of presentations of distinct datasets which share at least one dimension are called Complex Tables (Lakhal et al 1989). These will not be included in the current discussion for reasons which are explored later in the chapter. This is clearly a special case of the assembly of different datasets which have no dimensions in common but share some other common theme or context, as is found in many market reports. Less obvious is the issue of distinct metadata. The implication of such a consolidation is that the data sets concern the same group of people at the same point in time. This may not be the case in assemblies but would be the case with partial presentations. At this stage partial presentations will be considered but not assemblies.

13.7. STAR+ NOTATION

Lakhal and Cicchetti (1990) have developed an intuitive notation based on the marginal labelling of a statistical table omitting the blank cells and using the top left hand corner to define the summaries which would appear in the cells of a completed table.

A Count		Gender
Employment	Age	account

Figure 13.9 : An example of the Star+ notation

The notation in Figure 13.9 defines a summary table with gender categories for columns, Employment * Age for rows, and the cells contain a count of the number (of employees?). Note that what the table describes is not clear and would need further metadata to define it.

The partial table of Figure 13.8 would in this notation appear as in Figure 13.10.

A : Count B : Count		Division
Employment	Age	a count
Employment	Gender	b count

Figure 13.10 : Star+ notation for figure 13.8

13.8. THE COST-VIEW

Following from the previous discussion it is clear that a presentation model is required to specify the external views derived from logical summary tables defined by the COST model. This presentation model is called the COST-View. This completes the distinction between the logical statistical table model and the physical presentation of the table which is part of the strength of the COST approach.

Like a COST-Table a View will exist as a schema and will subsequently be applied either to a collection or to an instantiated COST-Table to produce a physical table. The actual display characteristics of a table on paper or on screen: line spacing, margins, font style and size are not included in the specification of the View. This accords with current practice in the specification of documents (SGML) or WWW pages (HTML), where what is conveyed is structural information about headings etc. and the addition of format specifications associated with the externalised table is a subsequent stage.

- A COST-View is owned by a COST-Table. A View consists of a number of attributes which have been specified in the parent COST-Table.
- COST-Table attributes are either category attributes or summary attributes.
- Category attributes may be used to define Row, Column or Page COST View attributes.
- Summary attributes may be used to define Cell values.
- An attribute has a parent and an order within parent.

13.8.1 The CORD schema definition of the COST View

The COST-View is only slightly more complicated than the COST-Table. The

relevant parts of the COST schema addressing the COST-View model follow:

```
Model.Cost
  {
    Domain.View_Attribute_Type;
    [
      Cell;
      Column;
      Page;
      Row;
    ]
    Class.View;
    {
      Part.View_Attribute;
      {
        Attribute.Type (View_Attribute_Type;)
        Attribute.Parent (View_Attribute;)
        Attribute.Order (Integer;)
        Attribute.Source (Table_Attribute;)
      }
   }}
```

13.8.2 Instantiation of a View

Although this thesis is not in general concerned with execution and data storage issues, it is appropriate to consider how the execution of a COST-View might take place. The execution of a report must specify either an instantiated table or a collection:

```
Execute report
  if an instantiated View does not exist then
    if an instantiated Table does not exist then
      Instantiate a Table from the Collection
    endif
    Instantiate the View from the Table
  end if
End execute
```

The actual execution will be more complex if it is wished to confirm that any existing Table or View are as up-to date as the collection, or to search for an instantiated Table which would include all of the data required by the View as a subset.

13.8.3 Management of instantiated COST-Tables and Views

It is a pragmatic issue to reduce both the processing and in some cases the storage

overhead associated with the management of large instantiated COST-Views. In some areas, such as the monitoring of a few variables, the volume of data may not be great and is continually being updated. In others, such as the processing of an annual report, or the results of a market survey, the data is not changing, and a fixed time snapshot is not only convenient but is what is required. There is no proposal to remember instantiated views. Since many views could be based on the same set of data this would be very inefficient. Rather, as suggested in the earlier discussion, an instantiated table could be saved and used to increase the speed of instantiation of a number of different views. This is consistent with the provision of functions which can create new views from old, rather than manipulate instantiated views as is proposed by the majority of other writers in the area. To implement instantiation remembrance, attributes need to be added to the COST-Table definition. A file name and date of instantiation would be sufficient.

```
Model.Cost
{
 ...
    Class.Table;
    {
      Part.Table_Attribute;
      {
        Attribute.Type (Table_Attribute_Type;)
        Attribute.Path (Path;)
        Attribute.Location (String;)
        Attribute.Saved_Date (Date;)
      }
    }
 ...
}
EndOfModel
```

13.9. CREATING SUMMARY TABLES

In developing practical systems it is necessary to address issues of where the instance data (and the conceptual models) are stored. It is a premise of many authors that data will have been stored in a relational database, but for efficiency, confidentiality or other reasons data may be stored in a non normalised form. This may be described by a statistical summary table, but the storage of the statistical summary table has not in general been addressed. This way of holding

summary tables is commonly the situation for official statistical information, marketing information and much data sourced from outside the organisation.

Much of the literature on summary tables discusses operations that can be performed on the table (Lakhal et al 1989, Ozsoyoglu et al 1989). Most of this discussion demonstrates a failure to distinguish conceptual, logical or even implementation levels of the problem and is rooted in the assumption that data will be stored in a RDBMS. The literature focuses on two ways in which a statistical table can be realised. A presentation may be instantiated by raw data extracted from a RDBMS or by operations performed on a previously compiled summary dataset. The set of operations proposed by Lakhal et al in RTL is one of the more extensive sets of operations on summary tables. Although the underlying models are substantially different, the set of operations can be used as a means of validating the COST-Table, COST-View approach.

13.9.1 Instantiation and storage of summary tables

Lakhal et al (1989) suggest a number of functions for the creation and storage of summary tables.

Table and Relation operators create and save summary tables from/to relational tables. Within the COST approach the issue of storage is not addressed, and whether data is stored in relational or other structures is invisible to the user; however tables do need to be instantiated and perhaps saved.

- Instantiating a summary table (*Table*)

A summary table is instantiated by applying it to a collection. Thus if m_subsidiaries is a collection of employees from across a group of companies and if e_table is a COST-Table then:

```
m_subsidiaries>e_table
```

is the instantiation of the employee report for the group of employees in the collection. The ">" notation is used to represent this instantiation because it is

very similar to applying a function to the collection.

- Saving a summary table (*Relation*)

A temporarily instantiated COST model may be made permanent by giving it a name:

```
subsidiaries_table : m_subsidiaries>e_table
```

This is purely for the purpose of managing reports and for saving on execution time, or possibly as a way of preserving archive information. Once saved the data will begin to age, and appropriate metadata needs to be saved with it, in particular its currency i.e. what selection of employees it refers to and at what date it was created. This is not intended as a way of populating a database table from data stored in statistical summary tables.

Aggregate and *Relational algebra* are all concerned with the storage and retrieval of tables.

- *Relational algebra* operations are used to create the appropriate data set to be summarised.

In COST this is seen as confusing conceptual table level operations with instantiation. All the operations supported by relational algebra in RTL will be achieved either in the specification of the COST-Table (projection) or in the selection of the members of the collection (selection, union etc.). The use of a relational join to construct single flat tables based on many normalised tables is replaced by the use of path definitions for category or summary variables. In any ODBMS implementation the relational query expressions would be replaced by OQL expressions; in COST this is effected in the path expressions defining table attributes.

- *Aggregate* produces summary tables collapsed along a given category attribute.

Within the COST approach aggregate is implicit in the instantiation of a COST.

Thus when the COST-Table e_report is applied to the collection m_subsidiaries, the necessary aggregation operations are carried out. To produce a particular presentation the operation for the production manager a COST-View p_report is specified. This may be applied to the permanently stored COST-Table or to a temporary one:

```
m_subsidiaries>e_table>p_report

subsidiaries_table>p_report
```

Hence no particular functions will be necessary to implement these Lakhal operations.

- *Paste* combines two identical summary tables by summing the summary variables.

This operation can be achieved by defining a collection which contains the component collections and than applying the COST model and View to the new collection.

13.9.2 Modification of layout

Permute, *Displace*, *Rotate*, *Agcst* are all operations at the presentation level.

Within the COST model the definition of an alternative presentation which:

- reverses rows and columns (*permute*)
- changes the hierarchical order, age within gender becoming gender within age (*displace*)
- moves a category from one dimension to another (*rotate*)
- collapses a table by aggregating over a category attribute (*agcst*)

can all be achieved by specifying a different presentation model. This can be done using the graphical interface associated with COSTed (see next chapter). No specific functions will need to be implemented. However as a presentation may be rather complex, it could be appropriate to provide functions which can be applied to a presentation to produce a new (modified) presentation. This would be

most useful where an instantiated presentation already exists.

13.9.3 Complex table construction

Concatenate and *Extract* are concerned with the transformation of simple tables into complex and vice versa.

- composing two summary tables with a common dimension into a single complex table (*concatenate*).

	White	Black
Male	10	20
Female	5	6

	Employed	Unemployed
Male	5	25
Female	3	8

Figure 13.11 Two compatible tables

	Ethnic Group		Economic Status	
	White	Black	Employed	Unemployed
Male	10	20	5	25
Female	5	6	3	8

Figure 13.12 Concatenated table

The proposal by several authors of a concatenation type poses many problems which have been referred to earlier in this chapter. Whilst concatenation reflects the practice as far as composing complex tables from more elementary ones, there is no recognition of the role of metadata, nor the generation of appropriate metadata for the resultant table. In the example it may be assumed that the counts refer to the same population, but it is not necessary, and the definitions which were attached to the two separate tables cannot easily be associated with the one combined table.

Under the COST model two possible situations apply:

1. either the data being concatenated belongs to the same collection, in which

case a new COST-Table and COST-View should be defined and applied to the collection,

2. or the source data sets are unrelated, or not known to be related, in which case it is misinformation to construct composite tables. The user is always free to import instantiated COST-Views into any material, and co-locate them for ease of cross-reference, but this is seen as being outside of the COST environment.

    - Selecting a simple table from a complex table (*extract*)

No provision is necessary for this function, because as suggested above, the construction and maintenance of complex tables is seen as being outside of the responsibilities of summary statistical table management. This is in the realm of publication and editing.

13.10. CONCLUSION

Traditional discussion of statistical summary tables confounds several elements which have been argued in this chapter and the previous chapter to be logically distinct. There is the data, the logical table schema, the presentation schema and the physical presentation itself.

Reflecting current practice it was decided to excluded detailed formatting of the tables from the scope of COST, producing a structured instantiated statistical table, but leaving such issues as font styles and sizes to the publication process, be it on paper or on screen.

A presentation model for COST-Tables called the COST-View has been defined. The model is simply defined in terms of the CORD model and can be stored in the Object Prototyper. The standard operations proposed for statistical table management have been compared with the facilities implicit in the COST approach. It is possible to conclude that the COST-Table and COST-View combination meets all the requirements it was possible to identify for statistical

summary table presentation and manipulation.

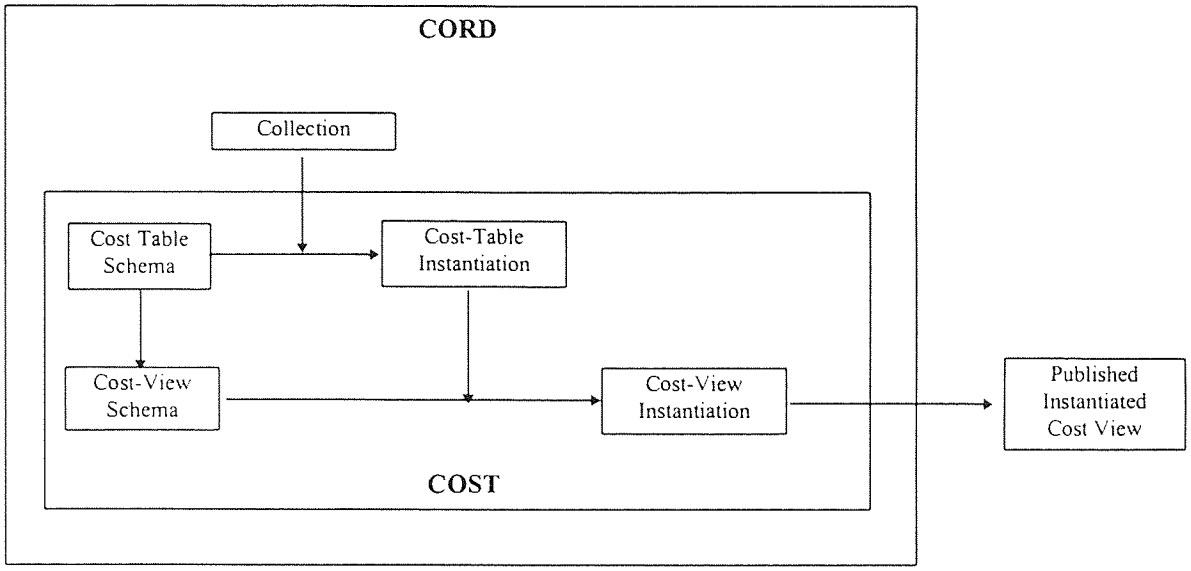

Figure 13.13 : COST-Table and COST-View in Context

Chapter 14

Implementing the COST Models

14.1. INTRODUCTION

In Chapters 11, 12 and 13, the requirements for a statistical table model were established. This involved the need to define paths for summary attributes and category attributes, together with appropriate aggregation functions. The definition of this model in the CORD schema has been introduced in two stages: firstly the COST-Table schema and then the COST-View schema. To validate the model the COST definition has been loaded into the Object Prototyper. A separate program called COSTed has also been developed to create and edit table and view definitions to validate the practicality of the model.

In this chapter, issues concerning the implementation of COST within the Object Prototyper, and the development of COSTed are addressed.

14.2. INTRODUCING CORD PATHS

The consolidated COST model is shown below. Most aspects have been discussed in some detail in the preceding chapters except for the nature of the definition of a path. In giving examples of paths in the previous chapters intuitive notation has been used. However, now that the nature of the requirements of the path element of a table definition have been explored, it is appropriate to lay the definition and interpretation of a path on a firmer basis.

```
------------------------------------------------------------
Model COST
------------------------------------------------------------
  Model.Cost
  {
    Domain.View_Attribute_Type;
    [ Cell; Column; Page; Row;]

    Domain.Table_Attribute_Type;
    [ Category; Summary; ]

              Class.View;
    {
```

```
        Part.View_Attribute;
        {
          Attribute.Type (View_Attribute_Type;)
          Attribute.Parent (View_Attribute;)
          Attribute.Order (Integer;)
          Attribute.Source (Table_Attribute;)
        }
      }
      Class.Table;
      {
        Part.Table_Attribute;
        {
          Attribute.Type (Table_Attribute_Type;)
          Attribute.Path (Path;)
        }
      }
-------------------------------------------------------------
```

A path expression is common to many programming languages and database interfaces. The path used in COST definitions is the same as should be available as a general purpose expression in the whole CORD environment. In the Object Prototyper a OQL window was installed which enabled queries in the form of path expressions to be entered, interpreted and executed. Within the COST model the main requirement is to interpret paths so that their suitability for defining Summary or Category attributes can be assessed. In Chapters 11 and 12 the requirements for defining summary and category attributes in a rich object environment were established. The following discussion considers in detail the semantics of a CORD path expression, and the problems of interpreting and eventually executing such expressions.

Most object models which may be alternatives to CORD have a very limited number of constructs.

Thus in the ODMG (1993) model:

```
employee_id.age
```

must refer to an attribute of an employee called age and:

```
employee_id.age()
```

must refer to a function called age. Both of these return values. In fact, some

object models may only allow access to attribute data via public methods or functions, so only the second interpretation is possible. In CORD however, there are many types of link between two objects, or between an object and a domain, so the simple notation will not be adequate. The basic step in a CORD path is similar to SQL, OMG and many programming languages:

```
object_id.An_attribute
```

This returns a collection corresponding to the values of the attribute of the initial object. A single value will be returned if the attribute has a cardinality of 1:1, or multiple or no values may be returned, but all are members of the domain of the attribute. If the values returned are primitive, then the evaluation of the path is terminated. If the values returned are Object Ids, then the path may continue without ambiguity.

Given the expression:

```
employee_id.address.town.branches.Count
```

it is reasonable to expect that it would return for the identified Employee the number of Branches in the Town in which they lived (an unlikely query, but no more so than most query examples in database textbooks!). On closer examination however, each of the four steps in the path is of a different type:

- Address is an attribute of Employee
- Town is an Object Id which is the value corresponding to part of the aggregate Address
- Branch is the Id of something which is part of Town
- Count is a function of the collection of Branches

However these are not the only steps in a path that can be encountered. A hierarchy path is a way of navigating the hierarchy of objects thus:

```
Class.Employee.John
```

is actually a reference to a single object (the one called John) within the class called Employee within the superclass Class.

Thus in the CORD model a pair of tokens:

```
a_token.b_token
```

may have more than one interpretation and the operator "." would be heavily overloaded. This overloading is a direct result of the richness of CORD.

In the CORD model the basic abstractions are instantiation and ownership. Both of these could be referred to using the implicit definers IsAnInstanceOf and Owner, but the use of symbols is more compact, and where a group of elements are of the same type or perform the same function, then gathering them together helps understanding. The CORD model schema notation uses {...} to indicate ownership, [...] to indicate instantiation and (...) to permit the assignment of values. It is clear that there is a need to develop a richer notation within path expressions to match the richness of the model.

14.2.1 Richness of the object model

The nature of a CORD path as used in the COST model or elsewhere will be discussed in the next sections.

14.2.1.1 Elements of the CORD model

The main elements of a model defined within the CORD implementation are

- Classes
- Domains
- Collections
- Roles
- Links

As a path starts with one or more objects (i.e. a collection) then it is worth asking what meaning could be attributed to the following pairs:

- Collection.Class
- Collection.Domain
- Collection.Collection
- Collection.Link

The only conceivable meaning that could be attributed to the first two is something like "do the members of this collection come from this Class or this Domain?". This is a possible but rather specialist relationship which will be dealt with later. Some combination of two collections would be governed by set operators creating the Union, Intersection, Difference etc. As usual, special symbols would be introduced for such operations and this has not been considered necessary in the context of the implementation of COST. Clearly the most interesting pair is the Collection-Link pair which would be expected to yield information on what the elements of the collection are linked to and it is this pair which will be explored more fully.

14.2.1.2 CORD links

A path must cope without ambiguity with the range of abstractions which have been identified in the CORD model. In particular, it must deal with the explicit links that have been defined:

- Action
- Attribute
- Definer
- Function
- Part
- Relationship
- Classification

It must also cope with the two major abstractions, Instantiation and Ownership, and the implicit relationships, Subtype, SubSet and Role, which link members of classes.

14.2.1.3 Identifying elements

Because names of objects are not unique across the entire instantiated model, a problem arises of identifying which particular object is being referred to at any one time. Traditional modelling solves this problem by restricting the number of types of elements in an expression, by specifying unique names in the appropriate scope, and by distinguishing between possible types of elements. So in the example already presented:

```
Employee.Age
Employee.Age()
```

are immediately taken to mean the attribute Age of Employee or the function Age() of Employee. However, without making unreasonable restrictions on the use of names, CORD presents a substantially more complex problem. An object typically has a unique name within its type and ownership group. This means that names can only be clearly resolved if both aspects are known.

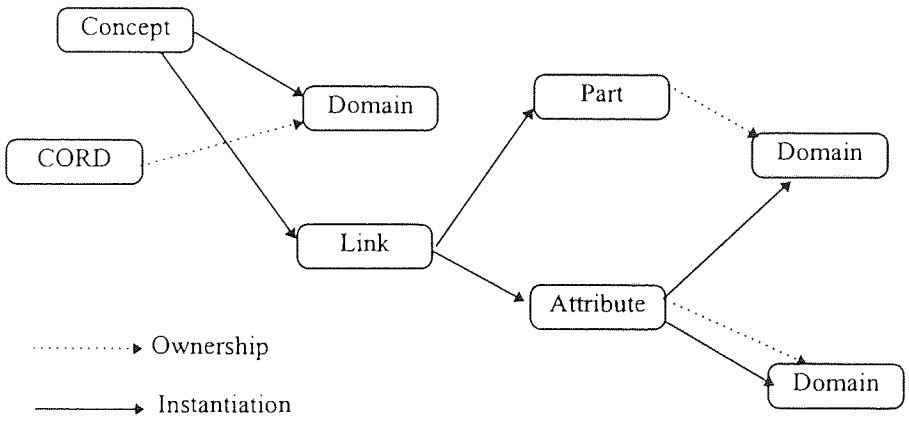

Figure 14. 1 : Different meanings of "Domain" in the CORD model

In Figure 14.1, three different interpretations of the item "Domain" are shown. Domain is a class owned by the CORD model, Domain is also an attribute owned by attribute, and a different attribute owned by Part. Just as in the CORD schema distinct symbols are used to group instances and to group owned items, so a path will need two symbols to trace the owner and type of any object named. The "." (dot) is reserved for defining instance relationships and the ">" (angle bracket) is used to describe ownership relationships. That this will not necessarily be

adequate to identify objects is suggested by the fact that the tree structure of object identity exemplified in Figure 14.1 will not easily be representable in a linear path expression. The CORD schema expression below would become the path shown:

```
Concept.Link.Part { Concept.Link.Attribute.Domain}
Concept.Link.Part > Concept.Link.Attribute.Domain
```

14.2.2 The instantiation link re-examined

The use of "." to identify instantiation paths needs further examination. There are two distinct forms of the specialisation relationship: that between classes (Subtype / Supertype) and that between an instance and a class.

14.2.2.1 Instantiation

Since an instantiation path is a reference to a single object, and it has been decided to reserve the dot notation for instantiation, the evaluation of such a path is straight forward.

```
a_class. a_name
```

becomes "find the member of a_class with the name a_name". This is a basic operation of any object modeller. There are three possible results: it returns an Object_Id, it fails to find a match, or it returns more than one Object_Id. The latter depends on the naming scope permitted. In CORD as in other models, it is not reasonable to insist that all attributes, for example, have globally distinct names. In practice, the default for CORD is that a name is unique within its class and owner. To overrule this, the definer *Scope* has been introduced which for any class can be set to "global". In the current implementation, only the names of instances of Concept and Definer have global scope. In general a path will need to have an owner defined for it and will typically be:

```
owner_id > a_class.an_instance_name
```

which is resolved in that the instance named is sought within the class and owned by the owner.

14.2.2.2 Supertype

Where the relationship is one of Supertype, it is not necessary to explicitly state an ISA path:

```
employee_id.Person>Name
```

If Name is an attribute of Person and Employee is a subtype of Person, then subject to naming conflicts, the path:

```
employee_id>Name
```

will be understood. Thus in the first path above, Person is redundant. Whether it should cause an error if presented to a path parser is another matter, which will not be discussed.

14.2.2.3 Subtype

The interpretation of a Supertype step in a path is straight forward. However there is a potential problem where the reference is to an attribute of a Subtype:

```
person_id > Employer_name
```

If the person is also an employee, then the path is interpretable, and the name of their Employer is returned. However if they are not an employee then the attribute Employer_name will be undefined. One possible way of dealing with this is to throw an exception, the other possibility is to return a NOT-APPLICABLE value. This is an extension of the treatment of NULLs. Rather than reserve a special value within the domain of a variable as being NULL or no value recorded, every attribute value has a possible qualifier, NULL or NOT-NULL. If it is NOT-NULL then the value of the attribute has some meaning, and if it is NULL the value is to be ignored. To cope with this further case, an extra state would need to be added to the indicator which could now take the values:

```
(NOT APPLICABLE | NULL | VALID).
```

Clearly a path will terminate with either a NULL or NOT APPLICABLE when these are encountered. These are not necessarily errors, as existence might be the

purpose of the query.

```
If exists(employee_id.child) then....
```

14.2.2.4 Roles

Where the link from an object is via a role, the path will behave rather like specialisation in the parent direction and Parts_Of in the child direction.

Thus:

```
employee_id>address
```

could mean the address of the person who is an employee or the address of the employer. This is a multiple inheritance naming conflict which can only be resolved by specifying the parent type.

```
employee_id>person>address
employee_id>employer>address
```

In the Parts_Of direction then:

```
person_id>employee.hours_ worked
```

would return a collection of integers, one for each employment in which the person was engaged.

14.2.2.5 Implementation of Null, and Not applicable

Although not essential to the validity of the CORD conceptual model, it is relevant to the general applicability of the Object Prototyper to consider how these indicators could be implemented at the Semantic Data Manager (SDM) level.

The basic element of information in the SDM is the triple

```
Object : Link : Value
```

Value represents a value from the domain of Link (as owned by the class of the object). The possibility that a Value is NULL could occur at this low level, so the addition of an indicator variable will have to be made at the triple level. A triple

thus contains four elements and becomes:

```
Object : Link : Value : Indicator
```

Where the domain of indicator is

```
( NOT APPLICABLE | NULL | VALID).
```

Note that NOT APPLICABLE will not need to be stored as a value in the physical storage layer, because it should be clear from the meta level definitions. In particular if the attribute A_Link is not defined for the Class of An_Object then there should be no triple of the sort

```
An_Object : A_Link : Value : Indicator
```

unless Indicator is NOT APPLICABLE.

Where A_Link is validly defined for the class of An_Object, then its value may be NULL or VALID. The Object Prototyper does not allow a null string to be written for the value part of a triple, so the absence of a particular triple can be interpreted as meaning that the value is NULL. But, by adding the indicator attribute to the triple, a NULL may be stored or not. Despite the technical inaccuracy triples will continue to be referred to as triples.

14.3. EXAMINATION OF CORD PATHS

Having explored some of the basic concepts, a detailed examination of CORD path expressions is possible. A CORD path returns a collection, and at any stage in its evaluation the evaluation to that point results in a collection, and the next stage is applied to each member of the collection so far, yielding in general another collection.

Thus

```
Employees > Address > Town > Population
```

If Employees is the name of a collection then the path Address>Town>Population would be evaluated for each member of the collection creating at each stage a

collection of Addresses, a collection of Towns and a collection of integers. Collections may be not only groups of class instances (employees, customers etc.) but also elements from any domain. The general step then in a path is:

```
a_collection.a_link
```

Because of the semantic richness of the CORD model, several different types of domain have been identified, and many different types of link. A possible means of analysis is to consider individually, firstly the different domains, and then the different types of link, to understand what steps could be validly specified and what would result from them.

14.3.1 Domain Types

Three main domain types have been identified and collections may consist of any of these:

```
Class
List
Primitive
```

A domain may also be an aggregation of any other domain types

```
Aggregation
```

14.3.1.1 Class instances

Where a collection is a collection of class instances, then the next step in the path can be any link defined for class members, as well as Supertype/Subtype and Subset links.

14.3.1.2 List / Primitive

Lists and primitives are terminal domains, and only functions defined on them could follow. See the discussion on functions later.

14.3.1.3 Aggregation

Where a variable has a complex domain, then it is not clear what would be an

appropriate next step.

There are three possibilities:

1. selection of one of the components for continuation,
2. extension over all components,
3. creation of a record composed of the elements from the aggregated domains.

It is difficult to identify a sensible meaning for option 2), because for another step to be appropriate it will have to apply to each component part, and this is only likely when they are from the same domain. Whilst 3) is an attractive terminal option, where a path continues it does not lead to intuitively sensible results. It was decided to implement 1) in CORD. Thus, where a path leads to a complex attribute, the next step will be the selection of one component of the attribute.

```
person_id>address>town
```

would be parsed successfully and return the town. The next step would depend on the domain of town.

14.3.2 Links

The CORD model defines a number of links which can be specified between objects:

- Action
- Attribute
- Property
- Function
- Part
- Relationship
- Classification

Not all links can be instantiated for each type of object. Every instantiated link must be owned by something and can only be understood in the context of its

ownership. The names of links can be overloaded. Each type of link in the current version of CORD will be discussed, to explore what role it might have in a CORD path expression.

14.3.2.1 Attributes / Properties

The most common link from one object to another is via an attribute. The intention of a path with an attribute is clear: the value of the attribute for the object should be returned. The next step following an attribute will depend on the type of domain it maps into. The user should not find themselves specifying paths which include properties explicitly. Properties will be implicit in paths involving instantiation, subtypes, roles etc. but should not appear in the path expression. Only in summaries which are reporting on the database itself would one find an explicit reference to properties.

14.3.2.2 Functions

One or more functions may be defined for any class of objects, and such functions may be applied to a collection of instances of the class, returning values. The CORD model includes some common functions which are available when constructing path expressions. In general functions are of two types: Summary or Aggregate functions which reduce the cardinality of the collection, and those which preserve the cardinality. Cardinality is most likely to be increased by multiply valued attributes rather than functions.

Typical summary functions are Maximum and Minimum, which return elements with the same domain as the original collection, and Count and Average, which return values in the number domain irrespective of the domain of the collection. A function has a domain which specifies the type of data returned, and subsequent steps in a path will depend on this domain.

Naturally functions can be concatenated:

```
person_id>name>length>maximum
```

would return the maximum (summary) length of the names of a collection of people.

For the COST model some summary functions are necessary to aggregate data, but as these have a wider applicability, they have been defined for many classes of object in the CORD schema.

Primitives

Integer data has traditional summary functions defined for it: Maximum, Minimum, Average, Sum and Count. All of these are aggregate functions, returning one value for an entire collection. Other non summary functions could be defined such as Double, which doubles the value of each integer in the collection.

String data: Count is defined for string data as it is for most data types; it returns a count of the number of items in the collection. Length is an example of a non summary string function, and returns a collection of integers representing the number of characters in each string.

Similar functions could be defined for *Boolean* and *Date.*

Lists / Classes / Collections

Unless they have been overloaded, basic functions exist such as Count for collections from all these domains.

Aggregations

No general functions have been defined for aggregations.

14.3.2.3 PartOf

When the next step in a path is a PartOf link the intention of the path is very clear:

```
book_id>series>editor
book_id>chapter>title
```

The first expression would return the editor of the series of which a book was part. The second should return the list of chapter titles in the book. These two PartOf links are in fact in different directions. Whilst the PartOf link is a standard element of CORD links its inverse Is_PartOf does not appear in the CORD schema. It can appear in path expressions to imply the inverse relationship.

14.3.2.4 Classification

Where the collection results in a classification, this is effectively a list with attributes and can be extended. If the town attribute of address had a classification domain with an aggregate level of region:

```
Person_id>address>town>region
```

would be parsed successfully and return a region.

14.4. FUNCTIONALITY NOT IMPLEMENTED

The focus of this work was to create paths which would permit the COST model to be demonstrated. This falls short of implementing a full query language. In particular not all the possible paths itemised in the previous section have been implemented or tested.

Some other issues which have also not been addressed in the development work are reviewed to show how they could fit into the framework of the path expression.

14.4.1 Assignment

It is intuitive to allow the standard meaning to

```
employee_id>name : "John"
```

That is, the employee referenced has their name set to John. This is only valid if the domain of employee>name is a string. Similarly:

```
employee_id>spouse : person_id
```

If the domain of spouse is that of person object instances, then this would set the spouse of the employee in question to be the person identified. Not only must the domain of the assignment be consistent, but also the cardinality. If the path resulted in a collection, then the assignment could only occur if it was matched by an ordered collection of the same size.

In evaluation of an assignment, both sides are evaluated and then the assignment takes place. If it is appropriate to consider the value returned by an assignment, then it is the set of assigned values.

Assignment has been implemented in the Object Prototyper in the form of:

```
Employee_id (name : John;)
Employee_id (John;)
```

The first case is clear. The second will assign the value "John" to the first ordered attribute of an employee. An attempt will be made to interpret the token "John" in accordance with the domain of that first attribute.

Assignment was not relevant to summary table paths but should be part of a general OQL.

14.4.2 Records

A record is an ordered set of elements.

In order to simplify repetitive definitions and yet achieve an intuitive schema, brackets are used to group attributes.

```
Employees>(name, address)
```

This returns the collection of pairs of literals: name and address of each employee. A NOT APPLICABLE will be returned for any element of the record which is not defined in its context, a NULL will be returned for a value which is simply missing.

14.4.3 Selection rules

In the context of this work, selection rules were not seen as being very important. On the whole statistical reports will be generated for a particular collection, and it is implied that the selection rules will have already been applied to the definition of the collection. However a simple extension of the current notation yields an intuitively acceptable selection specification.

```
Employee(name = "John")>Age
```

This will be treated as a mask and include all instances of employee for which the expression evaluates to true.

14.4.4 Multi level Statistical summary

As has already been discussed, a path typically returns a collection of values then:

```
count(path) or
path>count
```

could have the obvious meaning. They both would need to match the method count to the domain of path and if found return an integer. Similar constructions could be placed on:

```
path>average
path>max
   etc.
```

Where a path has several points at which it maps 1 : n then the application of count, average etc. could be interpreted in several ways.

```
employees>children>pets>count
```

could return a single count of all pets of employees' children, or the collection of the number of pets of the children of each employee, or the collection of the number of pets of all the children of each employee. An equivalent expression to the SQL Group By is required to resolve ambiguities:

```
employees>children>pets>count(group by children)
```

This would ensure that a collection results which maps to the children in the second collection elaborated by the path. This would require the return of a record:

```
aggregate.temp_collection
{
 attribute.child(person;)
 attribute.count(integer;)
}
```

14.5. COSTed

A program has been written to test the COST model. This was called COSTed, a COST model editor. This was developed in Visual Basic in a similar way to the Object Prototyper.

The program has two main functions, a COST-Table editor and a COST-View editor, each called from the main form.

The main form gives access to a menu which enables a prototype to be opened. All classes within the prototype are listed and can be selected for the creation of a summary. Tables can be created and edited for any class. Views can be created and edited for any Table.

14.5.1 The COST-Table editor

The main function of the COST-Table editor is to create category and summary attributes. The first version was implemented as a drag and drop interface. However this had to be abandoned when the complexity of path definitions became apparent. The current version displays a path and enables the user to select the next step in a path and hence to create and edit path expressions.

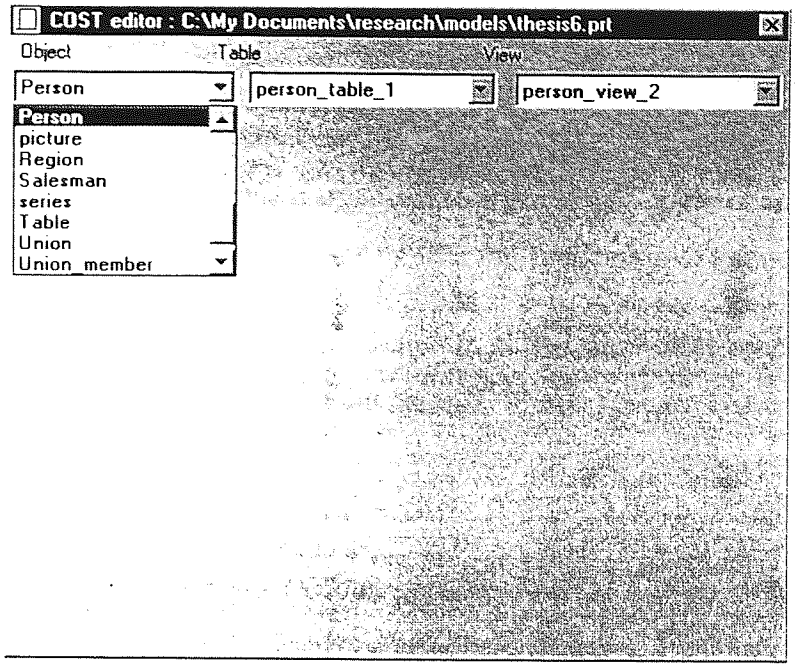

Figure 14. 2 : The COSTed main screen.

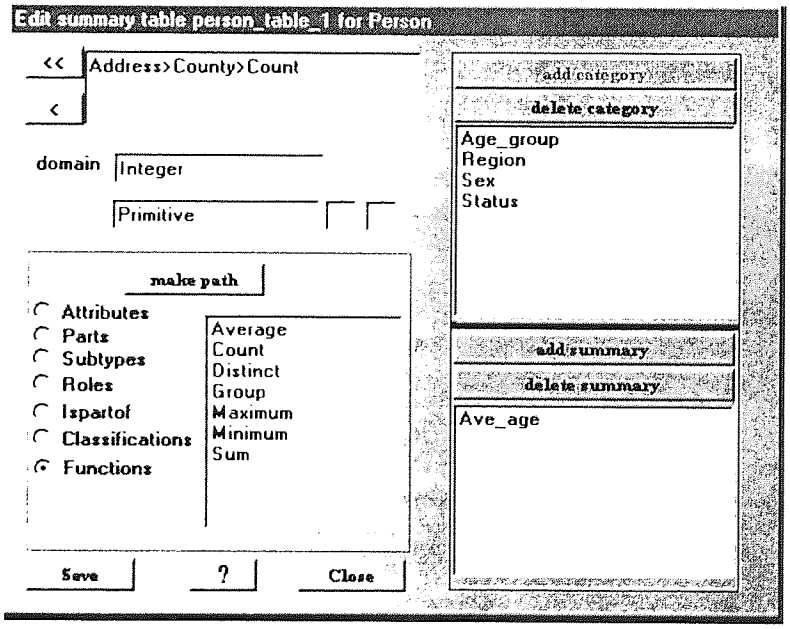

Figure 14. 3 : Table editor screen

At any stage the domain of the current path is displayed, and the various options available for a next step are shown. The user can view a variety of links: Attributes, Parts, Subtypes, Roles and IsPartOf.

For the current path the alternatives available are automatically displayed. For

example, if it has a List as a domain, then the **add category** is enabled and the **add summary** is disabled. On the other hand, if it has an integer domain both are disabled. The hint button (?) makes suggestions to the user. In the case of an integer it will point out that this is unsuitable for use as a category or summary attribute, but could be used as a summary after the application of a function.

A new attribute has a synthetic name created from the path expression, but this can be edited to give a more convenient name.

14.5.2 The COST-View editor

The presentation model has been implemented in the COST-View form. A view is created for any table and this can then be edited. The new view form displays the Categories and Summaries defined for the table and enables a view to be created by dragging and dropping them onto page, column, row or cell positions. This is not unlike current versions of the Microsoft Excel pivot table interface or the crosstabs element of Crystal reports.

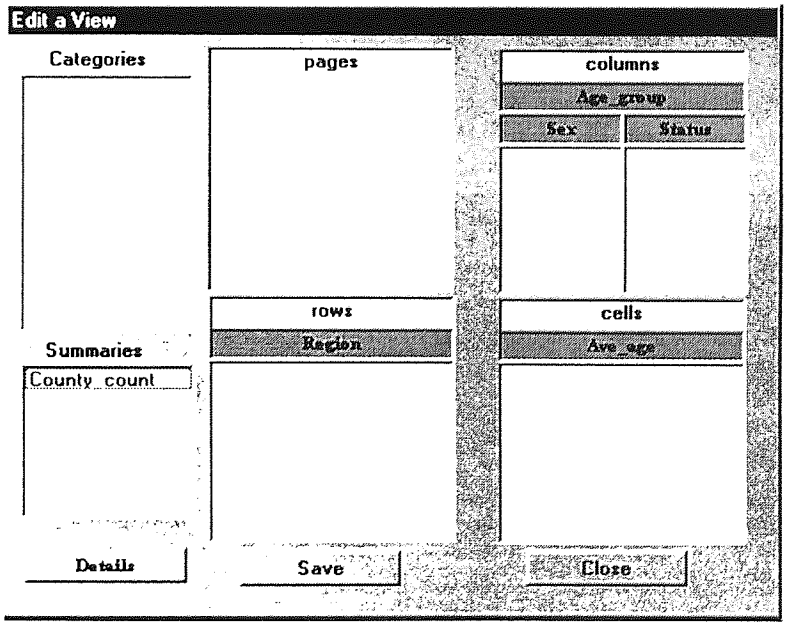

Figure 14. 4 : The View editor

14.6. CONCLUSION

The two elements of the COST summary table model, the COST-Table sub-model

and the COST-View sub-model, have been shown to be practical and complete by implementing the COSTed editor. The concept of a path has been explored, and shown to be practical. The models themselves have been defined in the CORD schema, and stored in the same SDM database as all the other meta data and any instance data.

The creation of a table editor posed problems in that the number of alternatives in the construction of a path is very large. This is a direct result of the richness of the CORD model. The COSTed interface constructed was simple enough to be usable by a domain expert wishing to construct a summary table, and the level of required computing knowledge is minimal and no understanding of the path schema is required.

The CORD model and its implementation in the Object Prototyper have proved to be adequate for the implementation of COST.

Chapter 15

Conclusions

15.1. INTRODUCTION

This research has been carried out over a period of years and it has always been a concern that the work of concurrent researchers could not be ignored. Where new contributions are an extension or continuation of earlier themes, then they have been included within the earlier chapters of this thesis. Where new contributions have the potential to raise completely new themes bearing on this work, it was considered appropriate to bring them together in the first part of this concluding chapter. The subsequent review of the contribution of the present work can then, if necessary, comment on the implications of these new directions.

15.2. THE NATURE OF INFORMATION AND INFORMATION SYSTEMS

After a long period in which information itself and the nature of information systems has been taken for granted, there have been some new attempts to clarify issues in this area.

In an evaluation of theories of information, Mingers (1996) considered 16 distinctive theories of information with reference to their applicability within the discipline of information systems. He observed that the majority were based to a greater or lesser extent on Shannon's (1949) work. He concludes that Dretske's (1981) theory of information flow and knowledge has the greatest potential for making a contribution to IS. Based on Mingers' explanation, the main components of Dretske's theory are:

- Information carried by a signal is objective and exists even if not observed
- Information must be true to be information. The only information a signal carries is that which is true about its origin.
- Information can be transmitted and stored.
- The information carried by a message is different from its meaning.
- The meaning of a message depends on the recipient.

There are some implications of this work for the understanding of statistical summaries. In particular, statistical summary tables contain information (if they are "true") which can be stored and transmitted. However, the meaning of a table is distinct from its information content. The statistical summary as an externalised physical object, a set of symbols on paper, has a certain surface information content and much nested information content. The table may record educational attainment of children by social class, location etc. Its information content is closely related to the original source information, a national survey perhaps, however the meaning has been moderated by the structure imposed on the information during the process of producing the table. A simpler example is a marks list in alphabetical order and the same list sorted into mark order. At one level they clearly contain the same information and are equally true but are designed to generate different meanings in the recipient. According to Dretske (quoting Mingers) "it is the receipt of information which generates meaning in the individual. Information is objective, the meaning it engenders is subjective.". This supports the dichotomy proposed in this work between the COST-Table and the COST-View.

The COST-Table is an attempt to produce an objective message. The message has two parts; a structure based upon the original data model which is as "true" as the original model is "true", and a content based upon the data with which it is instantiated. This instantiation itself has two components; the selection of instances - its collection- and their data values. For the message to be information it must be true, that implies that the selection rules must be explicit and that the data must itself be accurate measures of the attributes of those instances.

The COST-View, on the other hand, is an attempt to influence the meaning generated by the message in a typical recipient. According to Dretske's model a message contains a large amount of information which is selectively accessed by the perceiver. He describes this process as the "digitalisation of the analogue". It is this process which generates meaning for the receiver. It is the nature of most statistical summaries that there is a large constituency of recipients, and the view

is an attempt, at best to facilitate the extraction of meaning from the data, at worst to coerce the receiver into imputing a certain meaning into it. The quantity of information in a message is, as might be expected from a Shannon based theory, related to its surprise value. This is not seen as completely objective but is a function of the expectation of the receiver. It is thus not possible to generate objective views which are based on the statistical properties of the information, by displaying the most informative dimensions in the top two dimensional page, for example.

However Mingus/Dretske do suggest that a possible criticism of the model proposed is that no simple browser of COST-Views has been developed. This would free the viewer somewhat from the meaning imposed by the designer of the view and make it easier for the viewer to find their own meanings. The fact that all conceivable views can be generated from the COST-Table using COSTed does not necessarily give sufficient flexibility to offset the default message presented by the designer's view of the data. This is especially true for the less IT literate user.

15.3. THEORY OF TYPES

One of the problems with researching into a pragmatic subject like IS is that the real world cannot wait for theory and is developing at a considerable speed. Thus it is not always possible to be sure if the research work is descriptive or normative. In the area of conceptual modelling, a large pragmatic research base has been inherited. Past research appears to have two objectives; one, to rationalise the pragmatic with a veneer of theory, the other, to try to develop a sound theoretical basis which may move practice ahead faster and on a surer footing than the accumulation of pragmatics. This is particularly true in the area of conceptual modelling, including object modelling. In continuing the search for some underlying theory, the work of Zalta (1996a) of the Metaphysics Research Lab at Stanford University appears relevant. This is a monumental study, aiming to establish an axiomatic theory of abstract objects. The clear relevance to

conceptual modelling does not necessarily make it easy to learn the appropriate lessons from the work.

Based on the work of Ernst Mally (1899-1944) the Austrian philosopher, Zalta distinguishes between two classes of objects and two types of predication, one associated with each class.

First, there are real concrete objects which have determinate spatiotemporal locations, and physical characteristics. President Clinton and Socks the cat are examples. Their properties can be discussed in the normal way; 'Socks is a cat', 'Clinton is President', 'Clinton owns Socks'. This is called exemplification. The objects Socks, cat, President, and Clinton can exemplify particular relationships: thus Clinton and Socks together exemplify 'owns'.

Second, there are fictional objects such as Sherlock Holmes and Watson. Although it is possible to make the same types of statements 'Watson is a doctor', 'Holmes is a detective', 'Watson helps Holmes', these statements do not represent the exemplification predication, because neither Holmes nor Watson are concrete objects. The predicates in these examples are said to be encodings of a property. Thus Holmes encodes the property of 'being a detective'.

Zalta constructs a notation and derives various conclusions about abstract objects. He also goes on, in Principia Metaphysica (Zalta 1996b), to develop a theory of fiction. What can be said about fictional characters? Clearly there is more truth in 'Holmes is a detective' than in 'Holmes is a doctor' so it is possible to make consistent arguments within a fictional context.

The work of Zalta suggests a new foundation for the understanding of information systems and the modelling of objects in conceptual models. In particular, it would appear that an information system could be considered to be a work of fiction; almost all the classes it contains are created by the analyst and are not concrete objects in Mally's sense. The system evolves and changes, just as a novel does, and the possibility exists of asking questions about the objects represented in the

system, but these are abstract objects which encode properties, not the concrete objects which exemplify them.

During this research, there was a growing realisation of the significance of the distinction between the information system and the real world. It is this very distinction which would appear to be formalised by Zalta. This does not seem to impact on the conclusions of the research, nor the validity of the COST model, but does emphasise that the resultant instantiated tables should be true statements about the fictional story "the organisation's information system". This represents a different position to that implicit in many approaches to modelling, where the model and conclusions drawn from the instantiated model have the same relationship to the organisation that an applied mathematical model has to the real world. Having established an accurate model, any conclusions drawn logically from it will usually be true, but not necessarily so. This is the basis for the concern with database integrity. As a consequence there is a dangerous confusion between the real world and the world of the information system.

15.4. THIS RESEARCH REVIEWED

The introductory chapter to this thesis is "Information from Data" and the research has attempted throughout to understand and derive ways of improving the links between the two. However it has relied on particular interpretations of both information and data.

Data

Data in this context has been assumed to be the ordinary every day information processed by transaction processing systems in organisations. This circulating data which exists in all organisations is a very valuable resource. Increasingly this resource is stored and managed by standard relational database management systems.

Information

Information is data, organised and used for making decisions, and as such is the main objective of all management information systems. In the context of this research a rather narrow view of information has been adopted. It is assumed that such processed data will be in the form of statistical summary tables. That this is restrictive is obvious, but during the research no sources suggested other structures for information than the report or the statistical table. No doubt researchers in the area of Information Retrieval have wider definitions and a richer range of informative formats, but this was not pursued in this research.

Information from data

Merely rearranging sets of random numbers into matrices will not produce information, even if these matrices take on the appearance of a statistical table. The creation of information depends not on the numbers but on the knowledge about the numbers which explicitly or implicitly accompanies them. This information, which is referred to in the statistical community as metadata, is similar to the information contained in conceptual data models created by database analysts. The difference between them is the purpose for which they are constructed. Statistical metadata is deliberately constructed to give meaning to the data (Chapter 10). Conceptual models are created to give structure to the data (Chapters 3,4,5,6). Both create some linkage between the figures and the real world which the figures are supposed to describe.

This work thus has a major theme of integrating two techniques which have essentially the same information content, in the absence of any substantive integrating theory (Chapter 9). The choice of an object model (Chapter 7) as the common framework was inevitable. A rich modelling environment was required and simply tinkering with the relational data model or the entity relationship models were not seen as providing adequate modelling frameworks. The derivation of specific models for describing statistical tables has been a fruitful field for many researchers (Chapter 10). This was seen as attempting to describe

the artefacts of the IS, not the real world which the system modelled. A refined object model was considered to be the more natural bridge between information systems builders and information users.

The decision to build a flexible Object Prototyper (Chapter 8), rather than attempt implementing the models directly as an application, evolved from two separate concerns:

One was the need to be able to experiment with the specification of models, and to be able rapidly to change the whole structure as ideas crystallised. For example, once the need to specify a classification hierarchy was identified, the problem of how to fit it into the model arose. With the Object Prototyper it was possible to explore: treating it as a special type of domain, treating it as a separate object class and treating it as a link between domains. The Object Prototyper permitted all of these to be tried and the consequences explored.

The second reason for building a general purpose Object Prototyper was to ensure that as few assumptions as possible were built into the modelling environment and hence details such as the attributes of attributes had to be made explicit. (Chapters 7 and 8). The environment in which the Prototyper was developed (Visual Basic) is not a fully object oriented development environment and the final semantic data manager was implemented through a standard relational DBMS (MS Access). This posed no problem in accepting the validity of the concepts but did mean that the development was not constrained by existing object models built into the development tools.

The COST model was developed and specified in two stages. In the COST-Table model (Chapters 11) the basic structure was defined in terms of the CORD model. This was expanded with a fuller discussion of the complex paths which followed on from the richness of the CORD model (Chapter 12). The separate COST-View was defined permitting the presentation of slices of data from a COST-Table (Chapter 13). The two tools, one for manipulating COST-Tables and the other for manipulating COST-Views, were developed in the same environment as the

Object Prototyper (Chapter 14).

15.5. CONTRIBUTION OF THE RESEARCH

The original aim of this research was to develop and demonstrate new modelling tools, based substantially on the object oriented paradigm, which would better support the mapping of the detailed data which is generated by the activities of the organisation, to the models of the organisation which underpin the decision maker's approach to his/her tasks. (Refer to the Introduction, sections 1.7 & 1.8)

- The research has substantially extended the traditional data models based on the E-R model and relational database technology. The research has clarified many issues glossed over by current object modelling approaches, and developed a rich modelling schema. In particular, more coherent views of temporal objects and domains has been integrated into the general object model. The schema developed builds up from the basic abstractions of generalisation and aggregation giving both an equal role in the construction and definition of complex objects. This balances the obsession in much of the Object Orientated literature with inheritance as the main abstraction. The schema therefore highlights not only objects and attributes but also the contributions of roles, domains and collections.

- The CORD schema has been implemented using the Object Prototyper, a general purpose object management tool created specially for the purpose. The ability to define a schema which could be bootstrapped into a general purpose tool was a partial validation of the tool and the schema.

- The statistical summary table, while not appearing high on the MIS developer's agenda, is the logical structure underlying the management report and all aggregate data used in Decision Support Systems. The issue of the definition and management of summary tables has been addressed, and, although the work does not cast any significant light on summary tables per se, it has produced a two part schema for their representation. The two part schema,

COST-Table and COST-View, clarifies the nature and management of summary tables, which in the current literature is obscured by a confusion of implementation and presentation issues. The relationship between summary tables and the CORD object model turns out to be very simple, mainly because the richness of the CORD model made it trivial to include two new complex datatypes.

- The ideas embodied in the COST models were validated by the construction of a two part tool for the creation and editing of COST-Tables and COST-Views.

- The implementation of the CORD and COST models within the Object Prototyper has been cited in previous paragraphs as validation of the models themselves. However, the fact that these are implemented within the same tool with no special functionality, is the demonstration that an integrated environment for conceptual object models and summary table definitions is achievable. In adding a rich definition to the underlying data and making that definition readily addressable by applications processing that data, it is claimed that a significant distance along the route from Data to Information has been achieved and even a few extra steps have been taken further along the road from Information to Knowledge.

15.6. FURTHER WORK

It is normal at this stage to discuss further work that may be undertaken to pursue the research theme. Such discussion frequently serves to highlight deficiencies in the work already undertaken.

The main weakness in the existing model lies in the modelling of activities. Further work would need to define a dynamic modelling language which would be executable so that the life cycle of objects could be fully specified within the model and executable functions made available which could be called by applications. This would achieve the author's aim that databases should not respond to user commands such as:

```
Insert into Employee values ("John", 24)
```

but only to actions defined in the real world like:

```
Hire employee "John" aged 24
```

the nature and preconditions of the Hire method being defined in the DBMS not in the application software. Unfortunately this would require the creation of a compiler/interpreter to parse and execute the action specifications and their associated conditions, so it was not completed in the current phase of this research. In terms of the conceptual model, no extra structures or objects would be produced by this extra effort, and so it does not add to the validity or otherwise of the CORD model. Where it most impacts on the current work is that in including the functionality to define and execute actions it will become necessary to face up to the issues of time in CORD and in the SDM (Semantic Data Manager).

Time impacts on the work in two ways. One is the creation of historical databases which record all past transactions. This is not a serious implementation problem, as every triple in the SDM could have extra fields added with their start date and end date, and some scope variables could determine the point in time addressed by a query. This flexibility could not only cover the evolution of data, such as "What was John's address in 1991", but could also include the evolution of the conceptual model itself. Thus at one point in time salesmen might operate in a geographical region, at another they might be assigned by industry type. Whilst the evolution of data is simple to understand and the evolution of the model is simple to implement, what is less clear is how an evolving conceptual model is to be interpreted.

The second area which needs completion is the inclusion of a facility to execute statistical tables. Again, as with the integration of time, no major problem in implementation is envisaged, and no significant contribution to CORD or COST will have been made. The main problem with this implementation was that it would not have been sensible to undertake it without tackling some of the data

management issues raised in section 13.9. Whereas the SDM proved appropriate for the development of the Object Prototyper and good for managing conceptual models, it would be a clumsy tool for the management of large amounts of instance data. The development of an ancillary table-based storage system for large amounts of instance data would be required. This is not of itself a problem - Visual Basic is capable of defining MS Access tables at runtime - however the problem arises with the query processing. It would not be clear where data is stored or even whether it is stored in the same place, so all queries to the SDM would have to pass through some form of redirection. This was expected to be complex enough to warrant being omitted from the current phase of the research.

There are however some areas of further work which could be seen as more useful that those discussed above. One is simply the loading of more example conceptual models representing a wider range of real world scenarios to confirm the general applicability of CORD. Another is to write a better Object Query Language and a OQL interpreter for processing ad hoc queries and for executing path expressions. It would also seem advisable to revisit the issue of Domains and their definition in the CORD model. While the solution adopted was adequate for the demands placed on it by the COST models the author feels the need to review from the highest levels of abstraction the relationships between Collections and Domains and relationships between Domains.

Implications of the research for other areas of Information Systems

The issues which prompted this research have not been completely ignored by other research nor by the industry. The need to have access to transaction data from a variety of sources for management decision making is being addressed in a number of other ways. The development of Data Warehousing, the collection of transaction data on larger homogeneous database servers, is clearly one path in the same direction. In some respect Data Warehousing is simply a hardware issue enabling the dynamic transaction data to be isolated from the static reporting data, however there are common issues with this research. To be successful a

Data Warehouse needs to have the structure of the data defined and accessible to users and to users applications. The value of the data will be directly related to the richness of this description. CORD provides a framework which could be used to provide a user and application accessible model of the contents of a Data Warehouse. In the discussion of COST the issues of the storage of and access to summary information have also been explored, this is also an important productivity issue, and implementors of Data Warehouses could benefit from the CORD-COST approach to the management of the conceptual model and the summary table definitions, even if the underlying data remained in relational database tables.

An allied development which is also related to this work is that of OLAP (Online Analytic Processing). OLAP is a technology aimed at the generation of management information from transaction data. OLAP takes a multidimensional view of data and give users acces to multidimensional browsers. This again is a technology which could benefit from the CORD-COST approach. COST provides a technique for managing multidimensional summaries which are characteristic of OLAP implementations. The success of the CORD-COST approach suggests that OLAP reports could be managed in a similar way to statistical summaries from within the object model, and that any extra conceptual definition could be handled by an extension of the COST model.

The main extra contribution to COST from addressing OLAP issues would be the generalisation of the COSTed user interface.

These examples reflect the growing awareness of the need to support MIS with rich conceptual models and to integrate both models and applications within a common framework. This research has made a significant contribution in this direction.

15.7. CONCLUSIONS

The CORD model developed in this thesis does contribute to bridging the gap

between data and information. The COST model contributes to a considerable simplification of the description and management of summary tables. Together with the Object Prototyper and the COST-editor, this work has demonstrated that practical user friendly tools based on these models can be developed.

However it is the author's experience that knowledge extraction is hard work, and there is no intention to suggest that the provision of a model and a set of tools is sufficient to improve future management information systems and hence the decision making of managers in the future.

Analysts will still be required to squeeze as much description of the real world out of their sources as possible. They will still have to encode it into the database applications they are developing. Many of them will surely wish for the blissful days when there were only Entities, Relationships and relational tables and it was for managers to remember how the data they were using related to the real world in which they worked.

References

ACKOFF(1967) Ackoff R. L., 'Management Misinformation Systems', *Management Science*, 14 (4) 147-157

ADIBA(1988) Adiba M., Collet C., 'Management of Complex Objects as Dynamic Forms', *14th International Conference on Very Large Databases*, ACM-SIGMOD, pp. 134-147

ALSHAWI(1991) AlShawi A., 'Semantic and Organisation Considerations in Database Conceptual Modelling: The Semantic Conceptual Organisational Model', *The University of Aston*

ANTHONY(1965) Anthony R. N., *Planning and Control systems: A framework for analysis*, Graduate School of Business Administration, Harvard University

ASHWORTH(1992) Ashworth C., Slater L., *An Introduction to SSADM version 4*, McGraw-Hill International Series

AVISON(1988) Avison D. E., Hassan M. A., 'View Integration for a Combined and Extended ERM', *Aston University*, Department of Computer Science

AVISON(1992) Avison D. E., *Information Systems Development: A database approach 2nd ed.*, Alfred Waller, Henley-on-Thames

AVISON(1995) Avison D. E., Fitzgerald G., *Information Systems Development: Methodologies, Techniques and Tools 2dn Ed*, McGraw-Hill, Maidenhead,

BEER(1979) Beer S., *The Heart of the Enterprise*, Wiley

BEYNON-DAVIES(1996) Beynon-Davies P., *Database Systems*, Macmillan Press, Basingstoke

BLANNING(1995) Blanning R. W., Ram S., Wang R. Y., 'Information technologies and systems', *Decision Support Systems*, 13 () 219-221

CHEN(1976) Chen P. P., 'The Entity Relationship Model - Toward a Unified view of Data', *ACM Transactions on Database Systems*, 1 (1) 9-36

CIBORA(1981) Cibora C. U., 'information Systems and transactions architecture', *International Journal of Policy and Information Systems*, 5 (4) 305-324

COAD(1991) Coad P., Yourdon E., *Object-Oriented Analysis 2 Ed.,* Yourdon Press

CODD(1970) Codd E. F., 'A relational model of data for large shared data banks', *Communications of the ACM*, 16 (6 June 1970) 377-387.

CODD(1979) Codd E. F., 'Extending the Database relational Model to Capture More Meaning', *ACM Transactions on Database Systems*, 4 (4) 397-434

CURTICE(1981) Curtice R. M., Jones P. E., 'Fundamentals of data element definition', *SIGMOD Record*, pp. 49-55

DRETSKE(1981) Dretske F., *Knowledge and the flow of Information,* Blackwell, Oxford

ECKERT(1993) Eckert G., Kempe M., 'Modeling with Objects and Values in software development', *EPFL - CH 1015 Lausanne,* EPFL - CH 1015 Lausanne

ECKERT(1994A) Eckert G, Golder P., 'Improving Object Oriented Analysis', *Information and Software Technology*, 36 (2) 67-86

ECKERT(1994B) Eckert G., 'Types, Classes and Collections in Object Oriented Analysis', *ICRE94 International Conference on Requirements Engineering,* Colorado USA,, April 18-22 Colorado Springs

ELMASRI(1985) Elmasri R., Hevner A., Weeldreyer J., 'The category concept: An Extension to the entity -relationship model', *Data & Knowledge Engineering*, 1 (1) 75-116

ELMASRI(1992) Elmasri R. Kouramajian V., 'A temporal Query Language based on Conceptual entities and Roles', Pernul G., Tjoa A. M. (eds), *11th International Conference on the Entity-Relationship Approach,* Springer Verlag Berlin, pp. 375-388

FALKENBERG(1991) Falkenberg E. D., vander Pols R., vander Weide T., 'Understanding Process Structure Diagrams', *Information Systems*, 16 (4) 417-428

FELDMAN(1985) Feldman P., Fitzgerald G., 'Action Modelling: A Symmetry of Data and Behaviour Modelling', *4th British National Conference on Databases,* pp. 81-104

FROST(1982) Frost R. A., 'Binary-Relational Storage Structures', *The Computer Journal,* 25 (3) 358-367

GHOSH(1988) Ghosh S. P., 'Statistical Relational Model', *Information Sciences An International Journal*, pp. 338-355

GIGCH(1990) van Gigch J. P., Le Moigne J. L., 'The design of an organisation information system', *Information and Management*, pp. 325-331

HEUSER(1993) Heuser C. A., Peres E. M., Richter G., 'Towards a complete conceptual Model: Petri Nets and Entity Relationship Diagrams', *Information Systems*, 18 (5) 275-298

HICKS(1993) Hicks J. O., *Management Information Systems: A User Perspective 3rd ed.,* West Publishing Company, St Paul MN

HURFORD(1983) Hurford J. R., Heasley B., *Semantics a coursebook,* Cambridge

JACKSON(1983) Jackson M. A., *System Development,* Prentice/Hall International, Englewood Cliffs NJ

JAQUES(1976) Jaques E., *A General Theory of Bureaucracy,* Heinemann

JAQUES(1978) Jaques E., 'Levels of abstraction in mental activity', Jaques E.,Gibson R. O., Issac D. J. (eds) *Levels of abstraction in Logic and Human action*

JARDINE(1986) Jardine D. A., 'Semantic Agreement and the Communication of Knowledge', Steel T .B., Meersman R., *Data Semantics (DS-1),* North Holland, pp. 71-81

JARDINE(1987) Jardine D. A., van Griethuysen J. J., 'A Logic-based Information Modelling Language', *Data & Knowledge Engineering*, 2 () 59-81

KENT(1986) Kent W., 'The Realities of Data: Basic Properties of Data Reconsidered', Steel T .B., Meersman R., *Data Semantics (DS-1),* North Holland, pp. 175-188

KLUG(1982) Klug A., 'Equivalence of Relational Calculus Query Languages having Aggregate Functions', *Journal of the Association for Computing Machinery*, 29 (3) 699-717

KORZYBSKI(1933) Korzybski A., *Science and Sanity*, Science Press

KUNG(1993) Kung D. C., 'The Behaviour network model for conceptual information modelling', *Information Systems*, 18 (1) 1-21

LAKHAL(1989) Lakhal L., Cicchetti R., Miranda S., 'Complex-Statistical-Table Structure and Operators for Macro Statistical Databases', Litwin, Schek, *3rd Int Conf Foundations of Data Organisation and Algorithms*, Springer-Verlag LNCS V367, pp. 421-438

LAKHAL(1990) Lakhal L., Cicchetti R., 'STAR+ Un language de manipulation de resumes statistiques structures', *Recherche operationelle/Operations Research*, 24 (4) 365-432

LAWSON(1987) Lawson K. W., *A semantic modelling approach to knowledge based statistical software.*, Aston University Thesis

LAWSON(1991) Lawson K. W., Golder P. A., 'The use of semantic knowledge to enhance statistical software', *Journal of Applied Statistics*, 18 (1) 3-21

MADNICK(1995) Madnick S. E., 'Integration Technology: The reinvention of the linkage between information systems and computer science', *Decision Support Systems*, 13 () 373-380

MCCARTHY(1982) McCarthy J. L., 'Metadata Management for large Statistical Databases', *8th international conference on Very large Data Bases*, pp. 234-243

MCLEOD(1976) McLeod D. J., 'High Level Domain Definition in a relational data base system', *SIGPLAN Notices*, pp. 47-57

MINGERS(1996) Mingers J. C., 'An evaluation of theories of Infromation with regard to the semantic and Pragmatic Aspects of Information Systems', *Journal of Information Systems*, 6 () 187 209

NACE(1975) *Nomenclature of Activities for the European Community NACE (English language edition),* EUROSTAT, Luxembourg

ODMG(1993) Cattell R. (Ed), *The Object Database Standard: ODMG-93,* Morgan-Kaufmann, San Mateo, California, 1993

OLLE(1988) Olle T. W., Hagelstien J., McDonaald I. G., Roland C., van Assche F., *Information Systems Methodologies: A framework for understanding,* Addison Wesley, Reading MA

OZSOYOGLU(1989) Ozsoyoglu G., Matos V., Ozsoyoglu M., 'Query Processing Techniques in the Summary Table by Example Database Query Language', *ACM Transactions on Database Systems*, 14 (4) 526-573

PATERSON(1972) Paterson T. T., *Job Evaluation,* Business Books, London

PUT(1988) Put F., 'The ER approach extended with the action concept as a conceptual modelling tool', *7th International Conference on the entity Relationship approach,* pp. 283-300

RAFANELLI(1990) Rafanelli M., Shoshani A., 'STORM a statistical object representation model', *Fifth International workshop on statistical and Scientific databases,* SPRINGER VERLAG, pp. 14-29

RAFANELLI(1993) Rafanelli M., Ricci F. L., 'Mefisto: A functional Model for Statistical Entities', *IEEE transactions on Knowledge and Data Engineering*, 670-681

RISHE(1989) Rische N., 'Efficient Organisation of Semantic databases', Litwin, Schek, *3rd Int Conf Foundations of Data Organisation and Algorithms,* Springer-Verlag LNCS V367, pp. 114-125

ROWBOTTOM(1977) Rowbottom R., Billis B., 'The stratification of work and organisational design', Human Relations, 30 (1) 53-76

ROWBOTTOM(1987) Rowbottom R., Billis D., *Organisational design : the work-levels approach,* Gower

ROWBOTTOM(1989) Rowbottom R., Billis D., 'Level of work: New application to management in large organisations', *Journal of Applied Systems Analysis*, 16 () 19-34

RUMBAUGH(1991) Rumbaugh J., Blaha M., Premerlani W., Eddy F., lorensen W., *Object Oriented Modeling and Design,* Prentice/Hall International

SATO(1989) Sato H., 'A Data Model, knowledge Base and Natural Language Processing for Sharing a Large statistical Database', Rafanelli M., Kleinsin J. C., Svensson P., *4th International Conference on Statistical and Scientific Database Management,* Springer-Verlag Lecture Notes in Computer Science Vol. 339, pp. 207-225

SEN(1987) Sen A., Kerschberg L., 'Enterprise modeling for database specification and design', *Data & Knowledge Engineering*, 2 () 31-58

SHAHABUDDIN(1987) Shahabuddin, 'MIS: Do we still need it?', *International Journal of Policy and Information Systems*, 11 (2) 1-8

SHANNON(1949) Shannon C., Weaver W., *The Mathematical Theory of Communication,* University of Illinois Press, Urbana

SMITH(1977A) Smith J. M., Smith D. C. P., 'Database Abstractions: Aggregation and Generalisation', *ACM Transactions on Database Systems*, 2 (2) 105-133

SMITH(1977B) Smith J. M., Smith D. C. P., 'Database Abstractions: Aggregation', *Communications of the ACM*, 20 (6) 405-413

STERNBERG(1982) Sternberg R. J. (ed), *Handbook of Human Intelligence,* Cambridge University Press, Cambridge

SU(1983) Su S. Y. W., 'SAM* A semantic Association Model for Corporate and Scientific-Statistical Database', *Information Sciences An International Journal*, 29 () 151-199

TEOREY(1986) Teorey T. J., Yang D., Fry J. P., 'A Logical Design Methodology for Relational Databases using the Extended entity Relationship Model', *ACM Computing Surveys,* 18 (2) 197-222

URWICK(1943) Urwick L., *The elements of administration,* Pitman Publishing, London

WIERINGA(1991) Wieringa R., 'Steps towards a method for the formal modeling of dynamic objects', *Data & Knowledge Engineering*, 6 () 509-540

YADAV(1983) Yadav S. B., 'Determining an organisation's information requirements: A state of the art survey', DATABASE 'Database', 14 (3) 3-20

YULE(1957) Yule G. U., Kendall M. G., *An Introduction to the Theory of Statistics 14th Edition 3rd Impression,* Charles Griffin

ZALTA(1996a) Zalta E. N., 'The theory of abstract objects (http://mally.stanford.edu/theory.html)'

ZALTA(1996b) Zalta E., 'Principia Metaphysica (Draft) (http://mally.stanford.edu)'

Appendix 1

Cord Model

In this appendix and the next two, the various models developed and tested during this research are reproduced. Most elements have been discussed in context in the body of the thesis but it is still useful to present them together here.

The Object Prototyper can read in and store models in the form presented here. It also has the facility to print out the stored models to text files. The models presented here are the result of printing out the data stored in the database. Remembering that the SDM stores data and all associated meta data as a series of triples then the ability to read in and printout an equivalent copy is a validation that the Prototyper has correctly interpreted the model and is able to access the meta data correctly in order, firstly to understand what is being parsed and subsequently to generate a suitably structured output.

All the models presented in these appendixes have been read in and printed out and re-read in again as part of the validation process. The printing process produces models in a arbitrary order, alphabetical order of name. The validation process ensures that the elements of a schema do not have to be in any particular order. This confirms that the schema is declarative rather than procedural.

This first appendix presents the CORD model. As explained in Chapter 9, despite having developed a three pass process for reading in models it was not found possible to load the entire CORD model in one go. Some of the basic definitions concerning Properties and Attributes needed to be known to the Object Prototyper before more complex structures could be assimilated.

The model is in fact loaded in three stages.

Stage 1

When a new prototype is generated a certain minimum amount of information has to

be pre loaded. This is shown in the Model CORD_0. In practice this is not loaded by reading in the model as shown but is generated by a lower level process of writing a number of triples directly to the database without semantic checking taking place. The triples used to bootstrap the Object Prototyper as copied from the Visual Basic program, are reproduced in Appendix 4.

```
------------------------------------------------------------
--Model Cord_0 listed on 15/09/97 at 11:24:38
------------------------------------------------------------
  Model.Cord
  {
    Concept (Scope : global;)
    [
      Domain;
      Link;
      [
        Attribute (IsOrderedBy : Link.Property.SysOrder;)
        Property (Scope : global;)
        {
          Attribute.D1domain;
          Attribute.D2domain;
          Attribute.Default;
        }
        [
          IsInstanceOf (Concept;)
          IsKnownBy (;;SysName;)
          IsOrderedBy (Attribute;)
          Owner (Concept; ;Model.Cord;)
          Scope;
          SysName;
          SysOrder;
        ]
      ]
      Model;
      [
        Cord;
      ]
    ]  }
------------------------------------------------------------
--End of Model Cord
------------------------------------------------------------
```

As can be seen from the above print out of the model as stored in the database after initiation of an empty prototype the basic CORD structures of Models, Domains and Links have been defined. Two particular links have been specified, Attribute and Property. As properties are very important for defining the structure of CORD models the essential properties, Instantiation, Ownership and naming have also been

defined.

Stage 2

The second stage defines more of the basic elements to reach a point where the Object Prototyper can 'understand' attributes and properties and has some knowledge of domains. This is loaded as a model schema and is shown as model CORD_1.

```
------------------------------------------------------------
--Model Cord_1 listed on 15/09/97 at 11:26:18
------------------------------------------------------------
  Model.Cord
  {
    Concept (Scope : global;)
    [
      Class;
      Domain;
      {
        Attribute.Type(Domain_Type;);
        Attribute.Class;
      }
      [
        Domain_Type;
        [
          Primitive;
        ]
        Integer (Primitive;)
        Path (Primitive;)
        String (Primitive;)
      ]
      Link;
      [
        Attribute (IsOrderedBy : Link.Property.SysOrder;)
        {
          Attribute.Domain;
          Attribute.Maxcardinality (Integer;)
          Attribute.Mincardinality (Integer;)
          Attribute.Default (String;)
        }
        Function;
        Property (Scope : global;)
        {
          Attribute.D1domain;
          Attribute.D2domain (Property;)
          Attribute.Default;
        }
        [
          IsIndexed;
          IsInstanceOf (Concept;)
          IsKnownBy (Attribute; ;SysName;)
          IsOrderedBy (Attribute;)
          IsOwnedBy;
```

```
            IsPartOf (Class;)
            IsRoleOf (Class;)
            Owner (Concept; ;Model.Cord;)
            PartOf (;IsPartOf;)
            RoleOf (;IsRoleOf;)
            Scope (String;)
            SubType;
            SysName;
            SysOrder;
          ]
        ]
        Model;
        [
          Cord;
        ]
      ]
    }
------------------------------------------------------------
--End of Model Cord
------------------------------------------------------------
```

Note that as the above listing is generated from the database, it is not incremental but includes all definitions loaded in the initialisation stage CORD_0. As can be seen above this phase concentrates on further defining Attributes, and Properties and includes definitions of the Domain class together with some instances of primitive domain types.

Stage 3

The final stage is to load the rest of the CORD model which is shown as CORD_2.

```
------------------------------------------------------------
--Model Cord_2 listed on 15/09/97 at 11:32:02
------------------------------------------------------------
  Model.Cord
  {
    Concept (Scope : global;)
    [
      Class;
      {
        Function.Count (Integer;)
        Function.Distinct;
        Function.Exists (Boolean;)
      }
      Collection;
      {
        Attribute.Class;
        Attribute.Selection (String;)
        Attribute.Member;
      }
      Domain;
```

```
{
  Attribute.Type (Domain_Type;)
  Attribute.Class (Class;)
  Attribute.Domain (Domain;)
  Attribute.Constraint (String; 10;)
  Function.Count (Integer;)
  Function.Distinct;
  Function.Selection;
}
[
  Blob (Primitive;)
  Boolean (Primitive;)
  {
    Function.Count (Integer;)
    Function.Sum (Integer;)
  }
  Date (Primitive;)
  Domain_Type (List;)
  [
    Aggregation;
    List;
    Primitive;
    SubSet;
  ]
  Gender (List;)
  [
    Female;
    Male;
  ]
  Integer (Primitive;)
  {
    Function.Average (Integer;)
    Function.Count (Integer;)
    Function.Distinct;
    Function.Group;
    Function.Maximum (Integer;)
    Function.Minimum (Integer;)
    Function.Sum (Integer;)
  }
  Path (Primitive;)
  String (Primitive;)
  {
    Function.Count (Integer;)
    Function.length (Integer;)
  }
]
Link;
[
  Action (IsOrderedBy : Link.Attribute.Order;)
  {
    Attribute.Condition (String;)
    Attribute.Do (String;)
    Attribute.Order (Integer;)
  }
  Attribute (IsOrderedBy : Link.Property.SysOrder;)
  {
    Attribute.Domain;
```

```
            Attribute.Maxcardinality (Integer;)
            Attribute.Mincardinality (Integer;)
            Attribute.Default (String;)
            Attribute.Order (Integer;)
          }
          Classification;
          Function;
          {
            Attribute.Domain;
            Attribute.Arguement (String;)
            Attribute.Code (String;)
          }
          Parameter;
          Part;
          {
            Attribute.domain;
            Attribute.Maxcardinality (Integer;)
            Attribute.Mincardinality (Integer;)
          }
          Property (Scope : global;)
          {
            Attribute.D1domain;
            Attribute.D2domain (Property;)
            Attribute.Default;
          }
          [
            IsIndexed;
            IsInstanceOf (Concept;)
            IsKnownBy (Attribute; ;SysName;)
            IsOrderedBy (Attribute;)
            IsOwnedBy;
            IsPartOf (Class;)
            IsRoleOf (Class;)
            Owner (Concept; ;Model.Cord;)
            PartOf (;IsPartOf;)
            RoleOf (;IsRoleOf;)
            Scope (String;)
            SubType;
            SysName;
            SysOrder;
          ]
          Relationship;
          {
            Attribute.Member;
          }
        ]
        Model;
        {
          Attribute.includes (Model;)
        }
        [
          Cord;
        ]
      ]
    }
-------------------------------------------------------------
--End of Model Cord
```

```
------------------------------------------------------------
```

The final stage in loading CORD fills in the definition of various elements of the CORD model which are mainly of interest in application modelling such as Action, Relationships and Parts.

Regmod

At this stage CORD has been defined and other models can be loaded. As other models need to share CORD definitions they cannot be treated as completely independent and will not be loaded by the Object Prototyper unless it has been pre warned. This is carried out by loading a model called Regmod to register the existence of other models to CORD. These other models are usually loaded one at a time during testing but Regmod declares all models referred to in the thesis. This is shown in the partial model listed below.

```
------------------------------------------------------------
--Model Cord Regmod  listed on 15/09/97 at 11:50:29
------------------------------------------------------------
  Model.Cord
  {
    Concept (Scope : global;)
    [
      Model;
      [
        Cord;
        Cost (Cord;)
        Emp (Cord;)
        Publisher (Cord;)
      ]
    ]
  }
------------------------------------------------------------
--End of Model Cord
------------------------------------------------------------
```

In Appendix 2 the COST model is presented, and in Appendix 3 two test models are shown, Emp and Publisher.

Appendix 2

COST Model

In this appendix the COST model is reproduced. Although shown earlier in the thesis in full it appears here to confirm that it has been successfully read in and printed out again.

```
------------------------------------------------------------
--Model Cost listed on 15/09/97 at 11:49:28
------------------------------------------------------------
  Model.Cost
  {
    Class.Table;
    {
      Part.Table_Attribute;
      {
        Attribute.Type (Table_Attribute_Type;)
        Attribute.Path (Path;)
      }
    }
    Class.View;
    {
      Part.View_Attribute;
      {
        Attribute.Type (View_Attribute_Type;)
        Attribute.Parent (View_Attribute;)
        Attribute.Order (Integer;)
        Attribute.Source (Table_Attribute;)
      }
    }
    Domain.Table_Attribute_Type;
    [
      Category;
      Summary;
    ]
    Domain.View_Attribute_Type;
    [
      Cell;
      Column;
      Page;
      Row;
    ]
  }
------------------------------------------------------------
--End of Model Cost
------------------------------------------------------------
```

No further discussion is necessary, the model has been discussed fully in chapter 13.

Appendix 3

Test models

In this appendix the test models Emp and Publisher are reproduced.

```
------------------------------------------------------------
--Model Emp listed on 24/09/97 at 11:41:00
------------------------------------------------------------
  Model.Emp
  {
    Domain.age_group (List;)
    [
      10-20;
      21-30;
    ]
    Domain.Job_type (List;)
    [
      Manager;
      Salesman;
      Technical;
    ]
    Class.Engineer (SubType : Employee;)
    {
      Attribute.Skill (String;)
    }
    [
      Oswald;
      Peter;
    ]
    Domain.Age (SubSet; ;Integer;)
    {
      Classification.age_group (IsKnownBy : age_group;)
      {
        Attribute.lower (Integer;)
        Attribute.upper (Integer;)
        Attribute.age_group (age_group;)
      }
      [
        10-20 (10; 20;)
        21-30 (21; 30;)
      ]
    }
    Class.Union;
    {
      Attribute.name (String;)
    }
    Class.Employee (IsPartOf : Company; IsRoleOf : Person;)
    {
      Attribute.Salary (Integer;)
      Attribute.Department (Link.Part.Department;)
      Attribute.Children (Class.Employee;)
      Attribute.JobTitle (Job_type;)
    }
    [
      Col (PartOf : AlphaCo; RoleOf : colin;)
      Dave (PartOf : AlphaCo; RoleOf : david;)
```

```
  Elly (PartOf : BetaCo; RoleOf : ellen;)
]
Class.Manager (SubType : Employee;)
{
  Attribute.department (Link.Part.Department;)
}
[
  Quentin;
  Russell;
]
Class.Company;
{
  Attribute.Name (String;)
  Attribute.IndustryCode;
  Attribute.NumberOfEmployees (Integer;)
  Part.Department (Link.Part.Department; 99;)
}
[
  AlphaCo;
  BetaCo;
]
Class.Department;
{
  Attribute.Manager (Class.Person;)
  Attribute.Location (County;)
}
[
  Accounts (garth; Essex;)
  Production (harold; Kent;)
  Sales (fred; Sussex;)
]
Class.Customer (SubType : Person;)
{
  Action.PlacesOrder (;;1;)
}
[
  ACustomer;
  BCustomer;
]
Class.Union_member (IsPartOf : Union;
   IsRoleOf : Employee;)
{
  Attribute.Joined (Date;)
}
Class.Mechanic (SubType : Engineer;)
{
  Attribute.vehicle (String;)
}
Domain.Address (Aggregation;)
{
  Part.County (County;)
  Part.Postcode (String;)
  Part.Street (String;)
  Part.Town (String;)
}
Domain.County (List;)
{
```

```
    Classification.Region (IsKnownBy : County;)
    {
      Attribute.County (County;)
      Attribute.Region (Region;)
    }
    [
      Cornwall (South_West;)
      Cumbria (North;)
      Devon (South_West;)
      Essex (South_East;)
      Huntindon (Midlands;)
      Kent (South_East;)
      Lancashire (North;)
      Lincoln (Midlands;)
      Northumberland (North;)
      Oxfordshire (Midlands;)
      Shropshire (Midlands;)
      Staffordshire (Midlands;)
      Sussex (South_East;)
      Warwickshire (Midlands;)
    ]
  }
  [
    Cornwall;
    Cumbria;
    Devon;
    Essex;
    Huntindon;
    Kent;
    Lancashire;
    Lincoln;
    Northumberland;
    Oxfordshire;
    Shropshire;
    Staffordshire;
    Sussex;
    Warwickshire;
  ]
  Class.Region;
  {
    Attribute.Population (Integer;)
  }
  [
    Midlands (1234;)
    North (1234;)
    South_east (1234;)
    South_west (1234;)
  ]
  Domain.Region (List;)
  [
    Midlands;
    North;
    South_East;
    South_West;
  ]
  Class.Electrician (SubType : Engineer;)
  {
```

```
    Attribute.specialty (String;)
  }
  Class.Person (IsKnownBy : Name;)
  {
    Attribute.Name (String;)
    Attribute.Age (Age;)
    Attribute.Sex (Gender;)
    Attribute.Status (Marital_Status;)
    Attribute.Address (Address;)
  }
  [
    andy (25; Male; Single;)
    bill (19; ;Single;)
    colin (59; Male; Married;)
    david (19; Male; Single;)
    ellen (26; Female; Single;)
    fred (23; Male; Single;)
    garth (29; Male; Married;)
    harold (19; Male; Single;)
    ian (39; Male; Married;)
    janet (29; Female; Married;)
    keith (49; Male; Divorced;)
    leonie (35; Female; Divorced;)
    sue (48; Female; Divorced;)
  ]
  Class.Salesman (SubType : Employee;)
  {
    Attribute.Region (Region;)
    Attribute.Commission (Integer;)
  }
  [
    Malcolm;
    Nancy;
  ]
  Domain.Marital_Status (List;)
  {
    Divorced;
    Married;
    Separated;
    Single;
    Widowed;
  ]
}
------------------------------------------------------------
--End of Model Emp
------------------------------------------------------------
```

Whilst not presented as a model of any reality and having some unlikely definitions, Emp contains the typical elements of any real world modelling and has been constructed to exercise Instantiation, Domains, Ownership Roles, Parts, Classifications and Subtypes and all the other parts of the CORD model. The different parts of Emp have occurred in many places throughout the thesis as examples of

issues being discussed. It is presented here to show that it is a consistent definition in that it can be read in and printed out again. The output order is rather arbitrary and not in the logical form that the analyst might have created the original definition; however as the modelling language is declarative the order is not significant.

As stated earlier the Publisher model was created because Emp could not easily plausibly accommodate any more PartsOf constructs.

```
------------------------------------------------------------
--Model Publisher listed on 15/09/97 at 11:55:13
------------------------------------------------------------
  Model.Publisher
  {
    Domain.series_type (List;)
    [
      Chemistry;
      Economics;
      Physics;
    ]
    Class.author;
    {
      Attribute.name (String;)
      Attribute.address (String;)
    }
    Class.series;
    {
      Attribute.title (String;)
      Attribute.type (series_type;)
    }
    {
      Class.book (IsPartOf : Class.series;)
      {
        Attribute.title (String;)
        Attribute.author (author;)
      }
      {
        Class.appendix (IsPartOf : Class.book;)
        {
          Attribute.title (String;)
        }
        Class.chapter (IsPartOf : Class.book;)
        {
          Attribute.title (String;)
          Attribute.number (Integer;)
          Attribute.pagecount (Integer;)
        }
        {
          Class.picture (IsPartOf : Class.chapter;)
          {
            Attribute.caption (String;)
            Attribute.image (Blob;)
```

```
            }
            Class.paragraph (IsPartOf : Class.chapter;)
            {
              Attribute.text (String;)
            }
          }
        }
      }
    }
------------------------------------------------------------
--End of Model Publisher
------------------------------------------------------------
```

Again, Publisher has been read in and printed out again to produce the listing reproduced above.

Appendix 4

The initial triples

In this appendix the triple used to populate a new empty prototype are listed in the form they appear in the Visual Basic program which is the Object Prototyper. As they are reasonably self explanatory the actual code can be reproduced here.

```
Add_Triple CONCEPT_ID, SYSNAME_ID, Concept_name
Add_Triple CONCEPT_ID, ISINSTANCEOF_ID, "*"
Add_Triple CONCEPT_ID, OWNER_ID, CORD_ID
Add_Triple LINK_ID, ISINSTANCEOF_ID, CONCEPT_ID
Add_Triple LINK_ID, SYSNAME_ID, Link_name
Add_Triple LINK_ID, OWNER_ID, CORD_ID
Add_Triple CORD_ID, SYSNAME_ID, Cord_name
Add_Triple CORD_ID, ISINSTANCEOF_ID, MODEL_ID
Add_Triple CORD_ID, OWNER_ID, CORD_ID
Add_Triple MODEL_ID, SYSNAME_ID, Model_name
Add_Triple MODEL_ID, ISINSTANCEOF_ID, CONCEPT_ID
Add_Triple MODEL_ID, OWNER_ID, CORD_ID
Add_Triple SYSNAME_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple SYSNAME_ID, SYSNAME_ID, SysName_name
Add_Triple SYSNAME_ID, OWNER_ID, CORD_ID
Add_Triple SYSORDER_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple SYSORDER_ID, SYSNAME_ID, SysOrder_name
Add_Triple SYSORDER_ID, OWNER_ID, CORD_ID
Add_Triple OWNER_ID, SYSNAME_ID, Owner_name
Add_Triple OWNER_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple OWNER_ID, OWNER_ID, CORD_ID
Add_Triple OWNER_ID, D1DOMAIN_ID, CONCEPT_ID
Add_Triple OWNER_ID, DEFAULT_ID, CORD_ID
Add_Triple ISINSTANCEOF_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple ISINSTANCEOF_ID, SYSNAME_ID, Isinstanceof_name
Add_Triple ISINSTANCEOF_ID, OWNER_ID, CORD_ID
Add_Triple ISINSTANCEOF_ID, D1DOMAIN_ID, CONCEPT_ID
Add_Triple ATTRIBUTE_ID, ISINSTANCEOF_ID, LINK_ID
Add_Triple ATTRIBUTE_ID, SYSNAME_ID, Attribute_name
Add_Triple ATTRIBUTE_ID, OWNER_ID, CORD_ID
Add_Triple ATTRIBUTE_ID, ISORDEREDBY_ID, SYSORDER_ID
Add_Triple PROPERTY_ID, ISINSTANCEOF_ID, LINK_ID
Add_Triple PROPERTY_ID, SYSNAME_ID, Property_name
Add_Triple PROPERTY_ID, OWNER_ID, CORD_ID
Add_Triple ISORDEREDBY_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple ISORDEREDBY_ID, SYSNAME_ID, IsOrderedBy_name
Add_Triple ISORDEREDBY_ID, OWNER_ID, CORD_ID
Add_Triple ISORDEREDBY_ID, D1DOMAIN_ID, ATTRIBUTE_ID
Add_Triple ISKNOWNBY_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple ISKNOWNBY_ID, SYSNAME_ID, IsKnownby_name
Add_Triple ISKNOWNBY_ID, OWNER_ID, CORD_ID
Add_Triple ISKNOWNBY_ID, DEFAULT_ID, SYSNAME_ID
Add_Triple SCOPE_ID, ISINSTANCEOF_ID, PROPERTY_ID
Add_Triple SCOPE_ID, SYSNAME_ID, Scope_name
Add_Triple SCOPE_ID, OWNER_ID, CORD_ID
Add_Triple D1DOMAIN_ID, ISINSTANCEOF_ID, ATTRIBUTE_ID
Add_Triple D1DOMAIN_ID, SYSNAME_ID, D1Domain_name
```

```
Add_Triple D1DOMAIN_ID, OWNER_ID, PROPERTY_ID
Add_Triple D1DOMAIN_ID, SYSORDER_ID, 1
Add_Triple DEFAULT_ID, ISINSTANCEOF_ID, ATTRIBUTE_ID
Add_Triple DEFAULT_ID, SYSNAME_ID, Default_name
Add_Triple DEFAULT_ID, OWNER_ID, PROPERTY_ID
Add_Triple DEFAULT_ID, SYSORDER_ID, 5
Add_Triple PROPERTY_ID, SCOPE_ID, "global"
Add_Triple CONCEPT_ID, SCOPE_ID, "global"
Add_Triple DOMAIN_ID, ISINSTANCEOF_ID, CONCEPT_ID
Add_Triple DOMAIN_ID, SYSNAME_ID, Domain_name
Add_Triple DOMAIN_ID, OWNER_ID, CORD_ID
```

The result of this pre-loading can be seen by examining CORD_0 in appendix 1.

Appendix 5

Examples of COST models

The following print out represents the CORD version of the example COST-Table and COST-View shown in figures 14.3 and 14.4

```
------------------------------------------------------------
--Model Emp listed on 21/09/97 at 22:03:40
------------------------------------------------------------
  Model.Emp
  {
    Class.Person (IsKnownBy : Link.Attribute.Name;)
    {
      Attribute.Name (String;)
      Attribute.Age (Age;)
      Attribute.Sex (Gender;)
      Attribute.Status (Marital_Status;)
      Attribute.Address (Address;)
    }
    {
      Table.person_table_1;
      {
        Table_Attribute.County_count (Summary;
           Address>County>Count;)
        Table_Attribute.Region (Category;
           Address>County>Region;)
        Table_Attribute.Ave_age (Summary; Age>Average;)
        Table_Attribute.Age_group (Category; Age>age_group;)
        Table_Attribute.Sex (Category; Sex;)
        Table_Attribute.Status (Category; Status;)
        View.person_view_2;
        {
          View_Attribute.Ave_age (Cell; ;1; Ave_age;)
          View_Attribute.Status (Column; Age_group; 2; Status;)
          View_Attribute.Sex (Column; Age_group; 1; Sex;)
          View_Attribute.Region (Row; ;1; Region;)
          View_Attribute.Age_group (Column; ;1; Age_group;)
        }
      }
    }
-further elements in the definition of Employee have been omitted here
}
------------------------------------------------------------
--End of Model Emp
------------------------------------------------------------
```

Instead of using the graphical user interfaces to develop the table and view it would of course be possible to load a text file containing the above schema. The user would lose the benefit of the knowledge about tables and views built into the COSTed program.

Appendix 6

Detailed Contents List