A CLASSIFICATION THEORY BASED INFORMATION SYSTEMS MODEL

By

Byron Jeffrey Parsons

B.Com., Memorial University of Newfoundland, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Commerce and Business Administration)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 1991

ABSTRACT

Information systems (IS) development is viewed as a process of transforming users' knowledge about some subject matter into a computer-based system which faithfully represents that knowledge. A critical step in this process is ***conceptual modelling*** - the development of an implementation-independent representation of the relevant knowledge. While the importance of conceptual modelling has gained increasing recognition, many existing conceptual models remain based on software, rather than knowledge, constructs.

This research adopts the premise that a conceptual model should provide constructs for *directly* modelling knowledge. Since the subject matter of organizational IS is typically *things in organizations*, theories of ***concepts (classification)***, which deal with the structure and organization of knowledge about things, are an appropriate source of modelling constructs. A *classical* theory of concepts suggests five knowledge constructs - instance, property, concept, specialization, and composition.

The thesis develops formal definitions of each of these constructs. The notion of *direct correspondence* is then used to define a conceptual information systems model called MIMIC, which contains a corresponding set of constructs - object, attribute, class, specialized class, and composite class.

The model offers several contributions to conceptual modelling research and practice, including:

1) minimal requirements for a "good" class structure;
2) naturalness of a lattice structure for class organization;
3) refinements to the meaning of IS-A connections between classes in a lattice;
4) distinction between simple relationships and those which can be regarded as objects;
5) simple foundation for treating time in conceptual modelling; and
6) a normative model of objects under the assumption that a fundamental objective of the object paradigm of computing is to provide a set of natural modelling constructs.

The value of the model is further illustrated by using it as a framework to evaluate several other conceptual modelling approaches. The results of this comparison indicate that, while other models do support cognitive constructs to varying degrees, each is weak in supporting some elements of the classical view of concepts. Finally, a detailed example is used to demonstrate the capability of the model for uniformly representing knowledge across several classes of applications.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I am happy to have the opportunity to thank the people who have supported me in various ways during the completion of this research.

I am indebted to Yair Wand for introducing me to research in information systems modelling and for providing insightful and constructive supervision of this thesis. I also thank the other members of my thesis committee, Alain Fournier and Carson Woo, for suggestions which helped me to focus the research. In addition, I am grateful for the stimulating intellectual environment provided by participants in the MIS Workshop series at the University of British Columbia.

It gives me greatest pleasure to acknowledge those whose personal support helped me weather each storm that arose over the past several years. Thanks to Dave and Jane for many interesting lunches. To Katherine, words cannot express how much your friendship has given me the strength to press on. Thanks for putting up with me. Finally, I want to thank my parents for the unending source of love and support they have been. Thanks for always being there when I needed someone to talk to.

CHAPTER 1

INTRODUCTION

1.1 TRADITIONAL VIEW OF INFORMATION SYSTEMS

In the 40 or so years since the introduction of computers to businesses, there has been a proliferation of applications and rapid advances in both hardware and software technology. Initial commercial uses involved primarily high-volume, transaction-oriented data processing in a few areas of an organization (e.g., accounting and inventory). Today, applications encompass data processing, management reporting, and decision support in all functional areas (e.g., purchasing, production, marketing, personnel) and at all managerial levels (operational, tactical, strategic) of a firm. In addition, advances in technology have created new opportunities and challenges in areas such as database management and the development of knowledge-based systems.

Throughout this period, though, the accepted view of what an *information system* (IS) is from a software point of view[1] has remained relatively unchanged. A typical diagram of an IS, similar to those found in introductory textbooks, is given below:

INFORMATION SYSTEM

INPUT ----> PROCESSING ----> OUTPUT

In this elementary model, there are essentially two components: data (input and output) and programs (processes). These are created and maintained independently and distinctly, and interact only

[1]Of course, an IS may also be viewed as a social system (e.g., London, 1976), or as a physical system. However, these aspects are outside the scope of this research.

during the execution of a program. Data capture facts about the world. Programs accept input data, transform data, and generate output data, either for storage or for use by a human or machine.

This orientation is reinforced by many of the tools and methodologies used in systems development. For example, many high level programming languages, such as COBOL and PASCAL, encourage the "data-program" dichotomization by separating the development of procedures and algorithms from the development of data structures. In addition, database management systems, by emphasizing data independence, contribute to the rift between data and programs. Furthermore, the most widely used systems development methodologies, such as structured analysis (DeMarco, 1979), support this programming orientation by handling processes and data quite separately during systems analysis and design.

This traditional separation of programs and data is recognized to be a source of difficulty in information systems development. According to Love (in Pressman, 1987):

> [t]he roots of software problems may lie in the most traditional terms for describing our industry - data processing. We have been taught that data and programs are two distinct "things" which are somehow fundamental to our business. That partitioning may be far more detrimental than we realize. (p. 364)

Several problems arise when the "data processing" view of IS is taken. All are rooted in the fact that there is not a one-to-one correspondence between things in the domain being modelled (e.g., a company) and things in the information system, in part due to the artificial separation of application data and programs which act on that data. As a consequence, a rather significant transformation is required in moving from *domain-oriented* constructs, which are familiar to the user, to *software-oriented* constructs suitable for implementation.

The most obvious problem is that there is nothing in the structure of files to indicate whether the information in a record relates to a single entity, several entities, or a relationship among several entities. Although data fields capture the values of properties of things, a single record in a file may contain fields describing any number of things (e.g., customers and accounts). In addition, one file may contain fields describing a relationship among several types of things (e.g., customers holding accounts), while another file may describe only things from a single class (e.g., customers). Furthermore, information about a single entity may be distributed among several files. In short, the *meaning* of the data has to be largely interpreted by a user based on knowledge of the domain. As well, there is no simple way of determining which data files may be used by which programs. There is no explicit recognition that the execution of procedures to change data values often reflects actual or potential changes to properties of entities.

This type of problem occurs frequently in record-oriented IS (Kent, 1978; 1979). The lack of direct correspondence between data records (and/or procedures which act on them) and distinct things in the environment of the IS makes it difficult to build and understand systems. As Kent (1979) states:

> the use of record structures depends on supplementary information, often reflected only in the special-purpose application programs written to process the data, and which may or may not still be remembered by the users of the data. (p. 107)

A second problem with the traditional view of IS is that there may be considerable duplication of development effort and, also, of programs and data. Systems development projects are often undertaken quite independently of each other. Solving the problem may be equated with writing the software to perform a particular task. As a result, applications are developed in an ad hoc manner. New data files may be constructed which contain data that already exist in files built for other applications[2]. In addition to this obvious duplication of data, the failure to take advantage of existing systems and files may result in unnecessary systems analysis and design efforts.

When a new application belongs to a different class from existing applications using data about the same entities, a closely related problem can occur. Not only may data be duplicated, but new programs

[2]This is a problem which database research has sought to address by providing application-independent representations of data. In particular, the reduction of duplication is a major justification for the database approach. However, as noted earlier, databases encourage the separation of data from the procedures which modify it. In addition, the redundancy mentioned here is more general, encompassing the duplication that occurs when applications as diverse as transaction processing and expert systems use data about the same entities. Such applications generally use different software technologies and data formats, leading to duplication.

may be developed to describe essentially the same activities as existing programs. As a result, IS development is fragmented across different classes of applications, again resulting in redundancy in development efforts. For example, consider a system to simulate arrivals of goods at a warehouse and subsequent shipments to customers. A simulation generates hypothetical shipments to and from the warehouse that can be handled in the same way as actual shipments that are processed by an existing inventory program (e.g., replenishment and depletion of quantity-on-hand). In this case, simulation and transaction processing constitute two *classes* of applications, characterized by different goals. Nevertheless, code for processing shipments into and out of inventory may be very similar for inventory tracking and simulation software.

In short, the traditional data processing view of IS creates problems of *comprehensibility, fragmentation, and duplication.* Although these problems are most obvious in IS implementation, they also impede earlier stages of systems development. A fundamental shift in perspective is required to support development of IS to cope with growing user and application demands. The next two sections argue that a logical place for the shift in perspective to originate is in systems analysis methodologies.

1.2 IMPACT OF TRADITIONAL VIEW ON SYSTEMS ANALYSIS

The view of an information system as a collection of software and data components has had a significant influence on the nature of systems development approaches. Many analysis and design methodologies are based on constructs closely related to files and programs. Perhaps the best known of these is *structured analysis* (DeMarco, 1979; Gane and Sarson, 1979), which is based on *data flows* and *processes* which transform these data flows. However, many other approaches also appear to be heavily influenced by implementation considerations (e.g., Jackson, 1983; Verheijen and van Bekkum, 1982: see [Olle et al., 1982; 1983; 1986] for descriptions and comparisons of a variety of methodologies).

Many methodologies fail to explicitly recognize that data and programs constitute a model or representation of some aspects of the domain or problem area constituting the subject matter of an information system. Consequently, they cannot be expected to provide a representation which can be easily understood by users who are immersed in the problem domain but are not well-versed in software technology (Sibley, 1986). The fundamental premise of the research in this dissertation is that:

> ***a systems analysis methodology should provide a set of constructs for directly modelling the subject matter of information systems.***

This premise is consistent with a view of IS development as the process of transforming knowledge about a domain (e.g., an organization) into an implemented system that properly reflects that knowledge (Wand and Weber, 1989). At early stages (referred to by terms such as *conceptual modelling* and *enterprise analysis*), it is particularly important that descriptions be based on constructs familiar to those who regularly deal with the subject matter and/or will be users of the implemented system. This should facilitate the verification of specifications so that the risk of problems in later stages of development is reduced. A methodology which conforms to the above premise will directly reflect the constructs of the domain of information systems, giving rise to a competing view of IS as representations of knowledge about some "world" such as an organization.

1.3 AN ALTERNATE VIEW AND RESEARCH PROBLEM

In recent years, several researchers have explicitly recognized that an IS can be viewed as a representation of some elements of reality (Wand and Weber, 1988; Bubenko, 1986; Borgida et al., 1985; Jackson, 1983; Nijssen, 1976). For example, Nijssen (1976) states that:

> [t]he origin of an information system has to do with the need of one and usually more persons to collect, note, process and distribute data, related to a certain part of the reality. (p. 2)

Given this view, information systems development can be regarded as the construction of a model of the target segment of reality. Key development issues can then be addressed by determining how best to identify and represent the important elements of the domain.

To be more precise, though, we know reality only as we perceive and conceive it. Therefore, an IS actually represents aspects of our *knowledge* (i.e., cognitive representation) of things in some part of reality. Wand and Weber (1988) hint at this distinction by claiming that:

> [a]n information system is an artificial representation of a real-world system *as perceived by humans*. (p.213, italics mine)

They then use this proposition to justify an ontological approach to developing a theoretical foundation for IS. In doing so, they focus on a model of the nature of reality, implicitly assuming that perception is selective, but does not distort. This assumption is identified by Lakoff (1987) as central to what he calls the "objectivist paradigm". He characterizes "objectivist cognition" as follows:

> Thought is the manipulation of abstract symbols. Symbols get their meaning via correspondence to entities and categories in the world. In this way, the mind can represent external reality and be said to "mirror nature". (p. 165)

A number of *conceptual modelling* methodologies have been developed which draw from knowledge representation principles (e.g., Mylopoulos, 1991; Mylopoulos et al., 1990; Greenspan and Mylopoulos, 1984; Hammer and McLeod, 1981; Bubenko, 1980). Despite this growth of interest in an important early stage in systems development (e.g., Bubenko, 1986; Brodie et al., 1984), such approaches are not explicitly or formally derived from a theory of the structure and organization of knowledge about things, and no model has emerged as dominant among those which have been proposed. Chapter 2 reviews a number of conceptual modelling approaches and highlights the need for greater emphasis on justifying the choice of representation constructs.

This thesis endeavours to show that additional insight into IS development (in particular, conceptual modelling) can be achieved by explicitly considering the nature of human *cognition*, especially in the area of representing knowledge about things or entities[3]. If an IS is viewed as a representation of human understanding of an organizational reality which is used by humans, a theory of how people

[3]The terms *thing* and *entity* are used interchangeably here. Both refer to existents in a reality (such as an organization).

represent the world can serve as a natural foundation for developing methodologies best suited for conceptual modelling[4]. Such a theory need not be one of general human cognition, since information systems represent only a small segment of human knowledge - that pertaining to things in organizations[5]. Moreover, since things in organizations (as well as in everyday life) are understood by classifying them into relevant categories, a theory of *concepts and classification* may be adequate to deal with the knowledge typically contained in IS.

The field of cognitive science offers a number of theories of *concepts and classification* that attempt to describe the bases for organizing things in the perceived world into meaningful categories. In general, these theories attempt to:

1) define what concepts are,

2) explain how concepts are acquired, and

3) account for the ways in which concepts help us survive by reducing complexity.

Chapter 3 describes the importance of concepts and categorization in organizing knowledge about things, reviews three theories of concepts, and explores the suitability of one of these for representing knowledge about things in organizations[6]. Given the assumption that an IS represents knowledge, a conceptual model[7] developed from a cognitive foundation may be instrumental in allowing systems analysts to construct representations that can be more easily understood by users, so that errors and omissions in initial descriptions can be detected and corrected at minimal cost.

---

[4]If IS development is the process of transforming from knowledge of a domain to an implemented system which represents this knowledge, the products of intermediate stages of development - including the conceptual model - also represent domain knowledge.

[5]A general theory of cognition would deal also with learning, planning, and other cognitive activities.

[6]This knowledge constitutes the subject matter of various kinds of information systems.

[7]Interestingly, the term *conceptual model* embodies the term *concept*. Hence, it is somewhat surprising that existing conceptual models do not make more explicit and rigorous use of theories of concepts developed by cognitive scientists.

Interestingly, one recent approach to modelling, referred to as *object-oriented analysis*, appears to be implicitly based on a theory of categorization. However, a lack of agreement about exactly what the methodology should involve, and its underlying foundations, raises the issue of whether a grounded formal approach to defining an object paradigm can help resolve the debate over exactly what is meant by the term *object-oriented*, as discussed next.

1.4 ROLE OF THE OBJECT PARADIGM

Recently, a new software paradigm has emerged and is beginning to challenge the traditional data/program view. It is referred to in the context of programming as *object-oriented programming* and, more generally, as the *object paradigm* (Nierstrasz, 1987; Pascoe, 1986; Stefik and Bobrow, 1986). Of greatest interest here is the emergence of interest in object-oriented systems analysis and design (Booch, 1991; Coad and Yourdon, 1991; Bailin, 1990; Henderson-Sellers and Edwards, 1990) and object-oriented data models (e.g., Banerjee et al., 1987; Fishman et al., 1987).

The essence of the paradigm appears to be the encapsulation of data and procedures in a single unit or package. In addition, objects are intended to be in one-to-one correspondence with things in the problem domain. Beyond these two principles, however, there is widespread disagreement as to what features the paradigm should include (e.g., compare Nierstrasz, 1987 with Stroustrup, 1986).

Although the justifications most often forwarded for object constructs are technical and implementation-based, various claims have also been made about the "naturalness" of objects, suggesting that there may be an implicit theory of concepts underlying object-oriented approaches (e.g., Coad and Yourdon, 1991). However, the lack of consensus on an object model may in part be due to a confounding of technical and "naturalness" advantages. Thus, while the focus of this research is not the object paradigm, it is proposed that a theory of concepts can be used to suggest what objects should be, resulting in an object model that is *explicitly* based on the principle of providing natural representations of things in a domain of interest.

1.5 RESEARCH OBJECTIVES AND METHODOLOGY

In accordance with the research problem outlined above, the overall purpose of this research is to answer the general questions:

Can a theory of concepts be used as a foundation to define a uniform formal model of knowledge about entities that supports "direct" representation across several classes of information systems applications?

Does such a model provide insights and guidelines for conceptual modelling relative to existing models?

To answer these questions, the research seeks to achieve the following specific objectives:

1. *To develop a precise formal (object) model for directly representing cognitive constructs.*

   This will provide a mechanism for directly representing knowledge about the entities in an organization. Chapter 4 presents a formal definition of the components of the model. The purpose of the formalization is to define a set of object constructs in precise and unambiguous terms based on an underlying theory of knowledge (thereby providing a normative model). Appendix 1 illustrates the use of the model on several applications in a banking environment, and shows its value in uniformly dealing with several classes of IS applications.

2. *To explore the model as a tool for addressing open problems in conceptual modelling.*

   Existing conceptual models do not deal clearly with a number of issues related to the semantics and usefulness of constructs such as classes, class specialization, and time. Chapter 5 identifies several contributions of the proposed model to understanding and resolving important questions about conceptual model semantics, and presents a number of guidelines for modelling which emerge from this analysis.

3. *To evaluate the model by comparing it to several other conceptual modelling approaches.* The purpose of this comparison is to assess the degree to which the constructs of the proposed model are supported by existing conceptual models which are not formally grounded in theories of concepts. If the model is to be valuable, it should be at least as expressive as existing models, without being more complex. Chapter 6 contains this evaluation.

The methodological approach taken in this research has two components. The first may be described as *formal model building* and involves the development of a small set of constructs for representing knowledge according to a theory of concepts. The second is *model evaluation*, which involves both a discussion of the insights offered by the model for representing knowledge about a domain and an assessment of its representation power with respect to several existing models.

1.6 EXPECTED CONTRIBUTIONS

The anticipated contributions of this research are both theoretical and practical. On the theoretical side, the value of the work is primarily in providing a comprehensive, uniform model of knowledge which offers a number of insights into the semantics of conceptual modelling constructs. This model provides a vocabulary for describing a subject matter in the early stages of IS development. Additionally, it is viewed as a model of the object paradigm that emphasizes the essence of objects in supporting direct representation. Existing literature has provided conflicting views of what the object paradigm is and/or should be. This thesis suggests a precise resolution based on the modelling objective of representing concepts and instances of concepts. Finally, the model provides a framework for understanding and evaluating the constructs of existing conceptual models.

Beyond this, an important practical outcome of the research is the demonstration of the model's usefulness in representing knowledge for several kinds of applications. In this way, the modelling constructs are shown to provide a uniform tool for describing several classes of applications, in which a single object corresponds to a single instance of a concept across application areas.

Through these contributions, the work should help address the stated problems associated with the traditional data-program dichotomy in IS. First, since the model is based on a theory of the structure and organization of knowledge, it is expected that specifications created from it will be more easily understood and verified by users since the constructs employed by the model reflect the theory's view of the way people represent the world[8]. Second, just as human knowledge appears to be integrated, the representation of knowledge about one thing by one object, which can participate in several types of applications, facilitates greater integration (less fragmentation) at the conceptual modelling stage of IS development. Consequently, use of the model may result in less duplication of development effort, and less redundancy in resulting systems.

The work also provides a starting point from which a program of future research can be developed. First, an *implementation of the model* might prove useful in reducing the "semantic gap" between constructs oriented toward users' knowledge of a problem area and constructs used to implement IS by providing implementation primitives which are based on the model. Second, either the model or an implementation of it may be used as a tool for conducting *experiments on the **naturalness** of the object paradigm* for users, a frequently claimed - but rarely justified - benefit of object-oriented computing. Third, the model may provide a foundation for *developing a methodology* for object-oriented systems analysis and design for integrated systems. This would be accomplished by further developing the contributions and guidelines which emerge in Chapter 5, and would be valuable since it is currently not at all clear how to handle issues such as determining appropriate classes when modelling an enterprise. Chapter 7 summarizes contributions and limitations of the research, and outlines areas for future work.

[8]This will hold to the extent that the theory of concepts on which the model is based is a suitable theory of knowledge for the domain of interest. Chapter 3 elaborates on this issue.

CHAPTER 2

CONCEPTUAL INFORMATION SYSTEMS MODELLING

2.1 INTRODUCTION

2.1.1 Conceptual Modelling and Models

It is widely recognized that the development of large-scale information systems proceeds through a number of stages, the collection of which is generally referred to as the *systems development life cycle*[9] (e.g., Davis and Olson, 1985; Wasserman et al., 1983). While the number and naming of these stages varies among systems development methodologies, most approaches can be divided broadly into *analysis*, *design*, and *implementation*. These stages are necessary because of the semantic gap between implementation constructs, such as data and programs, and the constructs by which users describe the domain modelled by the IS (Greenspan and Mylopoulos, 1984; Hammer and McLeod, 1981; Kent 1978).

Recently, the importance of developing models and languages for expressing users' understanding of the things in a domain of interest, as an early step in IS development, has gained widespread attention (e.g., Borgida et al., 1988; Olle et al., 1986; Brodie et al., 1984; Nijssen, 1976). The products of this research have variously been referred to as *conceptual models* (Bubenko, 1986; Brodie et al., 1984), *information systems models* (Bubenko, 1980), and *semantic data models* (Hull and King, 1987).

Despite this recognition of the importance of modelling users' knowledge about a domain, there is no general definition of *conceptual model*, or of related terms. According to Bubenko (1980, p.399), a conceptual information model is "an abstract model of the enterprise", while Schmid (1977, p.121) describes a conceptual model as a model which "is suitable to represent somebody's interpretation of some

[9]For certain kinds of applications, other approaches to development, such as *prototyping*, may be more appropriate (e.g., see Davis and Olson, 1985).

slice of reality". Similarly, Mylopoulos (1991) states that "*conceptual modelling* is the activity of *formally* describing some aspects of the physical and social world around us *for purposes of understanding and communication*" (italics original), and views a conceptual model as a notation for expressing such descriptions.

These definitions suggest that a conceptual model either represents reality directly or someone's knowledge (interpretation) of it. The models reviewed in this chapter predominantly adopt the first position, in the sense that the importance of the structure of knowledge in guiding the development of a conceptual model is not explicitly considered. By contrast, much of this thesis advances the argument that the second position leads to modelling insights which may be obscured when the role of human perception and cognition is not explicitly considered. Consequently, a ***conceptual model*** is, for this discussion, defined as ***a set of constructs for representing knowledge about things in some domain.*** A specific representation (e.g., for a particular organization) created from such a set of constructs will be referred to as an *instance* (or *manifestation*) of the model. In this chapter, attention is focused on the nature and origin of the constructs proposed for a variety of conceptual models[10].

2.1.2 Importance of Conceptual Models

Conceptual models provide constructs for creating high-level, implementation-independent representations of information about things in, or otherwise related to, organizations, in order to support the systems development process. According to Kung and Solvberg (1986), an instance of a conceptual model has the following uses:

1) ***It serves as a common reference framework, which is used during the systems analysis phase to communicate with the future users of the system.***

[10]This means that there is no discussion here of techniques for *using* these constructs. In other words, the methodology (or process) of conceptual modelling is not examined.

2) ***It serves as a model of reality, which gives insight into the application domain. In other words, the construction of the conceptual model enables the systems analysts to have a better understanding of the application and the users' needs.***

3) ***It serves as a basis upon which the design and implementation of the database can be carried out and, against which the design and implementation can be tested.***

4) ***It serves as a part of the documentation to be used during the maintenance phase to facilitate modification and enhancements of the system.*** (pp. 146-147)

From this characterization, it is clear that conceptual modelling is a pivotal activity in the early stages of systems development, and that problems at this stage will be magnified later in the development process. In short, conceptual modelling supports the development and use of information systems through the construction of an explicit description of the things in an organization (or application), on which an IS design and implementation are based. For this research, the second point mentioned by Kung and Solvberg is of particular interest, as it indicates the role of conceptual modelling in representing reality (as perceived by users).

2.1.3 Range of Conceptual Models

The move from implementation-dependent views of IS to more conceptual views which recognize the role of systems in representing knowledge about reality (Abbott, 1987; Borgida et al., 1985) has been observed in several contexts. Systems analysis and design methodologies have become concerned with developing an abstract description of the entities to be represented in an IS, which is independent of technology and implementation considerations (e.g., Olle et al., 1986; Borgida et al., 1985; Olle et al., 1983; Olle et al., 1982; Nijssen, 1976). In addition, semantic data models stress the use of constructs which are "natural" from the point of view of users (e.g., Hammer and McLeod, 1981; Shipman, 1981). Furthermore, programming languages have moved toward ever higher levels of abstraction (Abbott, 1987; Liskov and Guttag, 1986).

In the remainder of this chapter, a variety of approaches to conceptual modelling are reviewed. These are roughly classified into *systems analysis methodologies* (conceptual models), *semantic data models*, and *object-oriented models*[11]. The review is not exhaustive, as there are dozens, if not hundreds, of such approaches described in the literature (Hull and King, 1987; Bubenko, 1986). However, a number of diverse models are examined in order to illustrate the important constructs and emphases of conceptual modelling approaches. Examples used here are generally those provided by the authors. The chapter concludes by evaluating the commonality among models, and suggests an important reason for dissatisfaction with the current state of conceptual modelling research. Four of the models discussed here are again considered in Chapter 6, where they are compared with the model proposed in Chapter 4.

2.2 CONCEPTUAL MODELS

In the following discussions, a somewhat artificial distinction is drawn between conceptual modelling approaches which are part of a larger development methodology, and those which "stand alone". The latter will be dealt with in subsequent sections on semantic data models and object-based models. First, a number of conceptual models which are part of more comprehensive systems development methodologies are reviewed.

[11]This is simply a classification which parallels three streams of literature. There is no firm distinction between the categories apart from their evolution, with the first emerging to address the weaknesses of earlier systems analysis methods, the second emerging from the database community to address the limitations of traditional data models, and the third emerging from developments in programming languages.

2.2.1 **Taxis/RML/Telos** (Mylopoulos et al., 1980; Greenspan and Mylopoulos, 1984; Mylopoulos et al., 1990; Mylopoulos, 1991)[12]

The Taxis project originated with a database design language called Taxis[13] and a language, called RML (Requirements Modelling Language), for use in the conceptual modelling of an application (Borgida et al., 1985; Greenspan and Mylopoulos, 1984). The project has evolved to produce CML (Conceptual Modelling Language) (Borgida et al., 1988) and, most recently, Telos (Mylopoulos, 1991; Mylopoulos et al., 1990). The entire project covers phases from requirements specification to database design and implementation. The core of all the above languages consists of the use of *tokens* to represent individual entities, *attributes* to describe entities, and several abstraction mechanisms - *classification, generalization/specialization,* and *aggregation* - to model a domain.

The main differences between RML and Telos are that (1) Telos is intended, not only for modelling the subject matter of an information system, but also the system, development, and usage domains[14], (2) Telos offers more uniformity, using the notion of a *proposition* to represent all knowledge about a domain, (3) certain features, such as the kinds of attributes allowed, are fixed (or built-in) in RML, while Telos endeavours to provide a more flexible environment in which features can be added as needed, and (4) Telos offers a more sophisticated model of time, as well as a sublanguage for expressing rules and constraints (Mylopoulos et al., 1990).

Uniformity and simplicity of model constructs are stressed in Telos, with two primitive units - token (representing a concrete or abstract entity) and attribute (representing a binary association between entities) - forming the basis for defining what are called *structured objects* (which describe knowledge

[12]A collection of early papers describing various aspects of the Taxis project is found in (Nixon, 1984).

[13]Hence, Taxis could also be discussed as a semantic data model. However, since the evolution of languages in the project has increasingly emphasized conceptual modelling, a descendant of Taxis called Telos is discussed here as a conceptual model.

[14]Hence, Telos is more general in scope than most conceptual models (Mylopoulos, 1990).

pertaining to an entity). Taxis, RML, and Telos[15] all incorporate facilities to describe the structural and behavioral properties of entities. Structure is expressed using a hierarchy of object classes linked by IS-A associations. Each class is defined in terms of a number of attributes. In addition, the model also views classes as objects. These, in turn, are classified into metaclasses, permitting facts about sets of things to be represented. Behavioral knowledge is modelled in Telos as *activities*[16]. Activities describe changes to entities, may include prerequisites, and are instances of activity classes. As with object classes, activity classes may be organized in a hierarchy.

Another important feature of Telos and its ancestors, from the point of view of providing "natural" representation (Mylopoulos, 1991; Borgida et al., 1988), is the notion of a 1-1 correspondence between objects and entities in the domain being modelled.

Telos extends the Taxis and RML models by including a mechanism for representing and reasoning about temporal knowledge using time *intervals* as a primitive (cf. Allen, 1983).

Four features stand out in Telos and its ancestors. First, there is a strong emphasis on a uniform approach to modelling entities (structurally) and behavior via classes (object and activity, respectively) which are linked via IS-A associations. Thus, classification and specialization are strongly supported abstraction mechanisms. Second, uniformity is extended by considering classes as objects and then classifying them into metaclasses. Telos uses this notion to support infinite levels of classification. Third, there is the notion of a 1-1 correspondence between Telos objects and entities in the domain being modelled. Finally, while Taxis emphasizes data manipulation through transactions in modelling behavior, RML and Telos handle behavior in a less procedural manner using events and activities, which are subject

[15]This review does not consider in detail the differences among Taxis, RML, CML, and Telos. Since the other models evolved from Taxis, there are many similarities. However, where appropriate, the improved modelling capability of successive refinements is mentioned.

[16]There are some terminological differences among models in this regard. Taxis supports procedural description of behavior through *transaction*, RML uses the term *event* to describe non-procedural specification of behavior (and constraints), and Telos uses *activities* to model behavioral knowledge.

to pre- and post-conditions.

2.2.2 NIAM (Nijssen, 1976; Verheijen and Van Bekkum, 1982)

NIAM (Nijssen's Information Analysis Method) is a modelling approach based on the *structure* and *flow* of information in an organization. Knowledge about the structure of things in the domain of interest (referred to in NIAM as the *object system*) is expressed in a binary model[17] which recognizes associations between *lexical* and *non-lexical* objects and object types. Lexical objects may be viewed as surrogates or names (e.g., the string "John"), whereas non-lexical objects are things in the domain or object system (e.g., the person ***John***). Lexical and non-lexical object *types* are classes of lexical and non-lexical objects, respectively (e.g., **Name** and **PERSON**).

Binary associations among objects are of two kinds: *bridge types* and *idea types*. A bridge type is a binary association between a non-lexical and a lexical object type, such as **PERSON Has_name Name**, and is often referred to in other models as an *attribute*. An idea type reflects a binary association between two non-lexical object types, such as **PERSON Employed_by COMPANY**, and is more commonly referred to as a *relationship*. Bridge and idea types are populated by *bridges* and *ideas*, linking members of the respective types.

Additionally, NIAM recognizes a number of constraints on the participation of objects in ideas and bridges. *Identifier constraints* restrict the cardinality of lexical/non-lexical objects (of some type) in a binary association (more commonly referred to as *cardinality* constraints). *Subset constraints* restrict the objects involved in one kind of binary association (e.g., **presenters** of papers) to be a subset of those involved in another type of association (e.g., **authors** of papers). *Disjoint constraints* indicate that the populations of two subtypes are mutually exclusive (e.g. **ACCEPTED_PAPERS** and **REJECTED_PAPERS**). *Total role constraints* indicate that all instances of one type participate in a

[17]In NIAM, an instance of the model is expressed as an *information structure diagram* (ISD).

certain binary association with another type (e.g. **PERSON Child_of PARENT**).

In addition to information structure, NIAM allows the representation of *information flows* and the transformation of information by functions. The mechanism by which this is achieved in NIAM is referred to as *information flow diagrams* (IFDs). These are similar to data flow diagrams (DeMarco, 1979). The intent of this part of the model is to capture information about the changes (processing) which objects undergo in a domain.

To summarize, NIAM is a methodology for expressing knowledge about the structure of things in some domain via binary associations (represented in information structure diagrams). In addition, information flow diagrams are introduced to capture processes (dynamics). However, ISDs and IFDs do not constitute an integrated package, as they use different constructs, and behavior is not part of object type definitions.

2.2.3 ACM/PCM (Brodie and Ridjanovic, 1984; Brodie et al., 1983; Brodie and Silva, 1982)

ACM/PCM (Active and Passive Component Modelling) is a development methodology based on the semantic data model SHM+ (Brodie and Ridjanovic, 1984)[18]. SHM+ permits the modelling of structural properties of things in a domain. It is based on one fundamental construct, the object, and utilizes four abstraction mechanisms[19] - classification, aggregation, generalization, and association - to relate objects.

Classification abstracts the common properties of each of a collection of objects (e.g., ***john*** INSTANCE_OF **PERSON**). An object is an instance of a class if it possesses the properties defining the

[18]This is a good example of the fuzzy distinction between conceptual models and semantic data models. SHM+ could as easily be discussed under the heading "semantic data models". However, it is considered here because it is a part of the larger methodology that is ACM/PCM.

[19]Abstraction mechanisms permit the hiding of certain details to highlight other information. These features constitute an important part of most conceptual and semantic models.

class. *Aggregation* abstracts a collection of objects (parts) into a more complex object (e.g., <Name, Address, Birthdate, ...> = **PERSON**)[20]. *Generalization* abstracts the IS-A relationship between specialized and more general classes (e.g., **CUSTOMER** IS-A **PERSON**). *Association*, which is a less widely used abstraction, captures the "member of" relationship between a set and its elements (e.g., **EMPLOYEE** MEMBER_OF **UNION**). These abstractions and their semantics have been defined formally in SHM+ using predicate-based axioms and functions on sets.

Together, these mechanisms form the basis for representing static (structural) knowledge about things in a domain. In addition, *transaction modelling* is used to specify structural and behavioral properties of transactions and queries. Behavioral properties of applications are supported in SHM+ by primitive operations on objects (including *insert*, *delete*, and *update*), by three kinds of control abstractions (*sequence*, *iteration*, and *choice*) to compose operations, and by two forms of procedural abstraction (*actions* and *transactions*).

As with NIAM, there is an effort in ACM/PCM to capture structural and behavioral knowledge about things, although the behavioral knowledge is closely related to the detailed specific content of programs or procedures (i.e., the how, rather than the what, of change) and structure and behavior are not integrated in class definitions.

2.2.4 JSD (Jackson, 1983; McNeile, 1986)

JSD (Jackson System Development) is a methodology consisting of six steps which span the systems development process from analysis to implementation. The concern here is with the first two steps, which involve the description of the real world, since these steps deal with the conceptual modelling component of the method. These steps are called the *entity action* step and the *entity structure* step.

[20]This is more precisely called *Cartesian aggregation*, since the parts here are attributes instead of objects which, in turn, possess attributes.

In the entity action step, entities and the actions they *perform* and *suffer* are listed. JSD does recognize some abstractions with respect to entities, such as classification in terms of common properties. However, it does not appear to deal with generalization or aggregation. An action is defined as an event which occurs in the real world modelled by the system. Entity actions express the behavior of entities over time.

In the entity structure step, the ordering of actions in time is described[21]. Entity structure actually describes the behavior of entities as events occur through the well-known control mechanisms of sequence, iteration, and choice.

Subsequent steps in JSD relate to the design and implementation of systems. JSD does not deal with the structural and relational properties of things, unlike most other conceptual modelling approaches. In addition, the support for abstraction mechanisms is weak.

2.2.5 OBCM (Takagaki, 1990; Takagaki and Wand, 1991)

OBCM (Ontology-Based Conceptual Model) is an approach to information systems modelling based on the ontology of Mario Bunge (Bunge, 1977; 1979). This work differs from other conceptual models in two important respects. First, OBCM is built on the strong theoretical foundation provided by Bunge's model of the nature of reality. Consequently, all the constructs introduced reflect primitives from the underlying ontology. Second, OBCM is intended to provide a uniform modelling environment that encompasses both conceptual modelling and implementation activities in systems development[22].

The ontological foundation behind OBCM offers a small number of modelling constructs. According to the ontology, the world is composed of *things*. A thing possesses *properties*, which

[21]Note that *structure* as described here bears no relation to the notion of *structural attributes* discussed in relation to other models.

[22]For this reason, OBCM is discussed here, although it could equally well be discussed as a object-oriented model (Section 2.4).

determine its state. States of things are subject to *law*-governed changes. This means that both allowed property values and allowed changes to property values are constrained. Things may be *composed* of other things and possess *emergent* properties. Things may also be grouped into *classes* which share common properties.

OBCM is, in turn, defined in terms of constructs which have ontological equivalents. *Surrogates* represent the existence of things. The notion of a *model object* (or simply model) is used to describe a view of a set of surrogates. A model object defines a class, and includes sets of *state functions*, *law statements*, and *change functions*. State functions represent properties. Law statements limit the values that may be assumed by one, or a combination of two or more, state functions. Change functions define allowed changes to state functions. A surrogate and a model together determine what is called an *object*. An object may thereby be thought of as a *view* of a particular thing as a member of one class (out of many possible classes). Conversely, a surrogate may be regarded as different objects by virtue of different models (or views) of it. For example, a given individual (whose existence is represented by a surrogate) can be considered as a CUSTOMER by modelling a certain set of properties, including **Account_balance**. The same individual can be considered as an EMPLOYEE by modelling a different set of properties, including **Salary**. In other words, the same surrogate can be combined with different models to produce different objects. OBCM also supports the notion of *composition*, by which objects may compose into others (conversely, an object may be decomposed into simpler objects).

Like several other conceptual models, OBCM supports the representation of both structural and behavioral information. The former is achieved through state functions, which map surrogates to values, and laws, which constrain allowed states. The latter is achieved through change functions. Furthermore, the model supports commonly used abstraction mechanisms. Classification is formalized through a pairing of surrogates to models. Specialization is supported via the lattice structure relating classes. Aggregation is realized through associations among objects.

In short, OBCM is a theoretically grounded, formal conceptual model for representing things in a domain. A major strength is its formal relationship to a well-developed theory of the nature of reality. A small set of ontological primitives provides a foundation for defining modelling constructs. Unlike some other conceptual models, OBCM does not introduce ad hoc constructs to model a particular situation. Instead, it uses only mechanisms which are fully justified from the ontology. The constructs used support the representation of both structure and behavior and capture many widely used abstraction mechanisms.

2.2.6 Other Conceptual Models

Many conceptual models have been proposed, as indicated by a series of books (based on IFIP working conferences) devoted to the description, comparison, and evaluation of such approaches (Olle et al., 1986; 1983; 1982). Among these models is CIM (Gustafson et al., 1982; Bubenko, 1980), which is a formal model based on entities, entity types, and events. Entities possess attributes and participate in relationships. This model is noteworthy for introducing a notion of *time* to conceptual modelling. Another formal model which includes time modelling is the set-function (SF) model (Berztiss, 1986).

In addition to these, a number of other modelling approaches have emerged with a database focus. These are generally referred to as *semantic (or conceptual) data models*. The next section reviews several semantic models.

2.3 SEMANTIC DATA MODELS

Paralleling the emerging emphasis on conceptual modelling in the systems analysis literature has been a movement in the database field away from machine-oriented data models and toward problem-oriented data models (Peckham and Maryanski, 1988; Hull and King, 1987). These so-called *semantic data models* have been developed to compensate for the semantic limitations of data structures in conventional data models such as the hierarchic, network, and relational (Brodie, 1984; Kent, 1979; Kent 1978). That is, they are intended to allow database designers to model data in ways that correspond more closely to users' knowledge.

Two noteworthy early models have contributed much to the field of semantic data modelling. The binary data model (Abrial, 1974; Bracchi et al., 1976) is an early attempt to capture additional semantics of an application by modelling relationships among things via binary associations. One advantage of binary models is the inherent simplicity achieved by representing all associations as binary links (Tsichritzis and Lochovsky, 1982). In addition, work by Smith and Smith (1977) highlights the importance of abstraction mechanisms such as aggregation and specialization in capturing application semantics. While these models are not reviewed in detail here, they have influenced many models which were subsequently developed.

A large number of semantic data models have been proposed and extensive reviews are found in Hull and King (1987) and Peckham and Maryanski (1988). In this section, a number of prominent models are briefly described.

2.3.1 Entity-Relationship Model and Extensions (Chen, 1976; Teorey et al., 1986)

The Entity-Relationship (E-R) model (Chen, 1976) is frequently cited as an early and important contribution to capturing application semantics in an implementation-independent manner. The basic constructs of the model are *entities*[23], *relationships*, and *attributes*. An E-R diagram[24] documents the logical structure of a domain by specifying entity types (sets), the attributes possessed by entities of each type, and the binary relationships between entity types. Furthermore, relationships may also possess attributes: however, the model does not distinguish in any way those which do from those which do not.

---

[23]Chen distinguishes between *entity sets* and *entities* (likewise between *relationship sets* and *relationships*). Entity and relationship sets denote classes of entity and relationship instances, respectively. As with most other discussions of the E-R model (e.g., Teorey et al., 1986), the term "set" will be omitted from this review.

[24]In the E-R model, the diagramming technique is the basic representation tool.

As well, the existence of entities of some types may be recognized to be *dependent* on the existence of entities of other types. In addition, the model allows the imposition of *cardinality constraints* to describe the participation of entities of various types in a relationship. One of the significant limitations of the E-R model is that it contains no mechanism to express behavioral properties of entities.

Extensions to the E-R model have been proposed (e.g., Teorey et al., 1986). The primary goal of these extensions has been to add abstraction mechanisms such as generalization/specialization to the model, and to deal with n-ary relationships, as well as mandatory versus optional associations.

The E-R model is widely used as a tool in conceptual database modelling. According to Teorey et al. (1986):

> [t]he entity-relationship model has been most successful as a tool for communication between the designer and the end user during the requirements analysis and conceptual design phases because of its ease of understanding and its convenience in representation. (p. 198)

In addition, techniques exist for converting E-R schemata to schemata of the traditional data models. For example, Teorey et al. discuss the conversion of an extended E-R schema to a normalized relational schema.

2.3.2 **SDM** (Hammer and McLeod, 1981)

The Semantic Data Model (SDM) is, as its name suggests, a data model intended to capture the semantics of data with a set of constructs that are claimed to be "natural" for modelling database applications. One of the stated objectives of SDM is to "serve as a conceptual database model in the database design process" (Hammer and McLeod, 1981, p.351). More specifically:

> Our goal is the design of a higher-level database model that will enable the database designer to naturally and directly incorporate more of the semantics of a database into its schema. Such a semantics-based database description and structuring formalism is intended to serve as a natural application modelling mechanism to capture and express the structure of the application environment in the structure of the database. (p. 352)

Despite this, the authors do not indicate what constitutes "naturalness". They do present a number of modelling constructs but do not justify the choice of these constructs nor evaluate their naturalness.

As with other semantic models, SDM views a database as a collection of *entities* corresponding to things in the application domain. Entities possess *attributes*, and are organized into *classes*, which share common attributes. Classes are related to other classes via *interclass connections*.

The specific treatment of classes, attributes, and interclass connections in SDM is as follows. Classes introduce a classification abstraction. A class consists of a set of entities, which may be concrete objects (e.g., persons), events (e.g., transactions), names, or even classes[25] (e.g., kinds of ships). Classes are characterized in terms of member attributes, which describe each member of a class by relating it to one or more members of the same or other classes. Classes also possess class attributes which describe properties of the class as a whole.

SDM also distinguishes *base* from *non-base* classes. Non-base classes are defined in terms of interclass connections to other classes (e.g., subclasses linked to superclasses by IS-A connections). SDM recognizes two essential kinds of interclass connections, subclass and grouping connections. Subclass connections establish IS-A links between classes. Subclasses may be defined based on common values for an attribute (e.g., Address='Vancouver'), or on other criteria. Grouping connections are used to define metaclasses.

Attributes in SDM are used to describe classes. The instances of a class possess a value for each attribute of the class. A value is either an entity in the database or a collection of such entities. Attributes may be mandatory (null values not allowed) or optional (null values allowed). In addition, an attribute may be specified as the inverse of another attribute.

SDM does not describe entities (or classes) in terms of their behavior, or how attribute values may (or may not) change over time. In addition, it provides a large number of convenient mechanisms (rules) for describing how subclasses may be defined and attributes constrained. However, there is no indication

[25]A class of classes is generally referred to as a *metaclass*.

that these are in any way complete, nor any argument as to why they are natural.

2.3.3 FDM/DAPLEX (Shipman, 1981)

The Functional Data Model (FDM) (and the accompanying data description language DAPLEX) is a semantic data model based on functional representation. The model has two basic constructs, *entity* and *function*. The stated goal of the model is:

> to provide a conceptually natural database language. That is, the DAPLEX constructs used to model real-world situations are intended to closely match the conceptual constructs a human being might employ when thinking about these situations. (Shipman, 1981, p.140)

However, the authors do not indicate how this matching is achieved.

In the language DAPLEX, data are modelled as *entities* which are intended to directly represent entities in the user's reality. *Functions* model relationships among entities (i.e., properties and other associations). Properties may be primitive, or derived from previously defined properties.

Functions are also used to define entity types or classes (as function without arguments). Although there is no explicit mechanism for class specialization, functions can be introduced to model this abstraction. Also supported is function *inversion*, recognizing that properties may be inverses of other properties (e.g., **Child_of** and **Parent_of**). Furthermore, derived functions may be used to define various views of a database, thereby supporting *semantic relativism*. Functions may also be used to enforce several kinds of integrity constraints.

To summarize, FDM/DAPLEX is a semantic model (and data language) based on two fundamental constructs, entity and function. Entities are intended to correspond to real-world objects. Functions play a large role in defining properties, abstractions such as classification and generalization, and user views, as well as in enforcing integrity constraints. FDM does not deal with representing information about the behavior of entities. Finally, FDM has served as the foundation for Iris, an object-oriented data model (see Section 2.4.2).

2.3.4 Other Semantic Data Models

This review has summarized a number of important semantic data models to highlight the nature of the constructs used and the similarities and differences among models. More extensive reviews contain details of these and other models (e.g., Peckham and Maryanski, 1988; Hull and King, 1987). Among the other significant models are the Semantic Association Model (SAM*) (Su, 1983), which introduces a large number of association types (membership, aggregation, interaction, generalization, composition, cross-product, and summarization), and the event model (King and McLeod, 1984), which supports classification, generalization, and aggregation, as well as dynamic modelling via events.

2.4 OBJECT-ORIENTED MODELS

2.4.1 The Object Paradigm

Historically, the technologies available for constructing information systems have not been based on a model that views an IS as a representation of knowledge. Perhaps as a consequence, the plethora of methodologies for creating a set of user requirements for a system, and then translating these requirements into a collection of programs, data files, and guidelines for use, have frequently failed to produce systems which meet users needs and/or expectations.

Recently, however, a new metaphor of computing - the *object paradigm* - has emerged and attracted a lot of attention, as evidenced by the proliferation of literature on the subject. For example, Kim and Lochovsky (1989), Meyrowitz (1989; 1988; 1987; 1986), Schriver and Wegner (1987), and Sigplan (1986) are devoted to object-oriented programming, while papers in Kerschberg (1986) and Kerckhoffs et al. (1986) apply the paradigm to databases and simulation, respectively. This interest is at least in part because of claims that object-oriented constructs are more "natural" than those of previous approaches. Since the objective of this research is to develop an IS model that provides for the direct representation of knowledge and, hence, could reasonably be called "natural", it is useful to determine the extent to which the object model of computing does allow for the direct representation of knowledge.

The essence of the object paradigm is the *encapsulation* of data and the procedures which operate on that data in a single unit called an object (Nierstrasz, 1987; Rentsch, 1982; Robson, 1981). A second fundamental construct is that class objects and instance objects are distinguished, with classes providing a common form for creating instances which are in one-to-one correspondence with entities in some domain. Other constructs are also used, such as *specialization/inheritance, independence, homogeneity, message passing, and composition.* However, there is widespread disagreement as to which of these are necessary to the paradigm (e.g., Nierstrasz, 1989; Wand, 1989; Banerjee et al., 1987; Pascoe, 1986).

The roots of object-oriented computing go back to the simulation programming language SIMULA (Birtwistle et al., 1973). This language introduced both the ideas of encapsulation and of class/instance distinction. Simulation applications dealing with entities serviced by physical facilities, both of which have properties important to the application, are well-suited to modelling via objects. The other primary contribution to object orientation was the development of the Smalltalk programming language. Through various refinements (Smalltalk-72, Smalltalk-74, Smalltalk-80, and more recent versions)[26], additional concepts, such as independence, homogeneity, and message passing, were introduced to provide a uniform programming environment in which everything is represented by objects.

The object-oriented approach has been taken beyond the domain of programming languages and simulation and used as a general modelling tool for databases and knowledge representation systems. Database systems such as IRIS (Fishman et al. 1987; Lyngbaek and Kent, 1986) and ORION (Banerjee et al., 1987) exhibit varying degrees of "object-orientedness". Frame-based knowledge representation systems, such as KEE (Fikes and Kehler, 1985) and Nexpert (Neuron Data, 1988) use concepts that are related to those available in object-oriented programming languages such as Smalltalk. In addition, the

[26]For a discussion of the evolution of Smalltalk, see Goldberg (1984).

constructs have been applied more generally to the analysis and design of information systems (e.g., Coad and Yourdon, 1991; Bailin, 1990; Henderson-Sellers and Edwards, 1990; Wirfs-Brock and Johnson, 1990).

Two uses of object constructs involving conceptual modelling are surveyed here. The first is object-oriented data models, which are closely related to, and extend, earlier semantic data models (King, 1989). The second is the recent explosion in interest in object-oriented analysis and design.

2.4.2 Object-oriented Data Models

In recent years, there have been several research efforts aimed at developing database systems based on object-oriented data models such as ORION (Banerjee et al., 1987), IRIS (Fishman et al., 1987), and ENCORE (Hornick and Zdonik, 1987). In general, these models extend concepts taken from object-oriented programming to consideration of technical issues critical in a database environment. Such issues include persistence of objects, data integrity, data security, data sharing, changes to database schema, and query management.

ORION (Banerjee et al., 1987) is an object-oriented data model supporting encapsulation, classification, inheritance, and message passing, as well as composite objects (i.e., objects that are comprised of other objects as parts and are treated as a unit for storage, retrieval, and integrity purposes), dynamic schema evolution (i.e., changes to class definition and hierarchy), and versions. Some of the additional features are intended specifically to meet the needs of engineering "design databases".

Other features of ORION expand on the object paradigm. For example, the model supports multiple inheritance of properties and behavior, so that classes are arranged in a lattice or directed acyclic graph. A special *set* class is predefined (instances of which are sets) to facilitate predicate-based queries on a database. This class has messages for searching a set, adding elements, and so on.

In addition, the semantic construct PART_OF[27] is captured through the specification of composite

[27]This corresponds to a composition abstraction.

objects. A composite object is one with a hierarchy of exclusive component objects (i.e., no component may belong to more than one composite object). In implementation, clustering of these composite objects (which are often used together) on secondary storage devices reduces retrieval time. Provision is also made for the more general concept of aggregate objects. These are loose collections of objects forming a *cover aggregation*[28]. Aggregate objects can represent many types of relationships among their constituents, as criteria for membership in the aggregate can be specified by the user. Consequently, an object can participate in many aggregate objects.

As with several other object-oriented database systems (e.g., Hornick and Zdonik, 1987), ORION supports multiple *versions* of objects. This is a consequence of the particular class of applications envisioned for the ORION system - engineering design databases.

ORION also supports the notion of a *stored-value variable* (for which all instances of a class take on a specified value). This is analogous to the use of *class variables* in the programming language Smalltalk (Robson and Goldberg, 1981). In addition, the use of *default-value variables* (for which the instances of a class whose value for this variable is not defined take on the specified default value) corresponds closely to the use of default values in frame-based knowledge representation schemes (Bobrow and Winograd, 1977).

The IRIS database system (Fishman et al., 1987; Lyngbaek and Kent, 1986) is the result of another effort to combine an object-oriented data model[29] with the features necessary to form a complete database system (e.g., transaction management, query management, concurrency control). As in other systems, IRIS objects represent entities and concepts from the application domain. These objects are completely

[28]SDM (Hammer and McLeod, 1981) provides the example of a convoy of ships as such an aggregation. In addition, this abstraction mechanism parallels what is called *association* in ACM/PCM (Section 2.2.3).

[29]In this case, the system is based on the functional data model DAPLEX (Shipman, 1981) described earlier (Section 2.3.3).

described by their defined and inherited behaviors and property values, and are classified by type. Many of the features supported by IRIS are similar to other object-oriented database systems such as ORION. These include:

- class definitions in terms of properties and methods,
- type hierarchies and multiple inheritance,
- object independence.

Another database system based on an object-oriented model is the GemStone system (Bretl et al., 1989). GemStone extends the Smalltalk approach to object orientation in the context of programming by providing mechanisms for concurrency control and data integrity in a multi-user environment, and for managing the physical storage of objects. The grounding in Smalltalk means that the system exhibits the object-oriented characteristics of the Smalltalk programming language.

2.4.3 Object-oriented Analysis

Within the last few years, attention has been focused on object-oriented approaches to systems analysis and design (e.g., Booch, 1991; Henderson-Sellers and Edwards, 1990; Bailin, 1989; Pressman, 1987). In particular, object-oriented constructs have been claimed to correspond to the way people organize knowledge and, hence, to be useful for representing knowledge (Coad and Yourdon, 1991). In this section, the main elements of the approach to analysis advocated by Coad and Yourdon is reviewed.

According to Coad and Yourdon, object-oriented analysis (OOA) borrows from both semantic data modelling and object-oriented programming domains. The first step in OOA is identifying objects. The approach allows for distinguishing between things that exist in the domain and a class for describing the common attributes of a collection of things. The justification offered for identifying objects is to "match the technical representation of a system more closely to the conceptual view of the real world" (p. 53). Classes, once identified, are related to other classes by two abstraction mechanisms:

specialization/generalization and whole/part (or composition). Each class is then described in terms of attributes which define the structural properties of instances. Finally, *services* model the behavior of objects in time. Services are described in terms of processing (i.e, the *how*, rather than the *what*).

On the whole, OOA contains mechanisms similar to those of other conceptual models and semantic data models, including the notion of correspondence between entities in the domain and surrogates (objects) in the representation, the abstraction mechanisms of classification, aggregation, specialization, and composition (the last not being adequately dealt with in most other approaches), along with the encapsulation of structural and behavioral knowledge.

The justification for OOA constructs centers around "naturalness". The authors explain their choice of constructs based on "classification theory" as described in the *Encyclopedia Britannica* (see Coad and Yourdon, 1991, p.1). However, there are at least two problems with the justification. First, the classification theory constructs used are not clearly or precisely defined, and the authors do not make use of widespread literature in cognitive science on classification or concept theory[30]. Second, the correspondence between these constructs and those of OOA is not precisely indicated.

2.5 SUMMARY

In most approaches to conceptual modelling (including semantic data models and object-oriented analysis) described above, several themes are evident. First, there is a strong emphasis on a correspondence between model constructs (objects, instances, and so on) and things in the domain of interest. Second, there is widespread use of abstraction mechanisms in the various models. This suggests the importance of abstractions in modelling knowledge about a domain. Although the exact mechanisms

[30]For example, Coad and Yourdon state:

> It would be intellectually satisfying to the authors if we could report that we studied the philosophical ideas behind methods of organization ... . Then, based on the underlying methods human beings use, we could propose the basic constructs essential to an analysis method. But in truth, we cannot say that, nor did we do it. (1991, p.16)

vary from model to model, some conclusions are clear. Classification and (some form of) aggregation are ubiquitous, generalization/specialization is nearly universal, but composition and association are present in only some models. Third, there is a strong tendency to capture both structural and behavioral properties of entities in an application, although the tendency to define classes in terms of both is less common.

Despite this commonality, with the notable exception of OBCM, there has been little effort to rigorously *explain* or justify the constructs chosen in any modelling approach on the basis of naturalness. In particular, many semantic data models have been described as providing a "natural" approach to modelling, without indicating what is meant by natural, or how naturalness is achieved in the model. Such models have been marked by a lack of theory to support constructs used, relying instead on intuitive appeal and convenience. OOA goes one step further by relating the model to classification theory, but provides neither a clear description of the theory nor a precise indication of how OOA directly reflects classification theory. On the other hand, while OBCM does present a rigorously defined set of constructs fully supported by a comprehensive ontology, it avoids issues related to the fact that conceptual modelling ultimately deals with representing *knowledge* of reality. Consequently, it may prove valuable to combine a formal approach, as taken by OBCM, with an explicit focus on theories of the organization of knowledge about entities, as hinted at (but not explored in detail) by OOA.

This thesis is motivated by the belief that a lack of attention to the nature of concepts and classification hinders the development of a "natural" conceptual model. The primary claims to be demonstrated are that a formal conceptual model can be developed based explicitly on a theory of classification, and that such a model provides useful modelling insights. Since a basic assumption underlying the work is that *an information system represents knowledge about things in an organization*, an obvious area to look for a foundation for conceptual models is one which deals with the theory of representing knowledge about things. To this end, Chapter 3 is devoted to a review of literature in cognitive science/psychology dealing with theories of concepts and classification.

CHAPTER 3

CONCEPT THEORIES AND MODELLING IMPLICATIONS

3.1 CONCEPTS AS A BASIS FOR INFORMATION SYSTEMS MODELS

3.1.1 Introduction

As stated earlier, a basic assumption of this thesis is that IS development can benefit by directly adopting the view that *an information system represents **knowledge** about things in an organization.* In order to develop a precise model of IS which embodies this assumption, a grounding in theories of knowledge about things is required. In this chapter, three theories of *concepts*, which deal with the structure and organization of knowledge about things, are briefly reviewed. One of these theories is then selected as a foundation for the model of knowledge that is developed in Chapter 4.

3.1.2 Reference Disciplines

Cognition is studied in many fields of research. Three fields which are most relevant and which have provided much insight are psychology, artificial intelligence (AI) and philosophy[31]. Work in these areas sometimes converges in a hybrid field referred to as *cognitive science* (e.g., Winograd and Flores, 1987; Haugeland, 1981).

While one goal of research in these fields is essentially the same (i.e., a better understanding of cognition), the research methodologies used differ considerably. Cognitive psychology is not just a study of cognition. It is also a methodology for conducting research, since the experimental method is the means by which research in cognitive psychology is carried out. Artificial intelligence focuses instead on the

[31]Additionally, considerable work in linguistics is relevant to cognition. Such work is covered here only to the extent that it overlaps with philosophical and psychological investigations (e.g., Lakoff, 1987).

performance of machines. Insight into cognition is gained by constructing computer programs that behave in some way as a human would under specified conditions. Finally, philosophical investigations of cognition rely on logical arguments that provide explanations consistent with commonly observable (as opposed to experimentally induced) human behavior.

Despite these methodological differences, there is considerable agreement between the theories of cognition advanced in each field. Consequently, while the theories described here are taken primarily from psychological literature, they are consistent with work done in AI, philosophy, and linguistics.

3.1.3 Cognition

For the purposes of this research, *cognition* is defined as:

> the acquisition and use of knowledge by humans, including "all processes by which the sensory input is transformed, reduced, elaborated, stored, recovered and used." (Best, 1986, p. 4)[32]

Cognitive activity is extremely varied and complex. Humans engage in such diverse activities as perceiving, understanding perceptions by relating new experiences to existing knowledge, locating and using relevant knowledge in diverse situations, making decisions, reorganizing knowledge based on new experiences, making plans or establishing goals, seeking means to achieve goals, and engaging in activity to accomplish these goals (e.g., Best, 1986; Medin and Smith, 1984; Simon, 1979; Crutchfield, 1973; Newell and Simon, 1972).

One component of cognition critical to survival is our *knowledge of things in the real world.* Every person is exposed to new things daily, and must learn to represent these things and classify them in terms of previous experiences in order to survive. The importance of classification or categorization to human life is widely recognized by cognitive scientists. As Lakoff (1987) states:

[32]The quote is actually offered by Best as a definition of cognitive psychology. However, in accordance with section 3.1.2, cognitive psychology is here viewed more narrowly as imposing a particular *methodology* - laboratory experimentation - for investigating the phenomena described above.

> Without the ability to categorize, we could not function at all, either in the physical world or in our social and intellectual lives. An understanding of how we categorize is central to any understanding of how we think and how we function .... (p. 6)

Similarly, Smith and Medin (1981) open their well-known book on concepts and categorization with the statement:

> Without concepts, mental life would be chaotic. If we perceived each entity as unique, we would be overwhelmed by the sheer diversity of what we experience and unable to remember more than a minute fraction of what we encounter. (p.1)

There appear to be two primary functions of concepts (Smith, 1978). The first is to support what has been called *cognitive economy* (Rosch, 1978). By grouping many instances into a single category, these instances are identified as being the same in certain respects. Usually, this can be taken to mean that the instances *share some set of properties*, referred to as a concept. This permits reasonable questions to be asked with respect to known instances of the concept. For example, if the concept **EMPLOYEE** has as one its defining properties **Department**, meaning that each employee works in one department, then it makes sense to ask "which department does *x* work in?", for any *x* who has been previously classified as an employee. Similarly, given a new individual *y*, if we are told that *y* works in a department, we may be able to classify *y* as an **EMPLOYEE**[33].

The second important role of concepts emerges from the previous statement. If an individual is classified in a particular way on the basis of *incomplete information* about its properties (i.e., on the basis of knowing only a subset of its properties), one can *infer* additional properties of interest that the individual must possess. That is, possessing a subset of properties may imply that an individual possesses another subset, all of which are defining properties of the concept. For example, people learn from experiences with fires that they generate heat. Consequently, on observing a distant fire (by the presence of flames), an individual "knows" that it will be hot, even if the heat is not felt at that distance. Of course,

[33]This holds only if **EMPLOYEE**, along with specializations of it such as **SUPERVISOR**, are the only concepts which have the property **Department**. Otherwise, knowledge of additional properties may be needed to classify an instance.

misclassification sometimes occurs. When this happens, we may question our perception of the thing, refine our conceptualization to either define the associated category more precisely, or develop a new concept based on the experience.

3.1.4 Concepts and World Knowledge

Every theory of knowledge about reality makes (sometimes implicit) assumptions about the nature of what is represented: that is, assumptions about reality. The branch of philosophy that deals with the nature of reality is *ontology*. Most research in cognitive science presupposes an *objectivist* ontology (see Lakoff, 1987; cf. Newell and Simon, 1976). That is, the world is assumed to consist of concrete things or entities[34] that are associated in various ways.

It is not the intent of this thesis to deal explicitly with the nature of reality. Instead, the focus is on providing a formal description of cognitive representation that constitutes the basis for a conceptual IS model. However, it is worth explicitly mentioning (without detailed comment) a few ontological assumptions that underlie this research, and the implicit relationship between reality and its representation. These assumptions are given without comment. A formal treatment of these and many more aspects of the nature of reality can be found in, for example, the ontology of Bunge (1977;1979).

The fundamental building block of reality is the *thing*. Things are described by characteristics (or *properties*). Collections of physical or abstract things may share some characteristics. A *concept* is, for this research, defined as *a mental abstraction of the common characteristics of a collection of things*. Things, in turn, are represented mentally by what are called *instances*. An important premise of this work is that *concepts and instances together, and their organization, make up our mental representation of the real world*. That is, they constitute our knowledge of things.

[34]Often, the word *object* is used synonymously with thing or entity. However, in this proposal, the word object has a specialized meaning, introduced in Chapter 2, and which is refined and formalized in Chapter 4.

3.1.5 Components of Conceptual Representation

To represent a thing adequately, its various properties must be accounted for. For this research, four components of knowledge are identified[35].

First, perception of an individual thing leads to representation of its **existence**. Second, there are various **structural** properties that characterize a thing (Smith, 1978; Bruner et al., 1956). Third, a thing has physical or abstract *relationships* to other things (Smith, 1978). A **relational** property describes an association among two or more things that holds for some period of time. This association may be characterized by its own structural and relational properties, and by constraints on the behavior of the participating entities. Some relationships are permanent, while others are transitory. Finally, there are rules governing the *behavior* of things in various contexts (Smith, 1978). **Behavioral** properties determine the allowable changes in the values of other properties. Each property of a thing belongs to exactly one of these categories.

This view of the components of knowledge is similar to the characterization of objectivism given by Lakoff (1987):

> objectivism holds that reality is structured in a way that can be modeled by set-theoretical models;
> that is, the world consists of
> entities [existence]
> the properties of entities [structure/behavior]
> the relations holding among those entities. (p. 159)

[35]These components are not always explicitly recognized in the literature. However, examples of each and their role in categorization can be found, for example, in *Categories and Concepts* (Smith and Medin, 1981). Chapter 4 formally defines each of these elements.

3.1.6 IS Models and Conceptual Representation

Knowledge of things in the world may be characterized in terms of high volume and complexity. This often makes it difficult to accurately recall large amounts of data, or to recall data about a specific thing in a reasonable period of time. Further, knowledge tends to decay over time if not frequently used. For these reasons, people have devised many aids to relieve the cognitive burden of "knowing the world". One of the tools that has proved most useful and reliable in this respect is the computer.

The data and programs that inhabit a computer can be viewed as representations of knowledge about some aspects of the real world (Abbott, 1987; Brooks, 1987). Data capture static knowledge about things, whereas programs may describe their behavior. According to Abbott's "principle of knowledge abstraction":

> It is the domain-level knowledge embodied by a program that represents the program's only real connection to the problem domain. (1987, p. 667)

In organizations, the knowledge contained in an IS relates to the elements of reality relevant to the functioning of that organization. Typically, these include employees, products, customers, suppliers, contracts, and so on. However, since we know the world only through our cognitive representation of it (concepts and instances), an IS is ultimately a representation of a conceptual system as is any intermediate product of IS development (e.g., an Entity-Relationship diagram). Consequently, the following working hypothesis motivates the current research.

Working Hypothesis:

Methodologies for IS development can benefit from an IS model based explicitly and formally on a theory of concepts.

3.2 THEORIES OF CONCEPTS

A theory of concepts generally specifies several things. Foremost, the structure or internal composition of cognitive abstractions, as well as the relationship between concepts and things perceived, must be explained (Medin and Smith, 1984). In addition, mechanisms for determining whether an instance belongs to a category are needed (Bruner et al., 1956). Furthermore, a comprehensive theory should describe how concepts are related to other concepts in memory (Smith, 1978; Collins and Loftus, 1975; Quillian, 1968), reflecting how things are linked in reality. Finally, a theory of concepts should offer an explanation of how concepts are acquired[36].

Theories of concepts that have been proposed in psychology, AI, and philosophy are in many ways similar. They differ primarily in the manner of accumulating evidence to support or refute their predictions. In particular, the three psychological theories examined here are each surrounded by a variety of *experimental* evidence (see Smith and Medin, 1981 for a summary). For each theory, there is evidence which is consistent with the theory's predictions, as well as evidence that contradicts its predictions.

3.2.1 Classical View

3.2.1.1 Theory

The oldest and most precise psychological theory of concepts has been with humans in one form or another since the ancient Greeks. In its modern incarnation, the classical view was first detailed and studied by Bruner, Goodnow, and Austin (1956)[37]. This theory holds that *a concept can be stated in*

[36]The review of concept theories in this chapter does not discuss the position of each theory with regard to concept acquisition.

[37]It should be noted that the studies reported by Bruner et al. were primarily concerned with strategies for *acquiring* concepts. However, the nature of the concepts used forms a useful basis for describing the classical view.

terms of individually necessary and jointly sufficient conditions for membership of a thing in a category. That is, the conditions define the concept.

To illustrate this, suppose that **A** designates the set of all known instances (each representing a thing) sharing properties $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$. Then a new instance is classified as belonging to the concept associated with **A** *if and only if* it has properties $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$. The concept, then, is the three properties. In symbols:

$C = \{P_1, P_2, P_3\}$, and

$A = \{a|P_1(a),P_2(a),P_3(a)\}$,

where **C** designates a concept and **a** designates an instance of **A**. For example, a classical definition of a triangle is:

P_1: closed figure

P_2: bounded by three line segments.

Obviously, other properties characterize a triangle (e.g., sum of the angles is 180). These lead to alternate definitions. However, the other properties of a triangle can be derived from the above definition. From this, a looser view of necessary and sufficient conditions can be stated, in which there may be several subsets of properties, each sufficient to classify an instance[38]. This supports classification on the basis of incomplete information, and enables one to infer the presence of other (possibly non-observed) properties as discussed in Section 3.1.3.

A tacit assumption of the classical view is that all concepts can be defined this way. Furthermore, concepts can be organized hierarchically by the consideration of *proper subset* relations among sets of properties (Keil, 1979). For example, the set of equilateral triangles is a subset of the set of all triangles, and possesses properties *in addition to* every property possessed by all triangles (e.g., line segments are of equal length).

[38]The model developed in Chapter 4 adopts this relaxed view.

Classical concepts can also be linked by *part/whole* associations (see, e.g., Miller and Johnson-Laird, 1976). In some contexts, a concept is viewed as a collection of parts or *components*. For example, in an automobile manufacturing context, a car may be best thought of as a collection of parts, including a motor, chassis, and frame. The configuration of these components, however, yields a *composite* concept with *emergent* properties. For example, considered holistically, a car has properties, such as **fuel consumption**, that are not properties of any of the components.

3.2.1.2 Extensions

Bruner et al. (1956) do not fully describe what may constitute a property or attribute that may be used to define a concept. They do, however, state that an attribute is:

> any discriminable feature of an event that is susceptible of some discriminable variation from event to event[39]. (p.26)

It is clear from the experiments they conduct that *structural* properties may be used in the definition of a concept, since the concepts they use differ on clearly discernible properties of visual structure. For example, a structural property of a triangle is that it is a closed figure.

In addition, *behavioral* and *relational* properties can be used to define a concept (Glass and Holyoak, 1986), as expected from the characterization of components of knowledge about things, stated earlier. For example, the concept of **STUDENT** can be defined as a specialization of the concept of **PERSON**. The specialization involves defining students as those instances of person that are involved in a particular relationship with the instances of another concept called **SCHOOL**.

[39]It is clear from the context that, by "event", is meant a perception of a thing or entity. For example, they define a "criterial variable" as "a discriminable feature ... used as a means of inferring the identity of *something*." (p. 26, italics mine)

3.2.1.3 Applicability

In the experiments described in Bruner et al., the concepts used are artificial, based on sets of geometric figures that share certain physical properties. In other empirical work on the classical view, the concepts used tend to be quite concrete (e.g., Armstrong et al., 1983; Bourne et al., 1976).

The use of classical concepts, in the form of technical definitions, is most obvious in domains such as mathematics and the physical sciences. In these fields, precise definitions are needed for theory development, experimentation, and measurement. Furthermore, the criticality of technical (formal) definitions to a field such as mathematics demonstrates that concepts need not contain instances that represent perceptible things in order to be subject to precise definitions in terms of necessary and sufficient conditions.

3.2.1.4 Limitations

The limitations of the classical view can be subdivided into two categories. First, the nature of the concepts to which technical definitions can be applied appears limited to those representing either physical things or abstract things which are not subjectively interpreted by the individual (Lakoff, 1987; Armstrong et al., 1983). Thus, concepts such as honesty and fairness are not definable under the classical view (Glass and Holyoak, 1986). Interestingly, these are concepts that would not generally be regarded as containing identifiable *things*.

The second set of limitations appears more substantive. Considerable experimental work has been done in testing the classical theory. As a result, a number of well-known criticisms have emerged (Smith and Medin, 1981). For one thing, the theory does not account for so-called *typicality effects* (Armstrong et al., 1983; Roth and Shoben, 1983; Rosch and Mervis, 1975). That is, in experiments, some instances or specialized concepts are described as more typical of the concept than others (e.g., **equilateral** versus

obtuse for **triangle**)[40]. Clearly, though, this need not be a problem with the ability to define concepts unambiguously. Instead, typicality may be an empirical phenomenon relating to the relative frequency of experiences with the various exemplars of a category. The level of exposure to, or familiarity with, certain exemplars (facts) may produce such an effect (cf. Tversky and Kahneman, 1984).

In addition, Roth and Shoben (1983) have shown that the degree to which exemplars are considered typical of a category depends on the context in which information is presented, indicating that typicality is situation specific. Furthermore, in most of the arguments that have proposed this as a weakness of the theory, the tests that have been used in fact compare concepts to concepts (e.g., Medin and Smith, 1984; Rosch and Mervis, 1975). Thus, typicality, as the term has been used, often implies a measure of the association between two concepts (as in the case of equilateral vs. obtuse triangles), and not between an instance and a concept. Consequently, researchers should be comparing, for example, instances of equilateral triangles and those of obtuse triangles in an attempt to assess typicality.

Another criticism of the classical view is that people have very limited success in accurately stating necessary and sufficient conditions for membership in most categories (Medin and Smith, 1984; Posner and Keele, 1968; Wittgenstein, 1958). This has been used to support the proposition that people do not use such conditions to classify. However, this situation may be akin to one of recognition versus recall. As most exam situations demonstrate, human recognition ability exceeds recall ability. Thus, information that is represented cognitively need not be immediately articulable. Similarly, a failure to state defining conditions does not demonstrate that subjects do not use them in making classifications. A less direct, but equally valid, test of whether people use technical definitions to classify is to measure their success in classifying instances of well-defined concepts. Evidence ranging from laboratory experiments (Bruner et al., 1956) to the survival and adaptation of the species suggests that people tend to be very

[40]Note that **equilateral** and **obtuse** are also concepts that are special cases of the more general concept **triangle**. This refinement of concepts makes possible their arrangement into hierarchies (see Section 3.2.4).

good at correctly classifying things.

3.2.1.5 Summary

Empirical work on the classical theory of concepts shows that while the theory does not account for the observed behavior of people in all classification and concept definition tasks, there is support for the theory as a description of our abstraction of the common elements of concrete and/or technically definable things in the world.

Even advocates of other theories of concepts concede that the classical view is appropriate in *well-defined domains* (Lakoff, 1987; Medin and Smith, 1984; McCloskey and Glucksberg, 1978). For example:

> the typicality rating, verification latency, and truth judgment data do not exclude the possibility that categories have clear boundaries. (McCloskey and Glucksberg, 1978, p. 462)
>
> [W]hile [people] may appear to have vague and fuzzy notions about the criteria for category membership, they could, in principle, learn more about such concepts and then be perfectly consistent in their judgments. (p. 466)
>
> A technical taxonomy would be an example of such an idealized category system. (p. 466)
>
> [Under the classical view], a conceptual category is defined in terms of necessary and sufficient conditions shared by all members. Such conditions include properties of entities and relations holding among entities. (Lakoff, 1987, p. 166) ... Such metaphysical assumptions about physical objects certainly won't get us into trouble when we are dealing with tables and other familiar physical objects. (p. 175)

3.2.2 Prototype View

3.2.2.1 Theory

The prototype theory of concepts and concept formation emerged largely as a response to the problems identified in experiments on the classical theory (Smith, 1988; Medin and Smith, 1984; Smith and Medin, 1981). In this view, concepts are defined by *prototypes*, or statistical abstractions, in which the value of each feature or property is computed as the average of the value of that feature for the known instances of the category (Rosch and Mervis, 1975; Rosch, 1973; Reed, 1972; Posner and Keele, 1968)[41]. Thus, properties are no longer defining, but are probabilistic. All properties need not be shared by all members of a category.

One interesting consequence of this theory is that concepts change as new instances are classified but, nevertheless, become more stable as the set of instances grows (Glass and Holyoak, 1986). Thus, the theory addresses the phenomenon of concepts from a developmental perspective (Rosch, 1973).

Membership of an instance in a category is determined by comparing a candidate instance against the prototype, and accepting if the candidate does not differ "significantly" from the prototype. The concept or prototype is then updated by incorporating the new instance into the average of features.

3.2.2.2 Applicability

As a developmental theory, the prototype view accounts for the types of errors that can be observed among children during the acquisition of concepts and the modification of a concept that occurs after a misclassification (Rosch, 1973). If a new instance differs on some important dimension from the existing instances, the prototype must be modified to accommodate this difference.

That the theory was developed specifically in response to the empirical problems of the classical view is evident from its ability to handle typicality effects. The prototype itself is regarded as the most typical instance (Medin and Smith, 1984). Less typical instances are ordered by their "distance" from the prototype (Rosch, 1973). Further, since the concepts used in tests of the theory are often not defined by necessary and sufficient conditions (e.g., Posner and Keele, 1968), it is not surprising that people cannot elucidate such conditions in experimental situations where statistically abstracted concepts are used.

As with the classical theory, the nature of the concepts that may be defined with a prototype

[41]There are several variations on prototype theory. These are encompassed in the citations given here. The details of these differences are outside the scope of this research and, hence, are not discussed.

theory seems to be those that abstract from physical things.

3.2.2.3 Limitations

The most serious problem with this theory is that most proposed versions of it do not specify how properties are averaged[42]. This is especially relevant when a property is qualitative rather than quantitative, and gives rise to the attachment of probabilities to the presence and/or values of various properties in a given instance (Medin and Smith, 1984). Unfortunately, procedures for calculating these probabilities are not clear. In addition, the notion of "significant" differences between a prototype and a candidate instance is generally ill-defined in prototype theories.

Additionally, the theory is unable to account for the fact that people are capable of elucidating necessary and sufficient conditions in many situations. In particular, sufficient conditions cannot be derived from a prototype. Also, the theory does not appear to account for abstract definable concepts such as those found in mathematics.

A final criticism of prototype theory as a superior alternative to the classical view (although not strictly a limitation) is that the apparent use of typicality information to classify instances (identification procedure) is not inconsistent with the existence of defining properties (core). In fact, experimental subjects often believe that defining properties exist, even if they cannot state them (cf. Armstrong et al., 1983).

[42]An exception is given in Osherson and Smith (1981). These authors give a formalization of prototype theory and use fuzzy set theory to demonstrate that the formalization yields contradictions. They also point out that the classical view, and the associated use of standard set theory, is immune to such contradictions.

3.2.3 Exemplar View

3.2.3.1 Theory

Both the classical and prototype views regard a concept as an abstraction that is distinct from the instances which satisfy its definition. In contrast, classification can be treated as a process whereby candidate instances are compared to stored instances. So-called *exemplar* theories do not regard a concept as an explicit abstraction. Instead, the concept is implicit in the instances of a category (Medin and Smith, 1984).

Exemplar theories hold that candidate instances are evaluated by comparing them to a collection of existing instances that constitute a category, rather than to an abstraction of these instances.

The basis for this position comes from the Wittgensteinian notion of *family resemblances* (Wittgenstein, 1958; see also Lakoff, 1987). That is, many things can be classified under a common label even though any given pair of things in a collection have little, if anything, in common. However, each instance shares different properties with different instances of the same category[43].

3.2.3.2 Applicability

As with the prototype view, the problems incurred by people in specifying necessary and sufficient conditions is consistent with the general theory. That is, since there is no abstraction, it is not expected that people would be able to state defining properties for concepts. In addition, the typicality effects reported by Rosch and her colleagues (e.g., Rosch, 1973; Rosch and Mervis, 1975) can be explained in terms of a high degree of similarity of an instance to other instances along several dimensions.

[43]Wittgenstein uses the common concept of *game* as an example. The diversity of activities that may be called games seems to demonstrate that there are no properties that all games share, and that it is nonsensical to postulate a prototypical game (Wittgenstein, 1958, pp. 31-34).

3.2.3.3 Limitations

The limitations of exemplar theories of concepts are similar to those of prototype theories. The theories are generally ill-specified (e.g., see Glass and Holyoak, 1986; Medin and Smith, 1984). The analogy of family resemblances gives only an intuitive understanding of the relationships among instances, and does not appear to be easily formalizable. As with prototype theory, there is also the issue of defining similarity and establishing cutoffs for the boundaries of a category. Finally, the theory does not account for the ability (and necessity) to rigorously define concepts in scientific and mathematical domains.

3.2.4 Summary

The conflicting evidence on the relative merits of these different theories of concepts has led some authors to suggest that each theory has certain strengths and is appropriate for certain situations. In particular, Smith and Medin (1981) and Lakoff (1987) argue that different theories produce verifiable predictions in different situations and, consequently, each may be appropriate in certain circumstances. This factor must be considered in selecting a theory for formalizing a model of knowledge about the organizational domains for which most IS maintain knowledge (see Section 3.3.3).

3.2.5 Semantic Organization of Concepts

A theory of concepts alone is inadequate to account for the creation and use of knowledge of the physical world. An important cognitive activity involves recognizing various associations among concepts. These relationships give further meaning to a concept by linking it to other concepts in certain well-defined ways.

The branch of research that studies the nature of the organization of knowledge is called *semantic memory* research, and has been pursued widely in both cognitive psychology and AI (e.g., Smith, 1978; Collins and Loftus, 1975; Minsky, 1975; Schank and Abelson, 1971; Quillian, 1968). Of the many theories proposed, some, such as the *hierarchical-network* model (Quillian, 1968) and the *predicate-intersections* model (Meyer, 1970), adopt a classical theory of concepts. Others, such as the *feature comparison* model (Smith, Shoben and Rips, 1974), appear to be based on a prototype theory of concepts. The latter are not examined here.

3.2.5.1 Major Principles[44]

A theory of semantic memory endeavours to explain how we understand concepts in the larger context of how they relate to each other. In fact, in some theories (e.g., Quillian, 1968), concepts are to be understood entirely in terms of links to other concepts. Consequently, the major contribution of any such theory is a set of constructs describing the nature of relationships[45]. These constructs should lead to predictions about performance on tasks that require using relationships among concepts (Smith, 1978).

One of the linkages proposed in the hierarchical-network theory of semantic organization is that concepts are linked *hierarchically*. Thus, many concepts are specializations of others. The hierarchy ranges from concepts of high generality (Keil, 1979), such as **PHYSICAL OBJECT**, to ones of high specificity (Quillian, 1968), such as **KITCHEN TABLE**. Further, properties of higher level concepts are inherited by all descendant concepts (Smith, 1978; Quillian, 1968). These connections are referred to in most theories as IS-A links and lower level concepts may be called *specializations*. Concepts which are specialized possess properties in addition to those of the concepts from which they descend (Smith, 1978).

A second major conceptual link is that of *composition*. Many concepts are composites of simpler

[44]These principles are evident from an examination of the literature, but are not always identified as such, particularly in some of the early theories. In fact, the principles are dealt with in other areas of computer science, such as database theory, under the label of *abstraction mechanisms* (Smith and Smith, 1977). As noted in Chapter 2, it is interesting that work in this area is often referred to as *conceptual modelling* (Brodie et al., 1984).

[45]Network theories of semantic memory typically represent concepts by nodes and links (relationships) by arcs connecting nodes.

concepts, each of which has properties and can be regarded, for some purposes, as independent of the composite. In theories of semantic memory, these links are usually referred to as PART-OF (or HAS) links (Smith, 1978). Thus, a **CAR** can be viewed as composed of a number of other things, including **MOTOR**, **CHASSIS**, and **FRAME**.

A composite is not, however, a simple collection of other concepts. It has *emergent* properties that are not present in any subset of components. For example, **fuel consumption** is a property of CAR that is not a property of any of its components.

The third major link is essentially a catch-all covering other types of associations. These relationships may or may not be required or permanent. That is, for any instance of a concept, they may or may not hold, and may or may not change over time (e.g., marriage, employment, and ownership relationships do change over time). Temporal links do not appear to be present in some theories of semantic memory (Quillian, 1968; cf. Smith, 1978, p. 14).

3.2.5.2 Some Findings

Much of the experimental work on semantic memory has focused on measuring the "closeness" of concepts in terms of the number of links of a particular type (usually hierarchical) that separate two concepts (Smith, 1978; Collins and Loftus, 1975; Quillian, 1968). The general prediction of these theories is that the closer two concepts are in meaning, the less time should be required by subjects to answer questions that involve the relationship between the concepts involved[46].

The findings of this work have been mixed. While some support exists for the organization of conceptual memory as described above (see Collins and Loftus, 1975 for a review), other work has yielded conflicting results. In particular, response times can be shorter for pairs of items that are postulated to be

[46]For example, "Is a dog a mammal?" would be predicted to be answered more quickly than "Is a dog an animal?", since there are presumably more intervening levels of concepts in the latter case.

connected by more (transitive) links than other pairs (see Smith, 1978). This suggests at least that knowledge can be retrieved in other than a strict hierarchical manner.

3.2.5.3 Summary

Hierarchical-network theories of semantic memory seem to be consistent with a classical theory of concepts. In fact, the classical theory can be used to describe how concepts can be defined as specializations, compositions, or in terms of temporal relationships among other concepts. In particular, specializations can be defined by the addition of structural, relational, and behavioral properties to concepts; compositions can be defined in terms of existing concepts (parts), along with *emergent properties* that do not belong to any of the parts; and concepts determined by temporal relationships can be defined by relational properties. This is dealt with formally in Chapter 4.

3.3 ON THE CONCEPTUAL REPRESENTATION OF ORGANIZATIONS

An information system exists to maintain information for use by members of an organization. The scope of information and its use ranges from accounting data kept to satisfy legal requirements to management reports and decision aids which support decisions critical to the survival of the organization. An IS supports human cognitive activity by reducing both memory and processing demands on individuals in an organization with respect to the entities about which information is maintained. Systems and methodologies, then, need to be able to represent static knowledge about the organization, along with information to describe the ways in which this static knowledge can be changed (e.g., Kung and Solvberg, 1986; Brodie, 1984; Verheijen and Van Bekkum, 1982).

3.3.1 Organization Entities

An organization can be viewed as composed of entities belonging to a number of classes. These classes are subclasses of very high level classes which describe an organization's products and the factors that facilitate production (physical resources, human resources, financial resources), as well as groups of entities in the environment that affect the organization's goals and performance (e.g., suppliers, customers, financial institutions, governments, etc.). The knowledge about such things that an IS needs to contain can be broken into the components of knowledge identified earlier: existential, structural, relational, and behavioral.

3.3.1.1 Structural Properties

Much of the knowledge of the types of entities described above is factual in nature, and describes static elements of instances of the kinds of concepts identified earlier. Employees are represented in terms of properties such as address, birth date and salary. Products are described by such qualities as physical attributes (e.g., size, weight), cost, and location. Other entities can be similarly described.

3.3.1.2 Relational Properties

Some information may not describe any single class of entities, but rather, possible relationships between instances of two or more classes of things. Such relationships include contracts of various sorts - including employment, loans, and agreements to supply certain quantities of products under certain circumstances - as well as relationships that describe physical or logical configurations of entities, such as the fact that a warehouse contains a certain quantity of a certain product at a particular time.

3.3.1.3 Behavioral Properties

The behavior of organizational entities that is of interest relates to the manner in which the values of structural and relational properties may change. In traditional IS, this behavior is implicit in the programs that process data. It is not always explicitly recognized that these procedures represent behaviors of things in the organization that have to be accounted for by the IS (e.g., Abbott, 1987; Kent, 1979).

3.3.2 Goals of Representation

An organization may conveniently be thought of in terms of the entities or things that comprise it. This is essentially an "accounting" point of view. For this research, an organization is viewed as containing a collection of things which operate together to achieve certain goals.

Within this context, an IS represents (some of) the things of interest to individuals in the organization. The development of an IS involves transforming knowledge to a form that can be represented and processed by a computer. As a first step in this process, a vocabulary for describing knowledge (i.e., a conceptual model) is needed. This model should reflect human concepts of things in the organization, as well as individual things that are instances of these concepts. In general, the concepts of interest relate to the aforementioned resources or assets of the organization, the external entities which affect the decisions and performance of the organization, and relationships among various entities (e.g., contracts).

The purpose of a conceptual model, then, should be to provide constructs to represent all four knowledge components which describe instances of concepts representing things in the organization, as well as to represent concepts and their organization.

3.3.3 Suitability of the Classical View

The primary things of interest in a representation of an organization are either concrete or describe abstract relationships among concrete things (Mattessich, 1989). Further, for legal, accounting, and practical purposes, it is most often the case that these things are clearly and unambiguously defined. It was pointed out earlier that the classical theory of concepts is the best available for providing precise technical definitions of concepts. Consequently, the classical view is chosen here as best-suited for providing a theory of representation on which IS models can be based.

It may be posited that concepts are inherently subjective and, hence, a general theory of conceptual representation does not account for the fact that different people may have different concepts to represent their "world". Two arguments reduce the validity of this claim.

First, the provision of legal and/or accounting and/or technical definitions of organizational entities increases the likelihood that different members of an organization will have the same conceptualizations of these entities. If they do not, inconsistencies will be reflected in the behavior of these individuals and should not be difficult to identify.

Second, there is a body of psychological research which shows that there is a nearly universal set of ontological concepts for describing reality (Keil, 1986; 1979)[47]. This ontological hierarchy describes a number of very high-level concepts, such as physical things, solids, liquids, animals, events, and so on, from which all concepts that serve us in day to day life can be derived as specializations. Hence, there is good reason to believe that concepts which can be defined by necessary and sufficient conditions will be shared across members of an organization (although the specific properties of interest may vary among organizational members). This supports an implicit assumption in information systems development that users share a largely common view of the things in an organization (Bubenko, 1986), although different users may be interested in different roles of the same things and, therefore, in different subsets of properties.

[47]At least, this seems to be the case within a given society. A discussion of differences among societies is contained in *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind* (Lakoff, 1987).

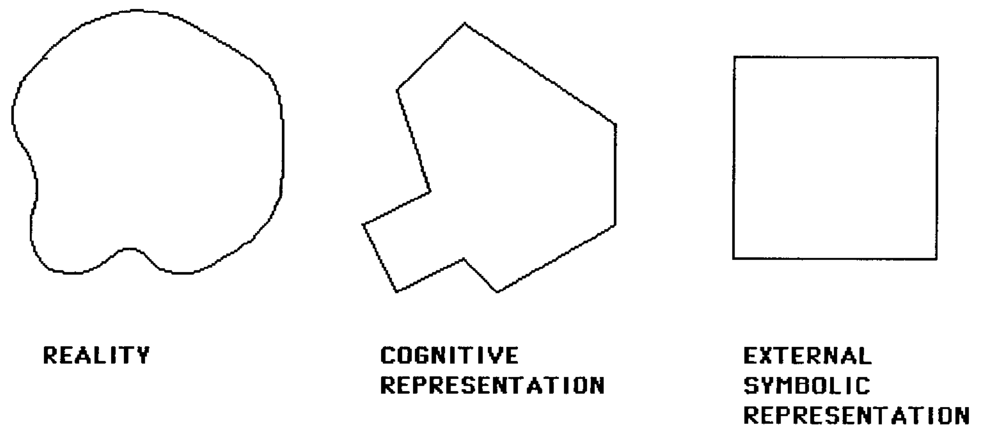
CHAPTER 4

A CONCEPTUAL INFORMATION SYSTEMS MODEL

4.1 INTRODUCTION

The previous chapters have dealt with three levels or domains of analysis (Figure 4-1). At the level of *reality*, there are things which possess characteristics. The level of *cognitive representation* (or knowledge) contains constructs which represent reality. Finally, the level of *information systems representation*[48] contains constructs which represent knowledge. This chapter deals with the second and third of these levels. ***The objective is to present a formal conceptual information systems model for representing knowledge about things, which conforms to the classical theory of concepts.*** In other words, a "vocabulary" is defined (at the third level) on the basis of a set of constructs at the second level.

Figure 4-1: Reality, Knowledge, and Information Systems



[48]More generally, this third level can be referred to as a level of *external symbolic representation.* This recognizes that the representation of knowledge need not be in a machine, but may equally well exist in another medium, such as paper, in which some form of symbols or surrogates represent elements of knowledge. Furthermore, this level may contain several sub-levels, such as a written description and an implemented system.

In order to construct this model, it is first necessary to have a precise view of the constructs of classical concept theory, which were introduced only informally in the previous chapter. Consequently, the next section reviews the major elements of the classical view and presents formal definitions of each. Section 4.3 then discusses the importance of *directness* in developing a conceptual IS model, and operationalizes directness in terms of a 1-1 correspondence between constructs at the cognitive and information system levels. Section 4.4 defines the model MIMIC (Morphological[49] Information systems Model of Instances and Concepts). In the concluding section, the model is shown to support a necessary condition for creating good representations. As this chapter introduces considerable notation, Appendix 2 summarizes the important symbols used, organized in the order in which they are introduced here.

4.2 A CLASSICAL MODEL OF CONCEPTS

4.2.1 Basic Constructs of Classical Concept Theory

To briefly review and set the stage for the remainder of this section, the main elements of classical concept theory, as introduced in Chapter 3, are:

1. ***Instance*** - represents the existence of an individual thing in the world
2. ***Property*** - describes an instance in terms of structure, relationships, or allowed behavior
3. ***Concept*** - abstracts the common properties of a set of instances
4. ***Specialization*** - establishes a concept as a refinement of another
5. ***Composition*** - recognizes a concept as consisting of simpler parts and possessing emergent properties.

As this research focuses on formalizing these basic constructs, two important simplifications

[49]***Morphological*** means pertaining to *structure* or *form*. Accordingly, the proposed model deals with representing the structure or form of knowledge about things.

influence what follows. First, issues involving the potential uncertainty of knowledge are not pursued. For example, a person may know only that the value of a property (e.g., weight) is in a certain range (e.g., 150 to 170 lbs) or that the value is a guess (e.g., 165 lbs). Database researchers will recognize this as a form of *closed world assumption* (Reiter, 1984). This assumption is deliberately adopted in order to focus on the task of formalizing the basic cognitive structures. Further research would be appropriate to extend the formalism to deal with various forms of uncertainty.

The second simplification is that the model considers only the representation of knowledge about some subject matter. In general, humans also have higher level knowledge about what is known about a subject matter. For example, a person may know the average value of some attribute, such as account balance, for the known instances of a class. Such *metaknowledge* (knowledge about knowledge) is not directly captured by the formalism. However, if knowledge is indeed the subject matter of interest, the definitions which follow do indirectly offer a framework for expressing knowledge about knowledge.

The constructs listed above are described in the cognitive science literature, but are not generally clearly defined. However, in order to have a formal conceptual model that corresponds to classical concept theory, precise definitions of each are needed. The remainder of this section is devoted to providing these definitions. The constructs are divided into two types. The notions of *instance*, along with *structural* and *relational* properties are **primitive** in that they are not derived from other constructs. After defining these fundamental constructs, the **derived** constructs of *behavioral property*, *class*, *specialization*, and *composition* are considered.

4.2.2 Primitive Constructs

Recall from Section 3.1.4 that reality is assumed to contain *things*. The basic constructs of classical concept theory suggest that people represent the *existence* of individual things and have knowledge of them in terms of structural facts and relationships with other things.

4.2.2.1 Instance

The most basic construct of interest is the ***instance***. An instance is a *surrogate* or symbol which designates the existence of a thing. Thus, every thing "known" in some domain is known via an instance, and there is a 1-1 correspondence (or a bijective function) between any set of instances and a set of things. The following definitions formalize notions of *identification function* (1-1 correspondence) and *surrogate*, and use these to define the *instance* construct.

Definition 1:

Let **T** and **W** denote sets of equal size, and **f**:T→W denote a bijective function. **f** will be referred to as an ***identification function of W in T***.

This means that **f** describes a correspondence between elements of **T** and **W**.

Definition 2:

Let **f**:T→W denote an identification function of **W** in **T**. An element t∈T is a ***surrogate*** of w∈W **iff** f(t)=w.

If **t** is a surrogate of **w**, it may stand in the place of (represent) **w** in suitable situations. For example, student numbers are surrogates of students in many administrative contexts in a university.

Definition 3:

An ***instance*** is a cognitive surrogate which designates a thing in the real world.

In other words, given some **T**, **W**, and a bijective function **f**:T→W, an element t∈T is an instance if and only if **t** is a symbol at the cognitive level[50] and **w** (i.e., f(t)) is a thing. Similarly, **T** is a set of instances if and only if **W** is a set of things. For example, **W** may be a set of students and **T** a set of names, each of which uniquely identifies a student.

Postulate 1:

For any finite set of things, **W**, there exists a set of instances, **T**, and a bijective function $f{:}T \rightarrow W$.

This assumption states that a one-to-one correspondence can be established between a finite set of things *of interest* in some domain, and instances in a cognitive representation of that domain. A useful way of viewing the nature of instances as surrogates is that they represent the *existence* of things. However, any things which might "actually exist" in the domain, but which have not been observed, are not "of interest" and, therefore, are not included in **W**.

4.2.2.2 Properties

As defined above, an instance tells us nothing about the referent thing except that it exists. In other words, an instance simply *names* its referent. In most cases, however, people have knowledge of specific characteristics of a thing that may distinguish it from other things. In this regard, *property* is viewed as the second fundamental construct in cognitive representations.

Three kinds of properties will be defined: ***structural***, ***relational***, and ***behavioral***. These correspond to the components of knowledge described in Section 3.1.5. A *structural* property describes an independent aspect of all elements of a set of things which is stable over some time interval. A *relational* property describes a relationship, or linkage, between elements of one or more sets of things which is stable over

[50]Ultimately, this means that **t** has some physical manifestation in the brain. However, the links between the neurophysiology of the brain and cognition are well outside the scope of this research.

some time interval. A *behavioral* property is derived from structural and relational properties, and describes a rule for the allowable changes in the values of the other kinds of properties. Structural and relational properties are fundamentally different from behavioral properties in that they can be used to describe the *state* of instances. Behavioral properties, on the other hand, describe the *allowed changes of state* that instances may undergo. Formal definitions of structural and relational property are given next. The notion of behavioral property is defined later.

4.2.2.2.1 Structural Property

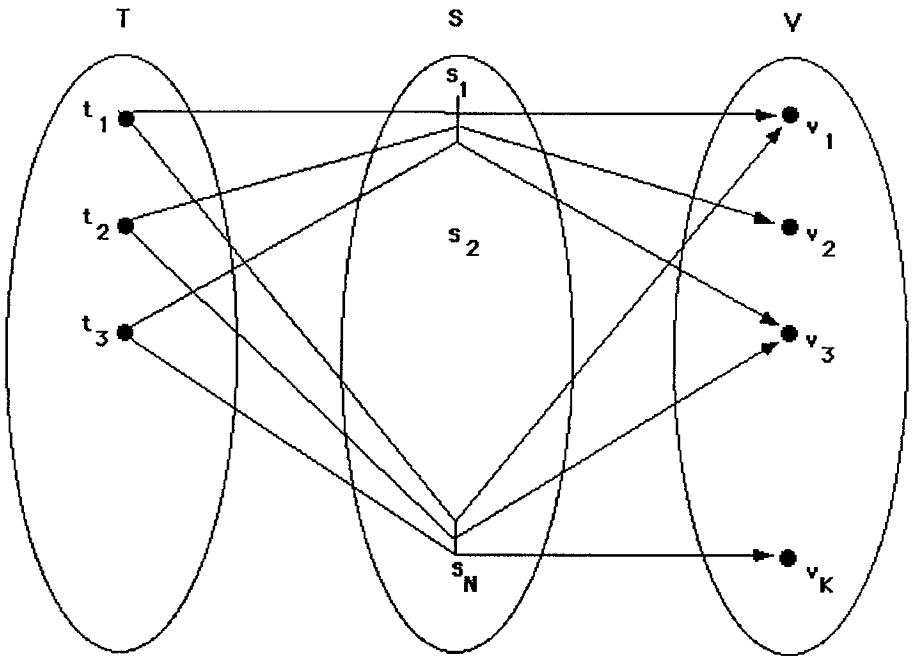
Structural properties describe independent qualities that assist in describing similar things. Independence means that a structural property does not describe any association of things. For example, knowledge about people includes knowledge about **height**, **weight**, **hair color**, and so on. Every human has a *value* associated with each of these properties. Values are distinguished from instances in that the latter possess properties while the former do not. Examples of values include units of measurement such as kilograms and meters. The values of some properties may be permanent, while others may change. In addition, properties generally describe or apply to more than one instance, as evidenced by those listed above. Consequently, a structural property, **S**, of a set of instances, **T**, will be modeled as a finite set of functions, each of which maps from **T** to a set of values, **V**[51]. The reasons for modelling a property as a set of functions are discussed later (Sections 4.2.2.3 and 5.7).

[51]All sets of instances, values and functions dealt with here are assumed to be finite.

Definition 4:

Let T be a set of instances, V a set of values, and S = $\{s_i | s_i : T \rightarrow V\}$ a set of functions. S will be referred to as a ***(simple) structural property*** *of T* (Figure 4-2)[52].

Figure 4-2: Simple Structural Property



Example:

Weight (S) is a structural property of humans (T) since (in principle) a set of functions can be constructed, whose domain is the set of instances representing humans[53] and codomain is the set of

[52]This may be extended to the case of *composite* (or emergent) structural properties. A composite (or emergent) structural property is a set S = $\{s_i | s_i : T^1 \otimes ... \otimes T^K \rightarrow V\}$. Emergent properties are considered further in Section 4.2.3.4.

[53]For simplicity, a set of instances representing a set of things, **W**, will hereafter be referred to as the set **W**. For example, the "set of instances representing humans" will be referred to as the "set of humans". Unless otherwise specified, such statements describe the cognitive representation level.

weight values. **Weight** describes the weight of every human, with each function in the set describing a potential mapping. That is, for any instance $t \in T$, at a given time (i.e., for some s_i indicating an assignment of weights to people), $s_i(t) \in \{x | x \text{ is a weight value}\}$.

<u>Example</u>:

Birthdate (S) is a structural property of humans (T) since a function s can be constructed whose domain is the set of humans and whose codomain is the set of dates. That is, $\exists\ s{:}T \rightarrow V$ such that $s(t) \in \{x | x \text{ is a date}\}\ \forall\ t \in T$. This means that Birthdate = {s}.

The depiction of a structural property in Figure 4-2 can be equivalently described using a set of tables, since each element of the set S in Figure 4-2 is a function. For example:

s_1

t_1	v_1
t_2	v_2
t_3	v_3
. .	. .

s_2

t_1	v_1
t_2	v_4
t_3	v_6
. .	. .

The property S, then, is a set $\{s_1,...,s_N\}$ of such tables. As the functions have a common domain, it will be convenient to speak of the members of the domain as *possessing* the property.

<u>*Definition 5*</u>:

Let $S{:}T \rightarrow V$ be a structural property, and **A** be any subset of **T**. **A** (and each of its elements) will be said to ***possess*** S.

Example:

The set of adults (A) is a subset of the set of humans (T). Therefore, adults possess the properties weight and birthdate.

In addition, A may constitute the intersection of the domains of several structural properties. Such a set will be called a *category*.

Definition 6:

Let $S_A = \{S^1,...,S^K\}$ denote a set of structural properties having domains $T^1,...,T^K$, respectively. The set $A = T^1 \cap ... \cap T^K$ will be called a ***category** on $S^1,...,S^K$* (or simply a *category*).

Example:

The set of humans is a category determined by the intersection of a set of properties which includes weight and birthdate.

Definition 7:

The function that describes the mapping for property S at some time δ will be referred to as the ***active function*** of S (at δ).

Example:

In the property illustrated in Figure 4-2, one function (e.g., s_2) describes the correct mapping at a given time. Using a specific example, there is a single function which accurately describes the assignment of weight to people. As weights change over time, the function which accurately describes these assignments changes.

There is an important distinction between the properties **weight** and **birthdate** described in the above examples. In the case of **weight, S** may contain many elements (functions), while for **birthdate,** S contains exactly one element. This distinguishes properties that are temporal (changeable) from those that are atemporal (unchangeable). In the former case, the active function may change over time, while in the latter it cannot. This distinction is represented by whether a property set contains more than one, or just a single, function (respectively), thereby capturing the different semantics of these cases. In fact, the treatment of time in this and subsequent definitions is implicit only (see Sections 4.2.2.3 and 5.7). That is, while the value of a property may change (as in the weight example), time is not a parameter of property functions[54]. The fact that values of a property may change is modelled by the presence of several elements (functions) in the property set, only one of which is active. Change, and the passage of time, is understood by the fact that different functions from a set containing more than one element may describe the instance. This is a simple way of distinguishing the permanent or temporal nature of property values for a set of instances.

4.2.2.2.2 Relational Property (binary, n-ary, optional, required)

Some knowledge of instances can only be described with reference to other instances. A relational property is an *association* of instances which describes an association of things. For example, common relationships such as **employment** and **ownership** constitute knowledge of things beyond structural properties. The former describes an association between a person and an organization. The latter describes an association between a person (or organization) and other things.

A *binary* relational property, **R**, of a set of instances, **T**, is defined as a set of functions, each of which maps from **T** to **Q**, a subset of the power set of another set of instances, T^1 (i.e., $Q \subseteq \wp(T^1)$). The

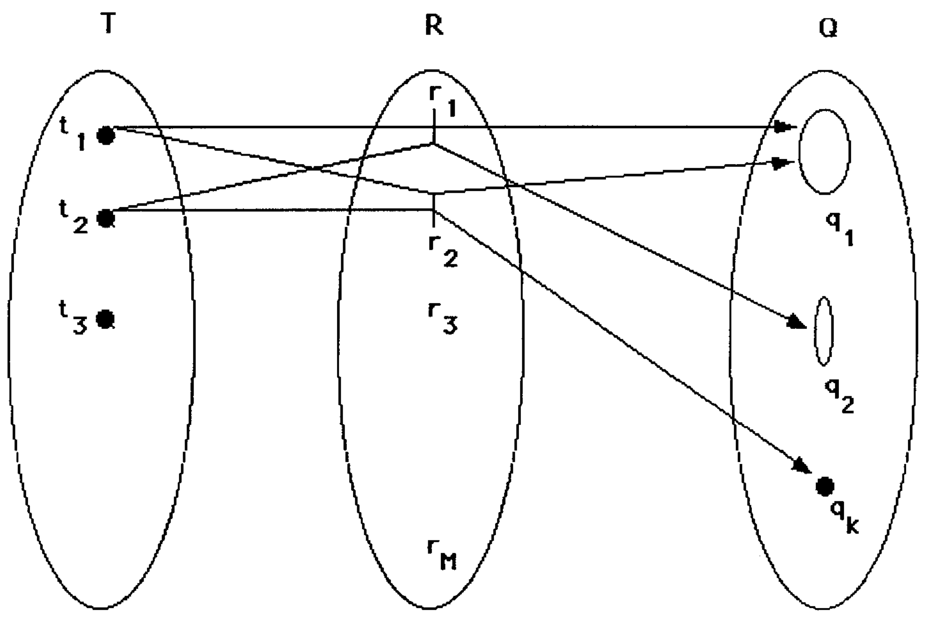
[54]An alternative treatment would be to model a property as a single function of both instances and time. That is, $S:T \otimes \Delta \rightarrow V$, where Δ denotes a set of time points.

codomain is specified as a subset of the power set to indicate that an instance in **T** may be linked with a fixed or varying number of instances of **T¹**. For example, a relational property "has parents" links each person with *two* other persons (with additional restrictions) and, therefore, the codomain of the property is restricted to a subset of the power set of persons which contains only sets with two elements.

Definition 8:

Let T, T^1 denote sets of instances, $Q \subseteq \wp(T^1)$, and **R** = $\{r_j | r_j : T \rightarrow Q\}$ denote a set of functions from T to Q. **R** will be referred to as a ***(simple) binary relational property*** *of T* (see Figure 4-3)[55].

Figure 4-3: Binary Relational Property



[55]As with structural properties, the definition may be extended to *composite* (or emergent) relational properties. A composite (or emergent) binary relational property is a set R = $\{r_j | r_j : T^1 \otimes ... \otimes T^K \rightarrow Q\}$. Emergent properties are considered further in Section 4.2.3.4.

Example:

$T = \{t | t \text{ is a person}\}$,

$T^1 = \{t^1 | t^1 \text{ is a company}\}$,

R = Employed_by.

Assuming a world in which there is no moonlighting, the codomain of this property is actually a subset of T^1 (which, of course, is a subset of $\wp(T^1)$).

As with a structural property, the depiction of a relational property in Figure 4-3 can be equivalently described using a set of tables, since each element of the set **R** in Figure 4-3 is a function. For example:

r_1

t_1	q_1
t_2	q_2
t_3	q_3
⋮	⋮

r_2

t_1	q_1
t_2	q_3
t_3	q_7
⋮	⋮

The property **R**, then, is a set $\{r_1,...,r_N\}$ of such tables.

As the functions have a common domain, it will be convenient to speak of the members of the domain as *possessing* the property. Specifically, any set of instances A (as well as each of its elements) will be said to ***possess*** relational property **R**:T→Q iff A⊆T (cf. Definition 5, page 64). In other words, A possesses **R** when A is a subset of the domain of **R**. In addition, A may constitute the intersection of the domains, $T^1,...,T^K$, of relational properties $R^1,...,R^K$: that is, $A = T^1 \cap ... \cap T^N$. In that case, A will be referred to as a ***category** on $R^1,...,R^K$* (cf. Definition 6, page 65). The set of relational properties possessed by A will be denoted by $\mathbf{R}_A = \{R^1,...,R^K\}$.

A binary relational property, **R**, is more completely described as a *binary relational property of instances in T with instances in T^1*. For notational convenience, this may be written **T R T^1** or R(T,T^1). For a particular instance $t \in T$ at a particular time, the relationship is designated **t r$_j$ q$_j$**, where $q_j \in Q \subseteq \wp(T^1)$, and $j \in \{1,\ldots,|R|\}$.

Since relational properties link instances from one set with instances from another set, one can think of "inverse" properties which describe mappings from elements of the second set to elements of the first. For example, if **Employed_by** links persons with companies, there should be a relational property **Employs**, which links companies with persons. Furthermore, if any person **t$_i$** is employed by company **t^1_i** (i.e., $t^1_i \in$ Employed_by(t_i)), then $t_i \in$ Employs(t^1_i). This is stated more formally as a postulate.

Postulate 2:

If T R T^1, then $\exists$ **R′**, a relational property of instances in **T^1** with instances in **T** (T^1 R′ T), such that for any $t \in T$, $t^1 \in T^1$, $t^1 \in R(t) \Rightarrow t \in R'(t^1)$.[56]

Each element of the binary relational property **R** is a function, **r$_j$**, whose domain is **T**, and codomain is **Q**, a subset of $\wp$(**T^1**). Now, **r$_j$(t)** describes the relationship holding at some time between an element of T and a subset of **T^1** ($q_j \in Q$). Several cases can be distinguished:

1) **q$_j$** is restricted to a single element (including the null element).

   This is the situation of the previous example where an instance **t** is associated with, at most, a single element of **T^1** at any given time. Relationships such as **Married_to** fit this description.

2) **q$_j$** may contain varying numbers of elements, depending on the active function of **R**.

   This is the case where an instance **t** may be associated with several elements of **T^1** at any given

[56] R(t) is a shorthand meaning "for every $r_j \in R$, $r_j(t)$", which is appropriate since every function in a property set has the same domain.

time, and includes relationships such as **Parent_of** and **Supplies.**

3) **R** contains a single element.

This means that the relationship is unchanging. The **Child_of** relationship satisfies this condition.

4) **R** may contain several elements.

In this case, various elements of **R** may hold over time. **Supplier** and **Employed-by** relationships satisfy this condition.

Note that the links commonly described by the term ***IS-A*** are not considered here as relational properties. Despite this, these associations are very important for cognitive representation (see Section 5.4). However, the IS-A link describes a connection between sets of instances, rather than between instances, such that all elements of one set also belong to another (more general) set. The distinction is best made by considering the following examples.

Married(PERSON, PERSON) is a relational property. Selecting an element of the property (e.g., a function $\mathbf{m}_j$) and an instance of the domain (e.g., a person) yields meaningful descriptions such as m_j(John) = Jane. On the other hand, IS-A(EMPLOYEE, PERSON) cannot be instantiated in the same meaningful way. The only correct instantiations are of the type IS-A_j(Jane) = Jane, which does not contain any information. Instead, IS-A links simply imply that all properties possessed by one set of instances are also possessed (inherited) by the other.

The definition of relational property provides the basis for distinguishing between *required* and *optional* relationships. The difference is that, for required properties, *every* element of **T** must be in a relationship with some non-null element(s) of $\mathbf{T}^1$ (hence, the exclusion of the null set from the codomain of functions in **R**).

Definition 9:

Let **T**, **T**^{1} be sets of instances, $\mathbf{Q}\subseteq\wp(\mathbf{T}^1)$, and $\mathbf{R} = \{r_j | r_j : T \rightarrow Q\}$ a set of functions from **T** to **Q**. **R** is referred to as a ***required relational property*** of **T** iff $\emptyset \notin Q$, or as an ***optional relational property*** iff $\emptyset \in Q$.

Example (required):

T = {x|x is a person},
T^1 = T,
R = Child_of.

In this example, there are a number of implicit constraints on the child relation which can be used to demonstrate how certain knowledge can be captured by the formalism. For example, for the single function **c** of the unchanging property **Child_of**, $y \in c(x) \Rightarrow x \notin c(y)$, where **x** and **y** denote person instances. That is, if **x** is **y**'s child, then **y** cannot be **x**'s child. Another restriction is that $x \notin c(x)$. That is, a person cannot be one's own child. Such restrictions are implicitly captured by the nature of the assignments made by the function **c**.

Example (optional):

T = {x|x is a person},
T^1 = T,
R = Married_to.

Thus far, only binary relationships have been considered. However, many associations may involve instances from several sets. The definition of relational property is, therefore, extended to handle arbitrary *n-ary* relationships.

Definition 10:

Let $T, T^1, \ldots, T^{n-1}$ be sets of instances, $Q \subseteq \wp(T^1 \otimes \ldots \otimes T^{n-1})$, and $\mathbf{R} = \{r_j | r_j : T \rightarrow Q\}$ a set of functions from **T** to **Q**. **R** is referred to as an ***n-ary relational property*** of **T**.

Example:

$T = \{x | x \text{ is a supplier}\}$,
$T^1 = \{x | x \text{ is a project}\}$,
$T^2 = \{x | x \text{ is a part}\}$,
$R = \text{Supplies} = \{r_j | r_j : T \rightarrow Q\}$, where $Q \subseteq \wp(T^1 \otimes T^2)$.

4.2.2.3 State, Change, and Time

An important part of our knowledge about things includes knowledge about how they *may change over time*. This knowledge will be formalized as behavioral properties, which constrain allowed changes. In order to formalize the notion of behavioral property, several ancillary definitions are first provided. Change is defined in terms of *events*, where an event is a change in *state*. A state is defined as a collection of active functions of structural and relational properties. In addition, this section offers some preliminary comments on the treatment of time.

4.2.2.3.1 State

At a given moment, a person's knowledge about a domain is in a certain state that reflects a potential state of known things in that domain. The usual way of describing state is in terms of the values taken on by a set of variables at a specific time. However, using the definitions of structural and relational properties given earlier, state can be defined in terms of the elements of the property sets that are active at any time. These elements (functions) in turn determine values when applied to specific instances.

An important assumption of concept theory is that knowledge of things is unavoidably linked to the categories to which they belong. In other words, it only makes sense to talk of the state of a thing *as an instance of a category*. For example, it is meaningless to speak of the state of an instance *Herbie*, without classifying *Herbie* as a member of PERSON, PARROT, AUTOMOBILE, or some other category. The act of categorizing an instance identifies a set of properties it possesses, thereby providing a basis for describing its state with respect to those properties. Furthermore, *every instance of a given category will have a state with respect to the set of properties shared by that category*. Defining the state of a category is an explicit reminder of this. Consequently, two complementary notions of state are presented. The state of an entire *category* of instances is introduced as a mechanism for giving meaning to a *instance* level view of state.

<u>*Definition 11*</u>:

Let T denote a category[57] of instances, and $\mathbf{P_T}=\{P^1,...,P^K\}$ denote the set of structural and relational properties shared by **T**. The ***state*** of **T**, designated σ(**T**), is a vector containing the active function of each of the structural and relational properties possessed by T <u>applied to the instances in T</u>:[58]

$$\sigma(T) = \langle p^k \rangle_{k\in\{1,...,K\}} = \langle p^1_{i1},...,p^K_{iK} \rangle^{59},$$

where $p^k_{ik} \in P^k$, and the index $i_k \in \{1,...,|P^k|\}$.

At a certain time, the state of an instance $t \in T$ is:

[57]*Category* is defined in Definition 6 (page 65). The set of properties that intensionally defines a category may have a special status and, in such cases, will be referred to as a *concept* (see Section 4.2.3.2).

[58]Each property in $\mathbf{S_T}$ and $\mathbf{R_T}$ has a domain which is a superset of T. Consequently, in considering the state of **T** (and in this notation), only the application of the active function of each of the properties possessed by **T** to the instances in **T** is of interest.

[59]The index **ik** should be read $\mathbf{i_k}$.

$$\sigma(t) = <p^1_{i1}(t),\ldots,p^K_{iK}(t)>.$$

At a later time, the state of **t** may be:

$$\sigma'(t) = <p^1_{i'1}(t),\ldots,p^K_{i'K}(t)>, \text{ where } i'_k \neq i_k \text{ for at least one } k \in \{1,\ldots,K\}.$$

Consequently, $\sigma(\mathbf{T})$ may also be written:

$$\begin{aligned}\sigma(T) &= <\sigma(t)>_{t \in T} \\ &= <\sigma(t_1),\ldots,\sigma(t_{|T|})>.\end{aligned}$$

That is, the state of **T** is the joint state of the elements (instances) of **T**.

The following simple example illustrates the independence of structural and relational properties, as well as the meaning of the indexing in the above definition. This example will be used to illustrate various points through the remainder of the chapter.

Example:

Suppose GLASS is a category containing two instances: GLASS = {glass1, glass2}. The instances of GLASS share two structural properties, S^1 = ORIENT and S^2 = CONTENT, with codomains {upright, upside_down} and {empty, full}, respectively. GLASS also has one binary relational property, R^1 = ON, with a set of tables, TABLES = {table1, table2}. The potential combinations of values of the properties of GLASS are:

ORIENT:	t	$s^1_1(t)$	$s^1_2(t)$	$s^1_3(t)$	$s^1_4(t)$
	glass1	upright	upright	upside_down	upside_down
	glass2	upright	upside_down	upright	upside_down

CONTENT:	t	$s^2_1(t)$	$s^2_2(t)$	$s^2_3(t)$	$s^2_4(t)$
	glass1	empty	empty	full	full
	glass2	empty	full	empty	full

ON	t	$r^1_1(t)$	$r^1_2(t)$	$r^1_3(t)$	$r^1_4(t)$
	glass1	table1	table1	table2	table2
	glass2	table1	table2	table1	table2

The potential states of, say, **glass1** are:

$<s^1_1(\text{glass1}),s^2_1(\text{glass1}),r^1_1(\text{glass1})> = <\text{upright,empty,table1}>$ (1)
$<s^1_1(\text{glass1}),s^2_1(\text{glass1}),r^1_3(\text{glass1})> = <\text{upright,empty,table2}>$ (2)
$<s^1_1(\text{glass1}),s^2_3(\text{glass1}),r^1_1(\text{glass1})> = <\text{upright,full,table1}>$ (3)
$<s^1_1(\text{glass1}),s^2_3(\text{glass1}),r^1_3(\text{glass1})> = <\text{upright,full,table2}>$ (4)
$<s^1_3(\text{glass1}),s^2_1(\text{glass1}),r^1_1(\text{glass1})> = <\text{upside\_down,empty,table1}>$ (5)
$<s^1_3(\text{glass1}),s^2_1(\text{glass1}),r^1_3(\text{glass1})> = <\text{upside\_down,empty,table2}>$ (6)
$<s^1_3(\text{glass1}),s^2_3(\text{glass1}),r^1_1(\text{glass1})> = <\text{upside\_down,full,table1}>$ (7)
$<s^1_3(\text{glass1}),s^2_3(\text{glass1}),r^1_3(\text{glass1})> = <\text{upside\_down,full,table2}>$. (8)

In this example, $i_1, i_2, j_1 \in \{1,2,3,4\}$. However, it will not generally be the case that two properties contain the same number of functions.

The potential states of **glass2** can be similarly enumerated. Furthermore, by considering **glass1** and

glass2 together, the potential joint states of the set GLASS can be listed[60].

This example demonstrates the general independence of the states of instances in a category. The fact that there are four functions in each of the two properties of GLASS (instead of two) indicates that the value taken on by **glass1** (*upright* or *upside_down*) is independent of the value taken on by **glass2**. However, by modelling properties as functions in this way, constraints on the joint states of instances *with respect to a single property* can be imposed[61]. For instance, it may be that both glasses are required to be in the same orientation. This semantic constraint can be implicitly imposed by omitting both functions s^1_2 and s^1_3 from the property $\mathbf{S^1}$.

The notion of state gives rise to two useful definitions of *state space* which together highlight the role of the functions in a property set in restricting the values which may occur for properties. That is, if particular instances have certain values for a property, other instances may be constrained to have certain other values. For instance, the salaries of employees in a department may be constrained so that no more than five employees may have a salary over $50,000. This eliminates certain potential functions from the property set.

[60]In terms of the states of the category, joint states should be considered. However, even in this simple case, there are $2^3 \bullet 2^3$, or 64 potential joint states. For manageability, the example considers only the potential state space of a single instance. In addition, the index numbering here does not completely coincide with the consideration of the joint state of instances, since in the latter case, the independence of states of different instances in a set needs to be considered. Hence, s^1_1 could be replaced by s^1_2, s^1_3 could be replaced by s^1_4, and so on, in describing the state of **glass1**.

[61]Constraints with respect to several structural or relational properties are also important, and are considered later in the discussion of behavioral properties.

Definition 12:

The ***potential (or conceivable) state space***, **Σ(t)**, of every instance **t** in a category **T** is the Cartesian product of the **codomains** of the structural and relational properties of **T**. That is,

$$\Sigma(t) = D^{P1} \otimes \ldots \otimes D^{PK},$$

where the **D**'s denote the codomains of the respective structural and relational properties of **T**.

Example:

In the "glass/table" domain, the potential state space of both **glass1** and **glass2** is given by:

{upright, upside_down}⊗{empty, full}⊗{table1, table2}.

Every instance in a category has the same potential state space with respect to the attributes on which that category is based. In fact, this is a derived view of what it means to categorize instances based on similarity (i.e., derived from the notion of classifying based on common properties).

Definition 12 suggests that, while the *conceivable* functions in a property collectively map each element of the domain to each element of the codomain, the *actual* functions in a property will generally be a subset of this.

Definition 13:

The ***possible state space***, **Σ(T)**, of a category, **T**, is the set of vectors formed by taking the Cartesian product of the structural and relational properties of **T**. That is:

$$\Sigma(T) = S^1 \otimes \ldots \otimes S^N \otimes R^1 \otimes \ldots \otimes R^M.$$

It will always be the case that the state of a set of instances is an element of the possible state space (i.e., $\sigma(T) \in \Sigma(T)$).

Example:

Not all the potential states of a thing or things in a category may be possible. Suppose there is a rule governing the states of glasses, stating that both glasses cannot be simultaneously **upside_down**. This rule implies that the potential element of ORIENT, $\mathbf{s^1_4}$ (where s^1_4(glass1)=s^1_4(glass2)=*upside_down*), is not an actual function in the property. Thus, whereas the potential state space of GLASS includes states where both glasses are upside-down, the possible state space does not.

In addition to these restrictions, there may be restrictions on the jointly active functions of several properties. For example, a glass may not be both *upside_down* and *full.* This is modeled implicitly by *behavioral properties* (Section 4.2.3.1), which restrict changes in active functions.

4.2.2.3.2 Change

Two important kinds of change to the state of a cognitive representation are proposed. First, the notion of state gives rise naturally to the view of an *event* as a state change of one or more instances in a category. An event is described by two states - before and after - and involves a change in the active function of one or more properties. The second kind of change involves modifying the *domain* of properties. This notion of change is dealt with first.

Our knowledge of the world changes as we "become aware" (i.e., perceive the existence) of new things. This is captured in this formulation by the creation of new instances to represent the perceived existence of these things. In addition, knowledge of the characteristics of these entities is represented by the properties which the new instances possess. In order for an instance to possess properties (and, therefore, to have values for properties), it must be an element of the domain of these properties. Consequently, the notion of *domain expansion* is defined to accommodate the representation of knowledge about new entities.

Definition 14:

Let **T** designate a category of instances, **A** a nonempty set of instances such that $A \cap T = \emptyset$, $\mathbf{T'} = T \cup A$, and $\mathbf{P} = \{p_i | p_i : T \rightarrow V\}$ a structural or relational property of **T**. A ***domain expansion*** is a change in the domain of **P** such that $P = \{p_i | p_i : T' \rightarrow V\}$.

The creation of a new instance will be accompanied by a domain expansion for each of the structural and relational properties which the new instance initially possesses. Similarly, if an entity is perceived to no longer exist in the real world, an instance is removed from the cognitive representation. This is modeled by the notion of *domain contraction.*

Definition 15:

Let **T** designate a category of instances, **A** a nonempty set of instances such that $A \subseteq T$, $\mathbf{T'} = T - A$, and $\mathbf{P} = \{p_i | p_i : T \rightarrow V\}$ a property of **T**. A ***domain contraction*** is a change in the domain of **P** such that $P = \{p_i | p_i : T' \rightarrow V\}$.

Example:

Consider the category GLASS = {glass1, glass2} introduced earlier (see page 75). The addition of **glass3** to the domain of ORIENT, CONTENT, and ON would be a case of domain expansion[62]. On the other hand, the removal of **glass2** from the domain of these properties (for example, by smashing it) would be an example of domain contraction.

The remaining kind of change deals with the change of state of existing instances. *Events* are

[62]This change may be relative to the glass-table system of interest, in which case **glass3** may have a previous existence, or the change may be understood as the creation (i.e., perceiving the existence) of **glass3**. The distinction is made here only to show that an instance may enter or leave categories.

defined to describe changes in the active function of one or more properties and, therefore, to describe changes in the state of sets of instances. In the notation which follows, $\mathbf{p}^k_{ik}$ denotes the $\mathbf{i}_k$th function of the structural or relational property $\mathbf{P}^k$.

Definition 16:

Let $\bar{p} = <p^1_{i1},...,p^K_{iK}>$ denote a vector containing the active functions of the **K** structural/relational properties $\mathbf{P}^1,...,\mathbf{P}^K$ of a category of instances **T**. An ***event with respect to T*** is a change in the active function of at least one of $\mathbf{P}^1,...,\mathbf{P}^K$, denoted:

$$e: <\sigma(T),\sigma'(T)> = <\bar{p},\bar{p}'>,$$

where $\bar{p}' = <p^1_{i'1},...,p^K_{i'K}>$ and $p^k_{i'k} \neq p^k_{ik}$, for at least one k=1,...,K.

This definition describes an event as a change in the state of instances in one or more of the sets $\mathbf{T}^1,...,\mathbf{T}^K$ (i.e., the domains of properties $\mathbf{P}^1,...,\mathbf{P}^K$). However, the interest here is only in changes to instances of $T=T^1\cap...\cap T^K$.

Example:

Consider the "glass/table" domain (see page 75). Suppose that **glass1** is *upright* and *full* on **table1** and that **glass2** is *upright* and *empty* on **table2**. That is, the state of the two glasses is given by the vector $\sigma(GLASS)=<s^1_1,s^2_3,r^1_2>$.[63] Further, suppose an event occurs whereby a person removes **glass1** from **table1**, drinks the contents and replaces it *upside_down* on **table1**. The resultant state is $\sigma'(GLASS) = <s^1_3,s^2_1,r^1_2>$. Using the notation introduced above, the event can be described by its before and after states:

$$e: <\sigma(GLASS),\sigma'(GLASS)> = <<s^1_1,s^2_3,r^1_2>,<s^1_3,s^2_1,r^1_2>>.$$

[63]Recall from the earlier description that S^1=ORIENT, S^2=CONTENT, and R^1=ON. Furthermore, s^1_1(glass1)=upright, s^1_1(glass2)=upright, s^2_3(glass1)=full, s^2_3(glass2)=empty, r^1_2(glass1)=table1, and r^1_2(glass2)=table2.

This recognizes that there has been a state change to the category GLASS: that is, to one or more *instances of the category*.

This event can also be described with respect to each glass:

e(glass1): <<s^1_1(glass1),s^2_3(glass1),r^1_2(glass1)>,<s^1_3(glass1),s^2_1(glass1),r^1_2(glass1)>>

= <<*upright,full*,table1>,<*upside_down,empty*,table1>>; and

e(glass2): <<s^1_1(glass1),s^2_3(glass1),r^1_2(glass1)>,<s^1_3(glass1),s^2_1(glass1),r^1_2(glass1)>>

= <<upright,empty,table2>,<upright,empty,table2>>,

where the italicized values are changed by the event.

A note about the semantics of an event is in order here. The event above involves two changes in active function. In addition, the event does not "detect" that a glass is off a table. If the codomain of the property **ON** was extended to include the value *nothing*, one could capture intermediate stages in the above change, and the above event could be described by two (or more) events. This is an issue of "granularity" (or "coarseness" versus "fineness") of events, and reflects an abstraction with respect to the granularity of knowledge. That is, if a person looks more closely at the changes involved in an event, it is often possible to refine a single event as a series of several events. Furthermore, different people may view the same domain with different degrees of granularity.

4.2.2.3.3 Time

The notions of state and event are very important in the handling of time in this model. There are at least two ways to model time as it relates to the states of things. First, time may be considered a parameter of property functions. This explicitly shows the fact that property values and, hence, states, are time-dependent. Using this approach, a property **P** might be regarded as a single function $P:T\otimes\Delta\rightarrow V$, where **T** designates a set of instances, Δ designates a set of time points, and **V** designates a set of values. However, the main drawback of this treatment is that an additional parameter is introduced which is unlikely to vary the value of the function over many values of that parameter. In other words, $P(T,\delta)$ will be constant over many values $\delta \in \Delta$, for a fixed $t \in T$. In addition, this treatment would not be useful for distinguishing unchanging properties such as **Birthdate**, from changing ones such as **Address**.

A simpler way to model time-based change is by "measuring time" only through the occurrence of *events*. In this view, time is not incorporated as a parameter of a property defined as a single function. Instead, a property is modelled by a set of functions, one of which is an accurate reflection of the (real or imagined) world with respect to that property. An event is defined as a change in the function from that set which correctly describes the domain. This is the approach taken in this formalization. Its main advantage is conceptual simplicity, in that time is strictly implicit (i.e., measured only by the occurrence of events) and need not be carried as a parameter of property functions[64]. In addition, it provides a useful way to distinguish changeable from unchangeable properties, the former having several functions in a property set and the latter only one. Furthermore, this treatment allows constraints on the joint values of a property for instances of a category to be conveniently captured through the absence of certain *potential* functions from the actual set of functions that constitutes the property. For example, in the glass-table environment described earlier, there may be a constraint that both glasses may not be on **table2**. This would be reflected in the specific example by the presence of only the functions $\mathbf{r^1_1}$, $\mathbf{r^1_2}$ and $\mathbf{r^1_3}$ in the property ON (thereby excluding the potential function $\mathbf{r^1_4}$ from the set of actual functions, since r^1_4(glass1)=r^1_4(glass2)=table2). Chapter 5 (Section 5.7) contains a more detailed consideration of the meaning of time in the model.

This completes the discussion of the primitive knowledge constructs. Using these, the important

[64]The major disadvantage of this approach is related to implementation, since the set of functions constituting a property may be very large. Since implementation is not the immediate concern of this research, determining potential implementation strategies is left for future research.

derived constructs of *behavioral property*, *concept*, *specialization*, and *composition* are defined, beginning with the notion of behavioral property.

4.2.3 Derived Constructs

4.2.3.1 Behavioral Property

In addition to knowledge about states, an important part of knowledge about things involves their *allowed behavior*, or how states may change over time. However, the inclusion of behavior in defining concepts is not precisely characterized by cognitive science researchers (e.g., Smith, 1988, p.27; Smith and Medin, 1981, p.71). Instead, behavior is often discussed in terms of predicates such as "can fly" (as a property of, say, birds).

Given that behavior is only informally incorporated in theories of concepts, an assumption is needed before the notion of behavioral property can be formally defined. Thus, what follows goes beyond what is explicitly stated in the classical theory of concepts, but is offered as an interpretation of more informal treatments of behavior based on the preceding formulation of structural and relational properties.

The assumption is essentially that predicates such as "flies", "swims", and "walks" (Smith and Medin, 1981, p.72) describe constraints on the ways in which instances may change state (in these cases, properties such as position and orientation of limbs). Everyday observations suggest that the states of things do not change arbitrarily, but behave in a manner that is to some extent regular, or even predictable. This is expressed in the following postulate.

<u>Postulate 3</u>:

The changes of state that a set of instances may undergo are subject to certain restrictions.

Consequently, behavioral properties will be defined as restrictions on changes to the active functions of one or more structural and relational properties. That is, given a certain combination of active functions for one or more structural/relational properties, a behavioral attribute specifies which events are allowed with respect to these same properties.

Definition 17:

Let $\mathbf{P}^1,\dots,\mathbf{P}^K$ denote **K** structural and/or relational properties with domains $\mathbf{T}^1,\dots,\mathbf{T}^K$, respectively. A ***behavioral property*** is a function $\mathbf{b}{:}P^1\otimes\dots\otimes P^K \rightarrow \wp(P^1\otimes\dots\otimes P^K)$.

In other words, given a K-tuple of active functions $(p^1,\dots,p^K)$ of structural attributes $(p^k \in P^k)$, $b(p^1,\dots,p^K)$ returns a set of K-tuples of functions, thereby indicating the allowed joint changes in the active functions of $P^1,\dots,P^K$ given that $\mathbf{p}^1,\dots,\mathbf{p}^K$ are active (see example following Definition 18). The structural and relational properties affected by a behavioral property will be referred to as the *parameters* of that property. Formally:

Definition 18:

Let $\mathbf{b}{:}P^1\otimes\dots\otimes P^K \rightarrow \wp(P^1\otimes\dots\otimes P^K)$ denote a behavioral property. $\mathbf{P}^1,\dots,\mathbf{P}^K$ will be called the ***parameters*** of **b**.

The definition of behavioral property is general and several special cases are possible. For properties having K=1, changes in the active function depend only on the current active function of that property. When K>1, changes in the active functions of **K** properties are jointly constrained by the active function of the remaining **K-1** properties.

Example (K=1):

In the "glass/table" world introduced earlier (see page 75), suppose that there is an arbitrary rule[65] that a single glass cannot change from *upside_down* to *upright* unless the other glass is also *upside_down.* The behavioral property (i.e., the function **b**) is specified in the following table, where S^1 denotes the property ORIENT, and the s^1_i (i=1,...,4) are the functions in the set which assign orientations to glasses[66].

b

Element of S^1	Element of $\wp(S^1)$
s^1_1	$\{s^1_2,s^1_3,s^1_4\}$
s^1_2	$\{s^1_4\}$
s^1_3	$\{s^1_4\}$
s^1_4	$\{s^1_1,s^1_2,s^1_3\}$

Here, allowed changes to the active function of ORIENT depend only on the present active function of ORIENT.

Example (K=2):

Suppose that there is a restriction that a glass which is *upside_down* must be turned *upright* before being filled and a glass which is *upright* must be *empty* before it can be turned *upside_down.* Prohibited by this behavioral property are all events in which any glass changes from a state in which its ORIENTation is *upside_down* and CONTENT is *empty* (or ORIENTation is *upright* and CONTENT is

[65]This means a rule which is not inherent to the system. Arbitrary rules may be akin to "business rules": that is, stated or implied practices followed by an organization. By contrast, the fact that a glass, for example, cannot (normally) be *upside_down* and *full* is based on the law of gravity.

[66]Recall that s_1 = {<g1,u>,<g2,u>}, s_2 = {<g1,u>,<g2,d>}, s_3 = {<g1,d>,<g2,u>}, s_4 = {<g1,d>,<g2,d>}, where ***g1*** and ***g2*** denote the glasses, and ***u*** and ***d*** denote possible orientations.

empty or *full*) to a state in which its ORIENTation is *upside_down* and its CONTENT is *full*.

This example illustrates another potential role of behavioral properties. A generalization of the above behavioral property is that no glass may simultaneously be *upside_down* and *full*. Consequently, a behavioral property can be useful in implicitly providing restrictions on the joint active functions of two or more properties by *prohibiting transitions to such states from any other state*. This requires the following assumption, which is adopted by the model.

***Assumption*:**

If a behavioral property prohibits a transition to a certain state from any other state, then that state may not be the initial state of a (set of) instances.

This assumption ensures that states which cannot be reached from other states are fully prohibited.

It follows from Definition 17 that, when $K>1$, $\mathbf{T}^1,\ldots,\mathbf{T}^K$ (the domains of $\mathbf{P}^1,\ldots,\mathbf{P}^K$, respectively) may be identical, partially overlapping, or disjoint. Consequently, behavioral constraints may involve structural/relational properties outside of $\mathbf{P}_\mathbf{T}$, the set of structural/relational properties shared by a set of instances, **T**. For example, for $\mathbf{P}_\mathbf{A} = \{P_A{}^1, P_A{}^2\}$ and $\mathbf{P}_\mathbf{B} = \{P_B{}^1, P_B{}^2\}$, there may be a behavioral attribute $b{:}P_A{}^1 \otimes P_B{}^2 \rightarrow \wp(P_A{}^1 \otimes P_B{}^2)$, indicating that behavior is linked across categories. Later, when the notion of concept is defined, it will be shown that a behavioral property may, therefore, "belong" to two or more different concepts. This illustrates how the formalization goes beyond what is explicitly discussed in theories of concepts with respect to behavior.

For example, consider a university context in which changes to the active function of the **Courses_offered** property of a department may depend on the active function of the **Teaching_load** property of the faculty members in that department[67]. A department may not be able to offer added courses if, for instance, every faculty member has a teaching load of four or more courses.

To conclude this section, an implied connection of behavioral properties to structural and relational properties is examined. The connection is that, for a given "universe" of properties, any subset of structural and relational properties determines a specific set of behavioral properties which affect the former. Thus, for any category, knowledge about the state of its instances is determined by the active functions of the structural/relational attributes possessed by those instances. Knowledge about the allowed changes of state is determined by the set of behavioral attributes which have as a parameter <u>any</u> structural/relational attribute defining the category.

To state this more precisely, if $\mathbf{P}_T = \{P^1,...,P^M\}$ is a set of structural/relational properties ($P^m:T^m \rightarrow V^m$) shared by a category $T = T^1 \cap ... \cap T^M$, there is an associated set of behavioral properties $\mathbf{B}_T = \{b^1,...,b^K\}$ such that each $b^k \in B$ has <u>at least</u> one property $P^m \in P$ as a parameter of its domain. Consequently, $\mathbf{P}_T$ may also be written $<P_T,B_T>$ or $<S_T,R_T,B_T>$ (since $P_T=S_T \cup R_T$) to emphasize that, in any cognitive representation, a collection of structural and relational properties "implies" a set of behavioral properties which constrain how the active functions of the former may change: that is, $P \Rightarrow B$. This has implications for the definition of concept in the next section.

[67]Such an attribute may be linked with the notion of composition (Section 4.2.3.4) in that a department may be seen as a composite instance which has a set of faculty members as one of its components. The allowed changes of the property of the composite are then dependent on the active functions of properties of components. Developing this notion of linking behavioral properties which "cross categories" to composites is left as future work, since the concept theory literature does not appear to deal with it.

4.2.3.2 Concept

The notion of *category* was introduced earlier as an extensional construct, defined as a set of instances sharing a set of structural and relational properties. However, both the classical and prototype theories of concepts view concepts as *abstractions* of the common properties of a set of instances. In these theories, this notion of abstraction is essential to what a concept is (Medin and Smith, 1984, p.115). For example, the concept of **PERSON** is distinct from knowledge of any individual human, and may be thought of as the set of properties that is common to all humans.

As discussed in Chapter 3, the abstraction of common features or properties is an essential aspect of managing or reducing the complexity of the world, and enables humans to categorize perceived things according to commonality of properties (Smith, 1988). In short, people group similar things together. Similarity will be operationalized here in terms of the *sharing* of properties. Two instances, $\mathbf{t_1}$ and $\mathbf{t_2}$, *share* a structural or relational property, **P**, if $\mathbf{t_1}$ and $\mathbf{t_2}$ possess **P**: that is, if $p_i(t_1)$ and $p_i(t_2)$ are defined for all functions $p_i \in P$. In this regard, a concept is a vehicle for expressing a group of shared properties (i.e., a concept consists of a collection of properties shared by a non-empty set of instances), and identifying collections of properties which are shared by sets of instances may be regarded as the fundamental task of concept formation.

In formally defining the notion of concept, the basic objective is to provide an intension (or abstraction) based on the sharing of properties, since we group instances based on their common properties. In addition, there are several other requirements that the formalization must satisfy. These requirements are proposed here as "principles of conceptualization". They are not explicitly articulated by any of the three major theories of concepts, but are consistent with claims about the fundamental reasons for categorizing the things we perceive: *cognitive economy* and *inference* (e.g., Smith, 1988; Lakoff, 1987; Smith and Medin, 1981).

Principle of Abstraction from Instances:

A concept abstracts the properties shared by a non-empty set of instances.

In *perceiving the world*, we classify instances according to the properties they share with others. If a concept is defined as a set of properties, there must be a non-empty set of instances (corresponding to real or imagined things) which possess all of these properties. Otherwise, the concept would not be useful[68].

Principle of Maximal Abstraction:

Every property possessed by all the instances of a concept is part of the definition of that concept.

This is related to the notion of *cognitive economy* discussed in Chapter 3. According to Rosch (1978, p.28), "the task of category systems is to provide maximum information with the least cognitive effort". In other words, classifying an instance as belonging to a concept enables certain questions to be answered about that individual. Clearly, if a concept definition omits some properties which are possessed by all its instances, it would not be possible to answer questions related to that property when an instance is classified. For example, there can be no concept defined only by the property **Works_in_department**, since all who work in departments are persons, and therefore, share numerous other attributes. In other words, if one can ask of an instance **x**, "Which department does **x** work in?", one can also ask "What is **x**'s address?", "What is **x**'s birthdate?", and so on.

[68]Consider a concept such as **unicorn**, which has no manifestation in the real world. Such a concept could be formed by "imagining" an instance which possesses a certain combination of properties. The point is that instances (corresponding to real or imaginary things) form the basis for the abstraction.

Principle of Non-redundancy:

A concept whose properties are a superset of those of each of several other concepts must contain at least one property not in the union of the properties of these other concepts.

This means that a concept which "specializes" one or more other concepts must contain information (i.e., additional properties) which cannot be found by considering an instance as a member of any of the more general concepts. For example, suppose that **CUSTOMER**, **EMPLOYEE**, and **CUSTOMER_EMPLOYEE** are concepts, and the set of properties of **CUSTOMER_EMPLOYEE** is a superset of those of each of the other two. If the collection of properties defining **CUSTOMER_EMPLOYEE** is to be a "meaningful" abstraction (in the sense of permitting more questions to be answered), it must contain at least one property that is not in the definition of either **CUSTOMER** or **EMPLOYEE**. Otherwise, any question that can be asked of an instance of **CUSTOMER_EMPLOYEE** can be asked of the same instance as either an **EMPLOYEE** or a **CUSTOMER**. This principle is based on an assumption, implicit in many theories of semantic memory (e.g., Smith, 1978), that conceptual organization is efficient in the sense that each refined concept adds new knowledge.

Principle of Completeness:

Given a "universe of knowledge" consisting of (1) a set of instances, (2) the set of all properties possessed by any of these instances, and (3) a set of concepts, every property will be used in the definition of at least one concept in that set.

This principle essentially states that no properties are omitted in the abstraction of some universe of knowledge. If a property was not abstracted in the definition of any concept, there would be some questions about the set of instances possessing that property which could not be answered by classifying those instances into any available concept. In other words, properties are a crucial classification tool, and are not normally considered independently of the concepts they define.

Principle of Variety:

> Given a "universe" of knowledge (a domain), there are many ways of conceptualizing that domain which satisfy the first four principles.

This principle recognizes that there is no single "correct" set of concepts for abstracting one's knowledge about a domain (cf. Lakoff, 1987). Instead, any "world view" which satisfies the previous principles is as valid as any other. Different individuals may abstract different sets of properties to form different concepts depending on various factors, such as the context within which the knowledge will be used (e.g., Barsalou, 1982; Rosch, 1978) and the culture within which the individual lives (Lakoff, 1987). However, any abstraction which violates any of the earlier principles violates some elementary assumption about the nature and reasons for classification.

These principles are offered here as "desiderata" of conceptualization. They are based on fundamental assumptions about why we categorize, and constitute a minimal set of conditions for deciding whether a collection of concepts is "good" for some universe of knowledge. In what follows, the principles are used to define two complementary constructs which give precision to the notion of a concept. A ***potential concept*** is defined as a set of properties which embodies the principles of *abstraction from instances* and of *maximal abstraction*, while a ***concept structure*** is defined as a set of potential concepts which satisfies the principles of *non-redundancy*, *completeness*, and *variety*.

In defining concepts, it is useful to recognize the connection between a concept as an intension, and its *extension,* which is a category (Definition 6, page 65). The extension of any set of structural/relational properties, **P**, is the joint intersection of the domains of these properties, and will hereafter be denoted **e(P)**.

To define *potential concept,* the following notation is used. **T** denotes the complete set of instances (universe) in a cognitive representation, $\mathbf{P_T} = \{P^1,...,P^M\}$ denotes the set of all structural and relational properties possessed by any members of **T**, and $\mathbf{T^m}$ denotes the domain of $\mathbf{P^m}$ (m=1,...,M). This means that $T=T^1\cup...\cup T^M$. Finally, $\mathbf{C} = \{P^j\}_{j\in J}$, $J\subseteq\{1,...,M\}$ denotes a subset of $\mathbf{P_T}$, and **e(C)** denotes the extension of **C**.

<u>*Definition 19*</u>:

C will be called a ***potential concept in T*** iff

a) $e(C)\neq\emptyset$, and

b) $\nexists$ $P'\in(P_T-C)$ such that $T'\supseteq e(C)$ (where **T′** is the domain of **P′**).

This definition ensures that every potential concept (a) has a non-empty extension, and (b) contains all properties possessed by all instances in its extension. Hence, it satisfies the principles of *abstraction from instances* and *maximal abstraction.* The example given earlier, where C = {Works_in_department}, does not constitute a potential concept, since all instances which possess the property **Works_in_department** also possess numerous other properties, such as **Address** and **Birthdate** (i.e., all who work in departments are persons). Hence, the domain of the latter two properties is a superset of that of the first.

Note that, for any $C=\{P^1,...,P^K\}$, $C\Rightarrow B$, where **B** denotes a set of behavioral properties. Therefore, C may be written <P,B> or <S,R,B> to emphasize that a potential concept also includes the behavioral properties which affect the structural/relational properties in its definition (see page 87).

The definition of potential concept leads to a theorem about an importance characteristic of distinct potential concepts which has implications for concept specialization (see Section 4.2.3.3). Note, first, that two distinct potential concepts cannot have the same set of properties (else they are the same).

Theorem 1:

Two potential concepts cannot have the same extension.

Proof:

Suppose that $C_A=\{P_A^1,...,P_A^M\}$ and $C_B=\{P_B^1,...,P_B^N\}$ denote two potential concepts. Since $C_A \neq C_B$, there is at least one property **P′** (with domain **T′**) which is in either $\mathbf{C_A}$ or $\mathbf{C_B}$, but not both. Suppose $P' \in C_A$. By Definition 19, $T' \not\supseteq e(C_B)$ (in particular, $T' \neq e(C_B)$). Yet $T' \supseteq e(C_A)$ since it is possessed by the instances of $\mathbf{C_A}$. Therefore, $e(C_A) \neq e(C_B)$. The result is the same if $P' \in C_B$. ■

Every person forms many concepts for categorizing instances. For a variety of reasons related to the goal of reducing the complexity of experience (Lakoff, 1987; Medin and Smith, 1981), people identify certain groups of properties as important. The reasons for identifying collections of properties as useful abstractions has not been fully explained by cognitive scientists and is beyond the scope of this research. However, it is reasonable to conclude that particular groupings are useful because they permit inferences to be made about non-perceived properties (given perceived ones), thereby supporting adaptation and survival (Smith, 1988).

A set of concepts used to organize knowledge about a domain will be called a ***concept structure***. A concept structure must satisfy the principles of *non-redundancy*, *completeness*, and *variety*, thereby imposing some constraints on the nature of the potential concepts it contains.

Let **T** denote a "universe" of instances, $\mathbf{P_T} = \{P^1,...,P^M\}$ denote the set of all structural/relational properties possessed by any members of **T**, **CS** = $\{C^1,...,C^K\}$ denote a set of potential concepts, and **CS′** denote any subset of **CS**.

Definition 20:

CS is a ***concept structure*** *over T* iff

a) $\cup_{k\in\{1,...,K\}}C^k=P_T$, and

b) $\nexists$ $J\subseteq\{1,...,K\}$ such that $\cup_{j\in J}C^j=C\in CS$, where $C\neq C^j$ for any $j\in J$.

If **CS** is a concept structure, each potential concept $C^k\in$ CS will be referred to as a *realized concept* (or simply a *concept*).

By its definition, a concept structure (a) contains every property possessed by any members of **T** in at least one of its concepts, and (b) contains no concept whose properties are exactly the union of the properties of any other concepts in the structure. The second condition is important in defining concept *specialization* (see Section 4.2.3.3). The definition prohibits a concept structure from containing potential concepts which do not have at least one extra property with respect to its superconcept (if only one), as well as potential concepts which simply contain the union of the properties of two (or more) superconcepts.

To illustrate the definition, an example of a collection of potential concepts which may not be part of the same concept structure is given. Suppose that CUSTOMER={Name, Address, Spouse, Holds_account}, EMPLOYEE={Name, Address, Spouse, Experience}, and CUSTOMER_EMPLOYEE={Name, Address, Spouse, Holds_account, Experience}. In this case, {CUSTOMER, EMPLOYEE, CUSTOMER_EMPLOYEE} does not constitute (a subset of) a concept structure since CUSTOMER_EMPLOYEE=CUSTOMER∪EMPLOYEE. This means that, as defined here, the potential concept CUSTOMER_EMPLOYEE abstracts no additional information with respect to the other concepts considered, since all questions that can be answered with respect to any instance of CUSTOMER_EMPLOYEE can be answered with respect to that instance as a member of CUSTOMER or EMPLOYEE.

4.2.3.3 Specialization

The concept is the fundamental abstraction mechanism by which instances can be compared. In addition, concepts can be associated with other concepts based on the degree to which they share properties. This leads to the consideration of concept **specialization**.

There is a great deal of evidence suggesting that we organize knowledge about things into general and more specialized categories. Biological taxonomies are a clear example. Keil (1979, 1986) suggests that there is an ontological hierarchy of generic concepts common to all humans. A segment of one path on this hierarchy is THING → PHYSICAL THING → SOLID → LIVING THING → ANIMAL → ....

Given the definitions of potential concept and concept structure, the notion of *concept specialization* can be formalized. In accordance with the intuitive meaning of specialization in classical concept theory, the following principle, which is a special case of the *principle of non-redundancy*, guides the formal definition.

Principle of Refinement:

The set of properties possessed by a specialized concept is a strict superset of the set of properties of a more general concept.

This principle is used to define specialization intensionally. Specialization also has the extensional meaning that the instances of a specialized concept are a strict subset of those of a more general concept and, therefore, are also instances of the latter. The definition will be shown to satisfy this condition.

Definition 21:

Let C^1 and C^2 denote two concepts in a concept structure (i.e., C^1 and C^2 are each a set of properties). Then, C^2 is a ***specialization (or subconcept)*** of C^1 (alternatively, C^2 ***IS-A*** C^1) iff $C^1 \subset C^2$.

If C^2 is a subconcept of C^1, then C^1 may be referred to as a *superconcept* of C^2. For example, if PERSON = {Name, Address, Birthdate} and CUSTOMER = {Name, Address, Birthdate, Holds_accounts} are two concepts in a concept structure, then CUSTOMER IS-A PERSON.

If a concept is a specialization of two or more others, it must possess added properties with respect to the union of those of all its superconcepts. Otherwise, all questions about instances can be answered with respect to one or more of the more general concepts. This is implied by the definitions of concept structure and specialization, as shown by the following theorem.

Theorem 2:

Let C^0, C^1 denote two concepts in a concept structure such that neither is a specialization of the other. Then, if C^2 is a specialization of both C^0 and C^1, $C^0 \cup C^1 \subset C^2$.

Proof:

Since C^2 IS-A C^0 and C^2 IS-A C^1, $C^0 \subset C^2$ and $C^1 \subset C^2$ (by Definition 21). Therefore, $C^2 \supseteq C^0 \cup C^1$. But, by the definition of concept structure, $C^2 \neq C^0 \cup C^1$. Therefore, $C^2 \supset C^0 \cup C^1$. ■

The definition of specialization satisfies the basic extensional intuition of specialization - that the extension of a specialized concept is a strict subset of that of a more general one, as shown by the following theorem.

Theorem 3:

Let $e(C^1)$, $e(C^2)$ denote the extensions of C^1 and C^2, respectively, where C^2 IS-A C^1. Then, $e(C^2) \subset e(C^1)$.

Proof:

Since C^2 IS-A C^1, $C^1 \subset C^2$ (by Definition 21). Let T' denote the intersection of the domains of the properties in (C^2-C^1). Since C^1 is a potential concept, there is no property in (C^2-C^1) whose domain is a superset of $e(C^1)$. Therefore, $T' \not\supseteq e(C^1)$. By the definition of extension (category) (Definition 6, page 65), $e(C^2) = e(C^1) \cap T'$. Therefore, $e(C^2) \subset e(C^1)$. ■

The restriction that a specialized concept must possess additional properties with respect to concepts which it specializes is not generally made in the existing concept literature. Hence, the formal model denies two potential grounds for defining specializations. The first is the definition of subconcepts based only on the addition of ***behavioral properties***. An example of this might be the distinction of two "kinds" of EMPLOYEE: those who are authorized to lay off employees versus those who are not. The second is the definition of subconcepts based only on differentiating instances of an existing concept by specific values of one or more structural or relational properties. An example of this might be the distinction of two "specializations" of EMPLOYEE - those who work at the *Vancouver* office versus those who work at the *Seattle* office. In both cases, it is proposed here that such subconcepts are useful only if they also possess additional structural and/or relational properties. In fact, according to the model this is the only basis for defining subconcepts. The importance and contribution of these distinctions is discussed further in Section 5.2.

In concluding this section, some implications of the definition of specialization for behavioral properties are examined. Recall from the previous section that a set of structural/relational properties implies an associated set of behavioral properties, each of which constrains changes to some properties in the former set. That is, a behavioral property is a function $\mathbf{b}:P^1\otimes\ldots\otimes P^K\rightarrow\wp(P^1\otimes\ldots\otimes P^K)$ specifying which changes in active function may occur given any combination of active functions of the structural/relational properties $P^1,\ldots,P^K$.

The set of behavioral properties of a concept $\mathbf{C}^1$ is denoted $B^1 = \{b^1\ldots b^M\}$ and is implied by the structural/relational properties in $\mathbf{C}^1$ (Section 4.2.3.1). If $\mathbf{C}^2$ is a specialization of $\mathbf{C}^1$, then $\mathbf{C}^2$ possesses at least one structural or relational property which $\mathbf{C}^1$ does not. Now, since each member of $\mathbf{C}^2$ is a member of $\mathbf{C}^1$, it is subject to the behavioral properties of $\mathbf{C}^1$. However, when considering an instance as a member of the specialized concept, there may be additional restrictions on behavior related to the extra structural/relational properties: that is, $\mathbf{B}^1\subseteq\mathbf{B}^2$.

Two cases can be distinguished. First, a behavioral property of the subconcept may extend (subsume) a behavioral property of the superconcept to incorporate restrictions with respect to an added structural/relational property. For example, there may be a property $\mathbf{b}$ of $\mathbf{C}^1$ such that $b:P^1\otimes\ldots\otimes P^K\rightarrow\wp(P^1\otimes\ldots\otimes P^K)$, and a property $\mathbf{b}^*$ of $\mathbf{C}^2$ such that $b^*:P^1\otimes\ldots\otimes P^K\otimes P^{K+1}\rightarrow\wp(P^1\otimes\ldots\otimes P^K\otimes P^{K+1})$, where $\mathbf{P}^{K+1}$ is a structural or relational property defining $\mathbf{C}^2$ as a specialization of $\mathbf{C}^1$.

The semantics of $\mathbf{b}^*$ with respect to $\mathbf{b}$ is as follows. For any fixed element of $\mathbf{P}^{K+1}$, $b^*(p^1,\ldots,p^{K+1})$ "subsumes" $b(p^1,\ldots,p^K)$. To see what this means, consider a case where $b:S\rightarrow\wp(S)$ and $b^*:S\otimes R\rightarrow\wp(S\otimes R)$, where $\mathbf{S}$ and $\mathbf{R}$ denote a structural and relational property, respectively. Now, for each $s_i\in S$, $b(s_i)=Q_i$, where $\mathbf{Q}_i$ denotes an element of the power set of $\mathbf{S}$. That is, $\mathbf{Q}_i$ is some subset of $\mathbf{S}$, such as $Q_i=\{s_1, s_2, s_3\}$. Similarly, for any $s_i\in S$ and $r_j\in R$, $b^*(s_i,r_j)=Q^*_{ij}$, where $\mathbf{Q}^*_{ij}$ denotes an element of the power set of $S\otimes R$ (i.e., some subset of $S\otimes R$, such as $Q^*_{ij}=\{(s_1,r_1), (s_2,r_1), (s_3,r_3)\}$). $\mathbf{b}^*$ subsumes $\mathbf{b}$ in the sense that, for any $s_i\in S$, the set of first elements of each pair in $\mathbf{Q}^*_{ij}$, denoted $\mathbf{Q}^*_i$ (i.e., $\{s_1, s_2, s_3\}$), is in the above case the same as the set $\mathbf{Q}_i$. This means that there should not be any function $\mathbf{s}_4$ in any ordered pair in $\mathbf{Q}^*_{ij}$).

Otherwise, $\mathbf{b}^*$ contradicts **b** when elements of the subconcept are considered as elements of the superconcept. In this situation, $\mathbf{b}^*$ will be said to be *compatible* with **b**.

The second case involves behavioral properties that do not extend those of the superconcept. An example would be a property of $\mathbf{C}^2$, $b^*:P^1\otimes...\otimes P^M\otimes P^{M+1}\rightarrow\wp(P^1\otimes...\otimes P^M\otimes P^{M+1})$, where $P^1,...,P^M$ are properties of $\mathbf{C}^1$, P^{M+1} is an added property of $\mathbf{C}^2$, and there is no attribute $b:P^1\otimes...\otimes P^M\rightarrow\wp(P^1\otimes...\otimes P^M)$ of concept $\mathbf{C}^1$.

4.2.3.4 Composition

The remaining major construct of classical concept theory to be formalized is *composition.* Composition/decomposition is, in addition to generalization/specialization, a means of managing the complexity of the world (e.g., Lakoff, 1987, p.273). Complex instances may be viewed as composed of a number of component instances, each having independent properties. However, the components can be combined to account for certain *emergent properties*: that is, properties ascribed to the whole, but not present in any subset of the components (including emergent behavioral properties which may enforce restrictions on changes of joint states in the composite). This is an important construct which recognizes that certain things "belong" together for some purposes, but their components can be understood as things possessing their own properties.

Intuitively, a *composite concept* is made up of several component concepts and, additionally, possesses a number of *emergent properties.* An instance of such a concept, then, is a surrogate composed of the surrogate of one instance of each of its components. The domain of the emergent properties of the concept will be such surrogates[69].

[69]An alternative way to model composition would be to have a separate surrogate for the composite instance that is independent of its components. One feature of such an approach would be that it would treat the composite instance as having an existence independent of its components, in that some components could be removed or changed without affecting the existence of the composite. By not adopting this approach, this work takes the position that a composite instance changes if any of its

To support the definition of a composite concept, the notions of *composite* (versus simple) structural and relational properties are first introduced. A *composite structural property* associates a collection of component instances from one or more categories with a unique value, and is not possessed by any subset of the components.

Definition 22:

Let **$e(C^1),...,e(C^K)$** denote the extensions of **K** concepts in a concept structure, **$e(C) \subseteq e(C^1) \otimes ... \otimes e(C^K)$**, **V** denote a set of values, and $S=\{s_i | s_i : e(C) \rightarrow V\}$. **S** will be called a ***composite (emergent) structural property of degree K.***

For example, **Grade** may be considered as a composite structural property of degree two. The domain of this property is a subset of e(STUDENT)⊗e(COURSE), where **STUDENT** and **COURSE** denote two concepts. A grade cannot be associated with either a student or a course, but only with a (student,course) pair.

Next, the notion of *composite relational property* is defined. A composite relational property associates a collection of component instances from one or more component concepts with a unique set of instances from one or more other concepts, and is not possessed by any subset of the components. Let $e(C^1),...,e(C^K)$ denote the extensions of **K** concepts in a concept structure, $e(C) \subseteq e(C^1) \otimes ... \otimes e(C^K)$, $e(C^{K+1}),...,e(C^{K+M})$ denote the extension of **M** other concepts, and $Q \subseteq \wp(e(C^{K+1}) \otimes ... \otimes e(C^{K+M}))$ (i.e. **Q** contains the sets of instances which are linked to the elements of the domain)).

components change. In addition, at any time there is a maximum of one composite of the same set of components.

Definition 23:

Let $\mathbf{R} = \{r_i | r_i\text{:}e(C) \rightarrow Q\}$. **R** will be called a ***composite (emergent) relational property of degree K.***

For example, if a student enrolled in a course is assigned a tutor for that course, **Tutor** may be considered an emergent relational property whose domain is, again, a subset of e(STUDENT)⊗(COURSE), and codomain is a subset of STUDENT. That is, a tutor may not be associated with just a student or a course, since a student may have different tutors for different courses, and a several tutors may provide help in a given course[70].

A composite structural/relational property is defined on a domain which includes instances from several concepts. These will be referred to as the *components* of the property.

Definition 24:

Let $\mathbf{C^1,\ldots,C^K}$ be a set of concepts, $\mathbf{e(C) \subseteq e(C^1) \otimes \ldots \otimes e(C^K)}$, and **P** be a composite property with domain **e(C)**. Then, $C^1,\ldots,C^K$ will be referred to as the ***components*** of **P**.

For example, the **Tutor** property just described has components STUDENT and COURSE.

Given a group of emergent structural and relational properties, it is possible to recognize restrictions on the changes to the active functions of these, giving rise to the notion of *composite behavioral property.*

[70]This example glosses over some added restrictions on the domain and codomain of such a property. For example, a student may not be a tutor in any course which s/he is enrolled.

Definition 25:

Let $P=\{P^1,...,P^K\}$ denote a set of composite structural and/or relational properties. A ***composite behavioral property*** is a function $\mathbf{b}:P^1\otimes...\otimes P^K\rightarrow \wp(P^1\otimes...\otimes P^K)$.

An example of such a property will be given shortly.

These definitions allow the notion of a *composite potential concept* to be defined as a triple of composite structural, relational, and behavioral properties.

Definition 26:

Let **C** be a potential concept <P,B> (i.e., $P\Rightarrow B$). **C** will be called a ***composite potential concept** of $C^1,...,C^K$*, iff **P** and **B** are sets of composite (structural/relational and behavioral) properties and the shared domain of the properties in **P** is a non-empty set[71].

Given this definition, an instance $\mathbf{c}_i$ of **C** is a surrogate $(c^1_i,c^2_i,...,c^K_i)$, where $c^k_i\in C^k$ and $(c^1_i,c^2_i,...,c^K_i)$ is an element of the common domain of $P^1,...,P^M\in P$. The nature of the emergent properties implies that, at any time, there is only one instance of a composite concept having exactly the same component instances. The implications of this are considered further in Chapter 5 (Section 5.5).

[71]In order for a composite concept to have a non-empty extension, the shared domain of its structural and relational properties must contain a number of composite objects. This means that all structural and relational properties must have the same components.

Example:

ship$_1$ = $(c^1,...,c^K)$, where

c^1 = propeller13
c^2 = hull4
c^3 = motor9
.
.
.

The composite concept definition is:

SHIP = <P,B> = <S,R,B> where

S = {Speed, Maximum_Speed, Cargo . . .}
R = {Docked(SHIP,PORT), Captain(SHIP,PERSON) . . .}[72]
B = {Change_course, Change_Speed, Load, Unload . . .}.

Note that some component instances (concepts) can, in turn, be composite instances (concepts). The following postulate suggests that there is finite number of component "levels".

Postulate 4:

A composite concept can be decomposed in a finite number of steps into *primitive* concepts[73].

To illustrate the idea of an emergent behavioral property on the emergent structural and relational properties, consider **ship1**, in state:

<<0,30,Nil,Wheat, . . .>,<Docked(ship1,Vancouver),Captain(ship1,Smith), . . .>>.

A reasonable behavioral property would be that the value of the **Speed** property cannot be changed to a

[72]Recall that R(X,Y) means that **R**:$X \rightarrow \wp(Y)$ is a relational property.

[73]A primitive concept is one which does not contain any component concepts: that is, its structural and relational properties are simple.

value greater than zero unless the value of the **Docked** property is **Nil** (or simultaneously changed to **Nil**).

For some emergent properties, there will be a *formula* that relates the value of the composite property to the value of certain properties of the components. For example, the **Weight** of a ship is the sum of the **Weight** of the components. For other properties (e.g., **Cargo**), there will be no such formula relating properties of the composite to properties of the components.

From these definitions, it is clear that composite potential concepts can form a composite concept structure which is distinct from the concept structure to which its components belong. A composite concept structure is defined as follows.

Definition 27:

Let **CS** = $\{C^1,...,C^K\}$ denote a concept structure. **CS** will be called a ***composite concept structure*** iff $C^1,...,C^K$ are composite concepts.

Composite concept structures will not be explored further here. However, to illustrate the definition, consider a concept structure containing the class SHIP and various specializations. When SHIP is considered as a composite concept, as in the previous example, the consideration of extra emergent properties defined on subsets of its extension can generate a collection of composite concepts (SHIP and its subconcepts, such as freighter, tanker, etc.) which constitute a composite concept structure.

This concludes the formalization of the important elements of classical concept theory. In the next section, requirements for the translation of this model into a model of an IS for constructing "good" representations of knowledge are discussed. Subsequently, the representing (or object) model is defined.

4.3 ON DIRECT REPRESENTATION AND GOODNESS

Given the formalization of classical concept theory presented in the previous section and the research objective of developing a conceptual IS model for expressing knowledge according to users conception of a problem domain, the issue at hand is how this model should be defined. The approach taken here is to define a set of constructs which are in *direct correspondence* with the formally defined constructs of the classical theory. Directness is operationalized in terms of a 1-1 mapping between the important elements of the classical view (as defined in the previous section) and those of the conceptual model or representation language[74].

Directness is sought in order to provide a model containing representation mechanisms which parallel or ***mimic*** the conceptual organization of knowledge about things (according to one theory). This model should support the creation of specific representations which are "good". Goodness of representation has been characterized in terms of four individually necessary and jointly sufficient conditions - *mapping*, *tracking*, *reporting*, and *sequencing* (Wand and Weber, 1988). These conditions essentially ensure that a good representation is:

1. well-defined (mapping and tracking) , and
2. well-behaved (reporting and sequencing).

Well-definedness can be taken to mean, in the context of the terminology of the previous section, that an information system representation has the capability of capturing any state (mapping) and any event (tracking) of a cognitive representation of reality. This allows the state of a specific representation to parallel the state of knowledge about a domain.

Whether an information system does actually track the state of knowledge is an issue of *well-behavedness* and depends on whether (i.e., reporting) and when (i.e., sequencing) changes to the state of

[74]Of course, the definitions in the previous section are a representation of, and not actual, mental constructs. Nevertheless, a distinction is made between the model of concepts or knowledge, and the model of IS which deals with constructs for representing that knowledge.

knowledge are reported to the information system. This is certainly important with respect to any implementation, but is not of concern in developing the conceptual model. Consequently, only well-definedness is addressed in the formulation which follows.

4.4 MIMIC: AN OBJECT-BASED INFORMATION SYSTEMS MODEL

4.4.1 Preliminaries

In this section, MIMIC (Morphological Information systems Model of Instances and Concepts) is defined as a model to support representation of cognitive constructs. Five important constructs are defined based on the five major elements of classical concept theory - instance, property, concept, specialization, and composition. As in the concept model, these are divided into *primitive* and *derived* constructs.

The model will be referred to as an *object model* (or model of objects). One of the arguments in favor of object-oriented computing has been its so-called "naturalness", or correspondence to the way people structure knowledge (Mylopoulos, 1990). However, object constructs have not been clearly related to cognitive constructs. Under the assumption that ***objects represent knowledge about instances of concepts***, MIMIC is presented as a prescription for what object-oriented approaches "should" contain to support the representation of cognitive constructs. A comparison of MIMIC to other characterizations of the object paradigm is given in Chapter 5 (Section 5.8).

Examples are not provided here for each of the constructs as they are introduced, since they are directly analogous to those of Section 4.2, with the difference being that they deal with the external symbolic representation level, rather than the cognitive level (see Figure 4-1). Instead, Appendix 1 develops a detailed example from a banking domain to illustrate use of the model.

4.4.2 Primitive Constructs

In the knowledge model, instances and properties were defined as primitives, from which the remaining constructs were derived. Counterparts of these two basic constructs, called ***objects*** and ***attributes*** respectively, are defined next.

4.4.2.1 Object

An *object* is defined as a surrogate of an instance of a concept. That is, a given set of objects is in 1-1 correspondence with a set of instances. Recall (Definition 1, page 60) that an identification function of a set **O** in a set **T** is a bijective function **f**:T→O.

Definition 28:

Let **T** be a set of instances, **O** a set of symbols, and **f** an identification function of **O** in **T**. **O** will be referred to as a set of ***objects representing T*** via **f** (or simply, **O** *represents* **T**). Furthermore, $o \in O$ will be referred to as an ***object representing*** $t \in T$ iff $f(t)=o$.

For example, T may be a set of instances representing students (i.e., T constitutes one's knowledge of the existence of a set of students) and **O** may be a set of student numbers assigned to these instances. Objects are, therefore, symbols which represent the *existence* of instances of concepts. The following postulate assumes the existence of a set of objects to represent any set of instances of interest.

Postulate 5:

For any finite set of instances, **T**, there exists a set of symbols (objects) **O** and an identification function **f**:T→O.

Corollary:

Let **O** denote a set of objects representing a set of instances **T** via the identification function **f**. Then, there exists a bijective function $\mathbf{f}^{\wp}$: $\wp$(T)→$\wp$(O) such that, for each $T^* \in \wp(T)$, with $f^{\wp}(T^*)=Os$, $\{f(t)\}_{t \in T^*}=O^*$.

This corollary means that there is a 1-1 correspondence between $\wp$(**T**) and $\wp$(**O**) via $\mathbf{f}^{\wp}$ and is used later in the definition of relational attributes.

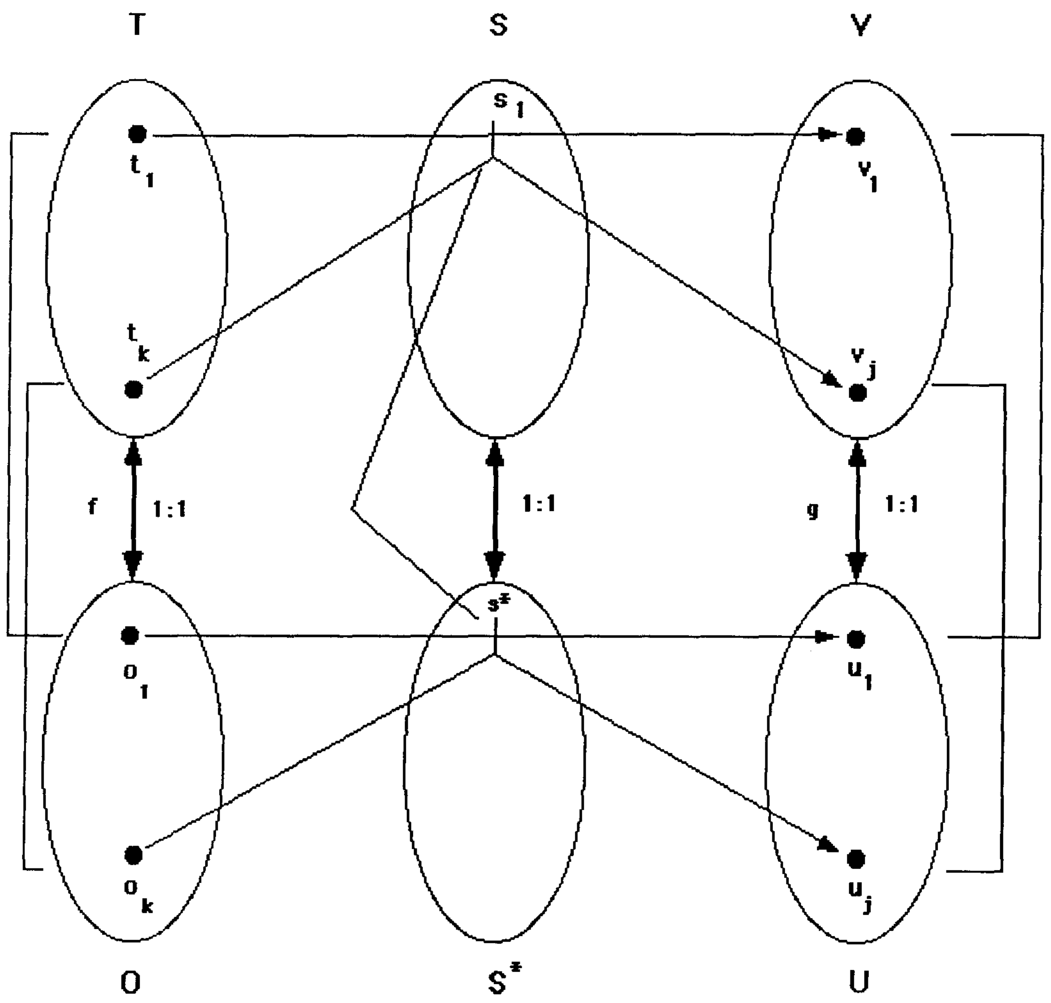
4.4.2.2 Structural Attribute

A *structural attribute* is a representation of a structural property. Intuitively, this means that there is a 1-1 correspondence between sets of structural property functions and structural attribute functions, between sets of instances and objects constituting the domains of these functions, and between sets of values of structural properties and surrogates values of structural attributes constituting the codomains of the functions.

In defining the construct formally, the following notation is used. $S=\{s_i|s_i:T\rightarrow V\}$ denotes a structural property of the set of instances **T** (e.g., Weight:PERSON→WEIGHT_VALUE), **O** denotes the set of objects representing **T** via **f**:T→O (e.g., **O** contains identification numbers), and **g**:V→U denotes a bijective function such that **U** is a set of symbols representing values (e.g., **U** is a set of symbols designating numbers). Finally, $S^*=\{s^*_i|s^*_i:O\rightarrow U\}$ denotes a set of functions defined on O such that $|S^*|=|S|$.

The essence of the definition is as follows (see Figure 4-4). Consider any $t \in T$ such that $s_i(t)=v_i \in V$ and $g(v_i)=u_i \in U$. It is also the case that $f(t)=o \in O$. S^* will be called a structural attribute if, for each $s_i \in S$, there exists $s^*_i \in S^*$ such that $s^*_i(o)=u_i$. Definition 29 expresses this formally.

Figure 4-4: Structural Attribute



Definition 29:

S^* is a *(simple)*[75] ***structural attribute representing structural property S*** iff for each $s_i \in S$, $\exists$ a unique $s^*_i \in S^*$ such that

$$s^*_i(f(t)) = g(s_i(t)) \ \forall \ t \in T.$$

A set of objects, **O**, sharing the set of attributes $\mathbf{S^{*O}}$, will be referred to as a *category* of objects on **O** (cf. Definition 6, page 65).

Next, the existence of structural attributes to represent structural properties is postulated.

Postulate 6:

For any structural property **S**, with domain **T**, there exists a structural attribute $\mathbf{S^*}$ with domain **O**, such that **O** represents **T** (in the sense of Definition 28) and $\mathbf{S^*}$ represents **S** (in the sense of Definition 29).

4.4.2.3 Relational Attribute

A *relational attribute* is a representation of a relational property. Intuitively, this means that there is a 1-1 correspondence between relational property functions and relational attribute functions, and between things and surrogates (on each side of the relation). A relational attribute maps each object in a set to a set of other objects.

[75]The definition can be extended to *composite* structural attributes (e.g., $S^*:O^1 \otimes O^2 \rightarrow U$), which are analogous to composite structural properties (e.g., $S:T^1 \otimes T^2 \rightarrow V$) (see Definition 22, page 100).

The following notation is used. **R**$=\{r_i|r_i:T\rightarrow Q\}$ (where $Q\subseteq\wp(T^1)$) denotes a relational property of things in **T** with things in $\mathbf{T^1}$. **O**, $\mathbf{O^1}$ denote sets of objects representing **T** and $\mathbf{T^1}$, respectively, via the identification functions **f**:T→O and $\mathbf{f_1}$:$T^1\rightarrow O^1$, and $\mathbf{Q^*}\subseteq\wp(O^1)$. Then, there exists a bijective function $f_1^{\wp}:Q\rightarrow Q^*$ which satisfies the Corollary to Postulate 5 (page 108). Finally, $\mathbf{R^*}=\{r^*_i|r^*_i:O^1\rightarrow Q^*\}$ denotes a set of functions defined on **O** with respect to $\mathbf{O^1}$ such that $|R^*| = |R|$.

Definition 30:

$\mathbf{R^*}$ is a *(simple)*[76] ***binary relational attribute*** representing the binary relational property **R** iff for each $r_i\in R$, $\exists$ a unique $r^*_i\in R^*$ such that

$$r^*_i(f(t)) = f_1^{\wp}(r_i(t))\ \forall\ t\in T \text{ (Figure 4-5).}$$

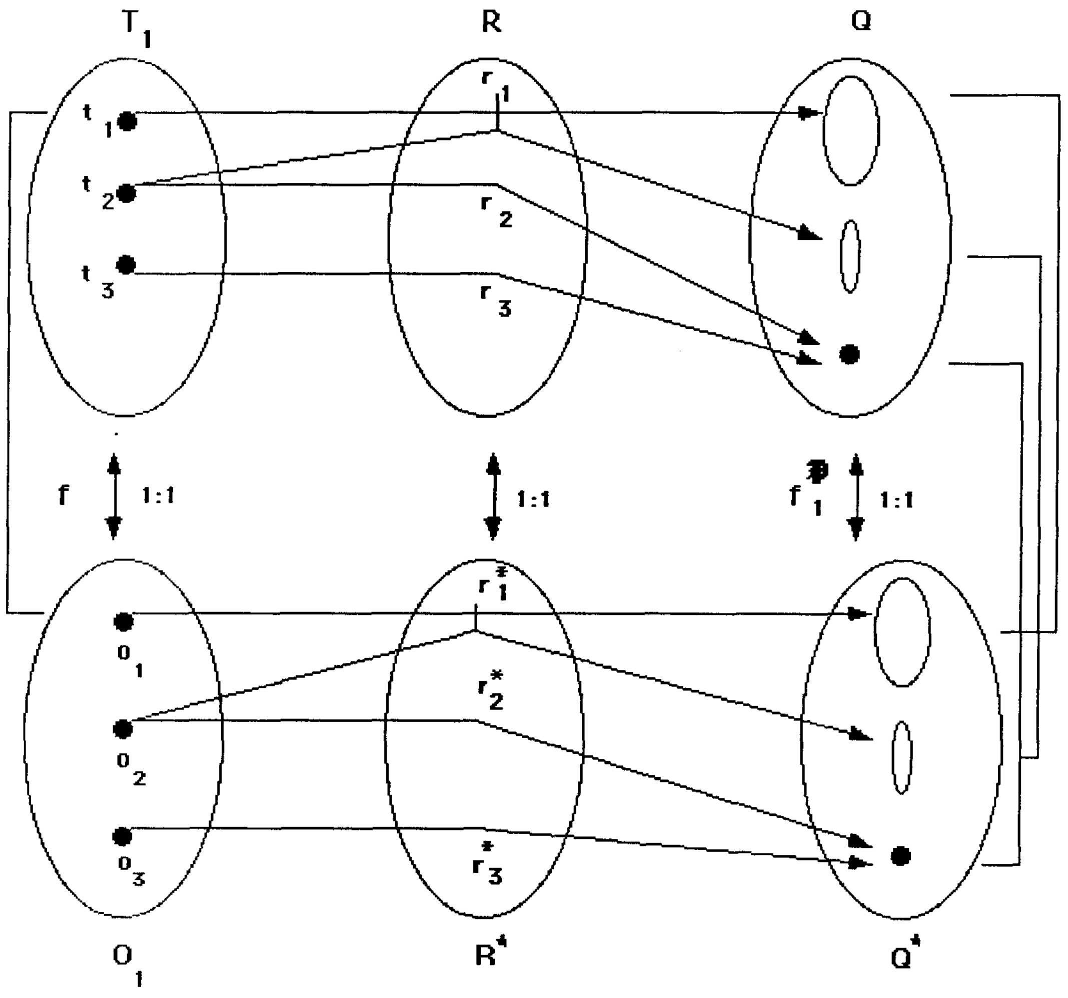
A set of relational attributes of **O**, denoted by $\mathbf{R^{*O}}$, will be referred to as a *category* on **O**.

Postulate 7:

For any relational property **R**, with domain **T**, there exists a relational attribute $\mathbf{R^*}$ with domain **O**, such that **O** represents **T** (in the sense of Definition 28) and $\mathbf{R^*}$ represents **R** (in the sense of Definition 30).

[76]The definition can be extended to define *composite* relational attributes.

Figure 4-5: Binary Relational Attribute



Extended definitions can be given to cover n-ary relationships and distinguish mandatory from optional attributes. As these mirror the corresponding definitions of properties (Definitions 9 and 10), they are not presented here.

4.4.2.4 Object State and Change

4.4.2.4.1 Object State

As with the instances being represented, the objects in a representation can also be described in terms of their state. Object states are characterized by the active functions of the structural and relational attributes of objects $o \in O$ (which identify things $t \in T$). These functions, in turn, determine values of surrogates $u \in U$ (of $v \in V$) or sets of objects $q^* \subseteq Q^*$ (of $q \subseteq Q$), when applied to objects $o \in O$. The following definitions for objects mirror corresponding state definitions of instances (Definitions 11-13).

Recall that defining the state of a category of instances is useful for emphasizing that we think of instances by classifying them based on shared properties (see Definition 11, page 73). Therefore, object states will similarly be described by the attributes they share. That is, it makes sense to speak of the state of an object only as a member of a category of objects.

Definition 31:

Let **O** denote a category of objects, and $\mathbf{P}^{*O}=\{P^{*1},...,P^{*K}\}$ denote the set of structural/relational attributes shared by **O**. The ***state*** of **O**, designated σ(**O**), is a vector containing the active function of each of the structural and relational attributes possessed by **O** applied to the instances in **O**[77]:

$$\sigma(O) = <p^{*k}>_{k\in\{1,...,K\}} = <p^{*1}_{i1},...,p^{*K}_{iK}>^{78},$$

where $p^{*k}_{ik} \in P^{*k}$, and the index $i_k \in \{1,...,|P^{*k}|\}$.

At a certain time, the state of an object $o \in O$ is:

[77]Each attribute in $\mathbf{P}^{*O}$ has a domain which is a superset of **O**. Consequently, in considering the state of **O** (and in this notation), only the application of the active function of each of the attributes possessed by **O** to the objects in **O** is of interest.

[78]The index **ik** should be read $\mathbf{i_k}$.

$\sigma(o) = <p^{*1}_{i1}(o),...,p^{*K}_{iK}(o)>$.

At a later time, the state of **o** may be:

$\sigma'(o) = <p^{*1}_{i'1}(o),...,p^{*K}_{i'K}(o)>$, where $i'_k \neq i_k$ for at least $k \in \{1,...,K\}$.

Consequently, $\sigma(\mathbf{O})$ may also be written:

$\sigma(O) = <\sigma(o)>_{o\in O} = <\sigma(o_1),...,\sigma(o_{|O|})>$.

That is, the state of **O** is the joint state of the elements (objects) of **O**.

Definition 32:

The ***potential (or conceivable) state space***, denoted **Σ(o)**, of each object, $o \in O$, is the Cartesian product of the **codomains** of the structural and relational attributes of **O**. That is,

$\Sigma(o) = D^{P*1}\otimes...D^{P*K}$,

where the **D**'s denote the codomains of the structural and relational attributes possessed[79] by **O**.

By this definition, each member **o** of a category **O** has the same possible state space.

Definition 33:

The ***possible state space***, denoted **Σ(O)**, of a category of objects, **O**, is the set of vectors formed by taking the Cartesian product of the structural and relational attributes of **O**. That is:

$\Sigma(O) = P^{*1}\otimes...\otimes P^{*K}$.

This notion is important, since not all the conceivable states are possible. Attributes represent properties and not all potential functions are actually contained in a property. Therefore, not all potential

[79]Possession of an attribute is directly analogous to possession of a property (see discussion following Definition 5, page 64).

functions are actually contained in an attribute set.

4.4.2.4.2 Object System Change

Two important kinds of change to the state of an object representation are described. First, the notion of state gives rise naturally to the view of an object system event as a state change of an object or set of objects. An event is described by two states - before and after - and involves a change in the active function of one or more attributes. The second kind of change is change to the domain of attributes: that is, to the set of objects possessing an attribute. This latter notion of change is dealt with first.

Our knowledge of the world changes as we become aware of new entities. According to the model of knowledge that was previously described, this "awareness" is expressed as the creation of new instances to represent the existence of these entities. In addition, knowledge of the characteristics of entities is represented by the properties which they possess. In order for an instance to possess properties (and, therefore, to have values for properties), it must be an element of the domain of these properties.

Since objects are intended to represent knowledge about instances of concepts, a mechanism is needed by which the creation of a new instance is recognized through the creation of an object which represents it, with this object being added to the domain of every structural and relational attribute it possesses. Consequently, the notion of *object domain expansion* is defined.

Definition 34:

Let **O** designate a category of objects, **A** a nonempty set of objects such that $A \cap O = \emptyset$, $\mathbf{O}' = O \cup A$, and $\mathbf{P}^* = \{p^*_i | p^*_i : O \rightarrow U\}$ a structural or relational attribute of **O**. An ***object domain expansion*** is a change in the domain of $\mathbf{P}^*$ such that $P^* = \{p^*_i | p^*_i : O' \rightarrow U\}$.

The creation of a new object will be accompanied by a domain expansion for each of the structural and relational attributes it initially possesses. Similarly, since instances may be removed from a cognitive representation, it is necessary to recognize that objects may be removed from an external representation. This is modeled by the notion of *object domain contraction.*

Definition 35:

Let **O** designate a category of objects, **A** a nonempty set of objects such that $A \subseteq O$, $\mathbf{O'} = O\text{-}A$, and $\mathbf{P^*} = \{p^*_i | p^*_i : O \rightarrow U\}$ a structural or relational attribute of **O**. An ***object domain contraction*** is a change in the domain of $\mathbf{P^*}$ such that $P^* = \{p^*_i | p^*_i : O' \rightarrow U\}$.

The remaining kind of change deals with the change of state of existing objects. Object *events* are defined to describe changes in the active function of one or more attributes and, therefore, to describe changes in the state of sets of objects (and individual objects within these sets).

Definition 36:

Let $\bar{p}^* = <p^{*1}_{i1},\ldots,p^{*K}_{iK}>$ denote a vector containing the active functions of **K** structural/relational attributes $\mathbf{P^{*1},\ldots,P^{*K}}$ (with common domain **O**). An ***object system event with respect to O***, denoted $\mathbf{e^*}$, is a change in the active functions of $\mathbf{P^{*1},\ldots,P^{*K}}$, denoted:

$$e^*: <\sigma(O),\sigma'(O)> = <\bar{p}^*,\bar{p}^{*\prime}>,$$

where $\bar{p}^{*\prime} = <p^{*1}_{i'1},\ldots,p^{*K}_{i'K}>$ and $p^{*k}_{i'k} \neq p^{*k}_{ik}$ for at least one $k=1,\ldots,K$.

This definition describes an event in terms of a change in the state of the set of all objects in the domain of any attribute $P^{*1},\ldots,P^{*K}$. However, the interest here is only in changes to objects in $O = O^1 \cap \ldots \cap O^K$, where $\mathbf{O^k}$ is the domain of $\mathbf{P^k}$.

Given the definitions of state and event at the object level, time is given the same meaning as in the concept model (see Section 4.2.2.3.3). Time is dealt with further in Chapter 5 (Section 5.7).

4.4.3 Derived Constructs

4.4.3.1 Behavioral Attribute

A *behavioral attribute* is a representation of a behavioral property. Hence, it consists of constraints on object system events with respect to one or more structural/relational attributes. The allowed changes of state of representations are not arbitrary, but mirror the allowed changes of state of a concept structure (see Definition 17, page 84).

Behavioral attributes describe the allowable events with respect to a set of structural/relational attributes. The following notation is used. $\mathbf{P^{*1},...,P^{*K}}$ denote **K** structural/relational attributes representing structural/relational properties $\mathbf{P^{1},...,P^{K}}$, respectively: that is, $\mathbf{P^{*I}}=\{p^{*i}_{j}|p^{*i}_{j}:O^{i}\rightarrow V^{i}\}$. The following proposition states that there is an identification function which maps the Cartesian product of $P^{1},...,P^{K}$ to the Cartesian product of $P^{*1},...,P^{*K}$.

<u>*Proposition 1*</u>:

There exists an identification function $\mathbf{h}:P^{1}\otimes...\otimes P^{K}\rightarrow P^{*1}\otimes...\otimes P^{*K}$.

The function **h** can be constructed, given Postulates 6 and 7 which assume the existence of structural/relational attributes to represent structural/relational properties.

<u>*Corollary*</u>:

There exists an identification function $\mathbf{h}^{\wp}:\wp(P^{1}\otimes...\otimes P^{K})\rightarrow\wp(P^{*1}\otimes...\otimes P^{*K})$.

Using this information, ***behavioral attribute*** is formally defined as follows.

<u>*Definition 37*</u>:

Let $\mathbf{b}$:$P^1 \otimes ... \otimes P^K \rightarrow \wp(P^1 \otimes ... \otimes P^K)$ denote a behavioral property. A ***behavioral attribute*** *representing* ***b*** is a function $\mathbf{b}^*$:$P^{*1} \otimes ... \otimes P^{*K} \rightarrow \wp(P^{*1} \otimes ... \otimes P^{*K})$ such that for each $x \in \mathbf{P}^1 \otimes ... \otimes \mathbf{P}^K$:

$$b^*(h(x)) = h^{\wp}(b(x)).$$

The essence of this definition is to mirror the definition of behavioral property, pointing out that from a given state of a set of objects, only certain events are allowed.

<u>*Postulate 8*</u>:

For any behavioral property **b**, there exists a behavioral attribute $\mathbf{b}^*$ such that $\mathbf{b}^*$ represents **b**.

In addition, the correspondence between properties and attributes means that any collection of structural/relational attributes has an associated set of behavioral attributes which constrain changes to their active functions.

4.4.3.2 Class

The *class* construct will be defined to mirror the definition of concept. In what follows, **O** denotes a "universe" of objects (i.e., a set of objects representing a universe of instances, **T**). $\mathbf{P}^*_O$ denotes the set of structural/relational attributes possessed by the objects $o \in O$. (i.e., $\mathbf{P}^*_O$ represents $\mathbf{P}_T$). The definition of class relies on the following theorem.

Theorem 4:

Let **C** = <**P**,**B**> = <{P^1,...,P^M},{b^1,...,b^K}> denote a potential concept in **T**. There exists a pair **C*** = <**P***,**B***> = <{P^{*1},...,P^{*M}},{b^{*1},...,b^{*K}}> of sets of structural/relational, and behavioral attributes of **O**, respectively, such that:

for each $P^m \in$ **P**, $\exists$ a unique $P^{*m} \in$ **P*** such that P^{*m} represents P^m,

for each $b^k \in$ **B**, $\exists$ a unique $b^{*k} \in$ **B*** such that b^{*k} represents b^k.

Proof:

The proof of the theorem follows directly from Postulates 6, 7, and 8 since, for each property of **T**, there exists a corresponding attribute representing that property. For any collection of properties **C**, **C*** can be constructed. ■

Since **C*** represents a potential concept, it will be referred to as a *potential class.*

Definition 38:

C* = <**P***,**B***> will be called a ***potential class*** (or *potential object class*) representing the potential concept **C** = <**P**,**B**>.

Proposition 2:

C* satisfies

a) $e(C^*) \neq \emptyset$, and

b) $\nexists$ $P^{*\prime} \in (P^*_O - C^*)$ such that $O' \supseteq e(C^*)$ (where **O**$'$ is the domain of **P**$^{*\prime}$).

Proof:

The potential class **C*** is a direct mapping of the potential concept **C**. Since the stated conditions define **C** as a potential concept, they are preserved in the mapping. ■

Definition 39:

Let $C^* = \{P^{*1},...,P^{*K}\}$[80] denote a potential class. $e(c^*) = C^{*1} \cap ... \cap C^{*K}$ will be referred to as the *extension* of C^*.

Theorem 5:

Two potential classes cannot have the same extension.

Proof:

This follows directly from the proof of Theorem 1 (page 93), given the fact that a potential class is a direct mapping of a potential concept. ■

Given the definition of potential class, the notion of *class structure* can be defined to correspond to concept structure. The definition uses a theorem about the existence of potential classes representing potential concepts. Let **O** be a universe of objects representing a universe of instances **T**, $\mathbf{P}^*_O$ be a set of attributes representing the set of properties possessed by instances of **T** (i.e., $\mathbf{P}_T$), and $\mathbf{CS} = \{C^1,...,C^K\}$ denote a concept structure over **T**.

Theorem 6:

There exists a set of potential classes $\mathbf{CS}^* = \{C^{*1},...,C^{*K}\}$ such that C^{*k} represents C^k, k=1,...,K.

Proof:

From Theorem 4 (page 119), there exists a class representing each concept in **CS**. $\mathbf{CS}^*$ can be constructed to contain the set of such classes. ■

[80]The behavioral attributes are dropped from the notation here, since they are not relevant to the definition.

Definition 40:

$CS^* = \{C^{*1},...,C^{*K}\}$ will be called a ***class structure*** *over* ***O***.

4.4.3.3 Specialization

Class specialization is defined to correspond to concept specialization. The essence is to recognize that certain classes refine others by possessing additional attributes.

Definition 41:

Let C^{*1}, C^{*2} denote two classes in a class structure (each is a set of attributes). Then, C^{*2} is a ***specialization (subclass)*** of C^{*1} (C^{*2} *IS-A* C^{*1}) iff $C^{*1} \subset C^{*2}$.

Theorem 7:

Let C^{*O}, C^{*1}, and C^{*2} denote three classes in a class structure. Then, if C^{*2} is a specialization of both C^{*0} and C^{*1}, $C^{*2} \supset C^{*0} \cup C^{*1}$.

Proof:

Given the fact that classes in a class structure represent concepts in a concept structure, the proof mirrors that of Theorem 2 (page 96). ■

Theorem 8:

Let $e(C^{*1})$, $e(C^{*2})$ denote the extensions of C^{*1} and C^{*2}, respectively, where C^{*2} IS-A C^{*1}. Then, $e(C^{*2}) \subset e(C^{*1})$.

Proof:

Given that $\mathbf{C}^{*1}$ and $\mathbf{C}^{*2}$ are classes in a class structure representing concepts $\mathbf{C}^1$ and $\mathbf{C}^2$, the proof mirrors that of Theorem 3 (page 97). ■

4.4.3.4 Composition

The remaining construct of the object model represents the notion of composite concept. This construct is called a *composite class*. A composite class consists of a number of component classes and, additionally, possesses a number of emergent attributes.

Definition 42:

Let **C** denote a composite concept and $\mathbf{C}^*$ denote a class representing **C**. $\mathbf{C}^*$ is called a ***composite class***.

The existence of a class representing any concept (including composites) follows from Theorem 4 (page 119). The attributes of $\mathbf{C}^*$ are composite attributes representing the composite properties of **C**.

A composite object is an n-tuple containing **n** objects of other classes. Each object corresponds to a component of the corresponding composite instance.

This completes the definition of the major constructs of the MIMIC model. In the next subsection, these constructs are shown to support the creation of *well-defined* representations.

4.4.4 Well-definedness of MIMIC

Proposition 3:

The MIMIC model supports the creation of well-defined representations.

To show this, it is necessary to show that

(a) any *state of interest* of an instance of the concept model (i.e., any state of a concept structure) can be represented by some state of a class structure in MIMIC, and

(b) any *event of interest* in an instance of the concept model (i.e., any change of state of a concept structure) can be represented by a corresponding event in a class structure in MIMIC.

MIMIC supports the correspondences: (1) Instances $\leftrightarrow$ Objects, (2) Properties $\leftrightarrow$ Attributes, and (3) Concepts $\leftrightarrow$ Classes. The correspondences between instances and objects, structural/relational properties and structural/relational attributes, and concepts and classes together allow any state of interest of an instance of the concept model to be represented by a corresponding state of MIMIC. The correspondence between behavioral properties and behavioral attributes allows any event of interest of an instance of the concept model to be represented by a corresponding event of MIMIC.

Appendix 1 presents a detailed example which demonstrates the use of the model in describing three kinds of applications (transaction processing, reasoning, and simulation) in a domain. This both shows the utility of the model, and its suitability as a basis for uniformly modelling knowledge about different kinds of applications. In the next Chapter, a number of contributions and insights into conceptual modelling offered by the MIMIC model are examined. This is followed in Chapter 6 by a comparison to several existing conceptual models.

CHAPTER 5

MODELLING INSIGHTS AND PRESCRIPTIONS

5.1 INTRODUCTION

In the previous chapter, a formal model (MIMIC) was developed for representing knowledge about things and their structural, relational, and behavioral properties. Use of the model is demonstrated in Appendix 1. In this chapter, several contributions of MIMIC to conceptual modelling are discussed in detail and, where applicable, practical modelling guidelines are suggested. Consequently, this chapter complements Chapter 4 by using the constructs developed there to gain some insights into the semantics of conceptual modelling. Subsequently, Chapter 6 compares MIMIC to several well-known approaches to conceptual modelling, demonstrating the representation power and uniformity of the model, as well as its usefulness as a framework for understanding other models.

The following sections discuss problems (or unresolved issues) in conceptual modelling. Each begins by identifying a problem and reviewing its treatment in previous conceptual modelling research. Next, the way in which MIMIC deals with the issue is examined. Consequences of the basic cognitive assumptions of the model, as well as of the chosen formalism, are derived and the resulting implications for understanding and resolving the issue in question are discussed. Finally, where applicable, sections conclude with a prescription or guideline for performing conceptual modelling.

5.2 ON CRITERIA FOR DEFINING A CLASS STRUCTURE

An important task in modelling or representing knowledge is organizing representations of individual things into categories or classes (e.g., Coad and Yourdon, 1991). This, of course, mirrors the importance of categorization to everyday life (Lakoff, 1987; Smith and Medin, 1981). However, very few guidelines exist to assist in this organization, and there do not appear to be any published measures of the *quality*[81] of a class structure. Consequently, it is difficult to determine the value of possible classes such as "the set of persons over 65" or "the set of persons who are both customers and employees".

At one extreme, there is conceivably a class for each individual in the domain. Clearly, this nullifies an important purpose of classification: namely, to abstract what is common about a group of things and, thereby, to provide economy of representation (see Chapter 3). At the other extreme, one monolithic class may be defined to describe all individuals. However, this level of abstraction is too general, and loses knowledge of important distinctions among subsets of the members of this class, since what is common about all things is very little. In practice, people appear to favour intermediate levels of abstraction or classification (Rosch, 1978).

More specifically, the issue considered here is that *existing conceptual modelling approaches do not provide a theoretical basis for deciding on an appropriate level of classification.* This may lead to a proliferation of "classes", many of which do not contribute to the basic function of categorization as a mechanism which abstracts common knowledge about a set of things. Two examples from the object-oriented programming and semantic data modelling literature are used to highlight one important aspect of prior treatment of this issue - the lack of guidelines for determining when to form subclasses of an existing class. In particular, neither requires that subclasses possess additional attributes with respect to superclasses. As will be seen, this is in contrast with the nature of class specialization in MIMIC. In addition, several other necessary criteria for a good class structure in MIMIC are discussed.

In the Smalltalk programming language (Goldberg and Robson, 1989), subclasses can be defined without adding new instance variables and without adding or changing methods. In fact, Goldberg and Robson state that "[n]ew variables *may* be declared and new methods *may* be added by the subclass" (p.58, italics mine), but Smalltalk does not enforce that this must be so. In Smalltalk, subclasses which do not

[81]The quality of a class structure is closely related to the questions of *whether* and *why* some classes are "better" than others.

add instance variables or methods contribute nothing to the organization of knowledge about a domain, since instances of the subclasses do not differ in any structural or behavioral way from instances of superclasses.

In the semantic data modelling area, SDM (Hammer and McLeod, 1981; see also Section 2.3.2 of this thesis) provides a number of guidelines for defining subclasses, each of which is discussed here using examples provided by the authors. The first is by value of an attribute. An example is **MERCHANT_SHIP**, a subclass of **SHIP** for which the value of the attribute **Type** (of SHIP) is "merchant". However, there is no requirement that the subclass be defined in terms of additional attributes. A second basis for defining subclasses in SDM is referred to as user-controllable. Such subclasses, also, are not based on added attributes or common values of attributes. An example is **BANNED_SHIP**, the instances of which are assigned by users[82]. The third criterion for defining subclasses is based on the intersection of two existing subclasses of a third class. The example given is **BANNED_OIL_TANKER**, which is the intersection of **BANNED_SHIP** and **OIL_TANKER**, where both of the latter are subclasses of **SHIP**. SDM also permits subclasses to be defined based on the union or difference of existing classes. In general, these are referred to as *set-operator-defined* subclasses. As with the previous criteria, there is no indication that the subclasses necessarily possess additional attributes. Finally, a class may be defined to consist of those members of an existing class that are currently the value of an attribute of another class. An example is **DANGEROUS_CAPTAIN**, defined as the set containing all members of class **CAPTAIN** which are currently the value of the **Involved_captain** attribute of any instance of the class **INCIDENT** (i.e., **DANGEROUS_CAPTAIN** contains all captains involved in an incident). Again, there is no requirement that the subclass must possess additional attributes.

One notable characteristic of SDM's criteria for subclass definition is that they appear quite

[82]Note that one could assign an attribute **Status** of class **SHIP**, one of the values of which is "banned". In that case, the subclass **BANNED_SHIP** could be derived as in the first example discussed. Therefore, it is not clear how this basis for defining subclasses differs in principle from the first.

arbitrary. That is, there is no evidence that the criteria are, in any sense, complete and others could easily be proposed. For example, a subclass of an existing class could be defined to consist of the instances of the latter having values of an attribute in a certain range. Similarly, subclasses could be defined for which certain conjunctions (or disjunctions) of conditions, such as restrictions on attribute values, are met. In short, there are no guidelines for defining a reasonable number of useful subclasses, since the criteria discussed above are not based on any underlying principles.

The examples show that, without appropriate criteria for defining subclasses, a proliferation of classes may result. That is, if specializations are arbitrarily defined on the basis of the criteria above (and perhaps others), the number of classes may be <u>very</u> large. More importantly, many classes may not add to the structuring and abstraction of knowledge.

To illustrate this, consider a class **SENIOR_CITIZEN**, defined as the set of all **PERSONs** such that **Age** $\geq$ 65. Unless there are *additional attributes* possessed by the set of people satisfying that condition which are not possessed by **PERSONs** in general, the only knowledge captured by defining **SENIOR_CITIZEN** as a class is that the value of the attribute **Birthdate** for its members has a restricted range relative to values for the more general class **PERSON**. This does not in fact constitute additional information, since it is already embedded in the values of the birthdate attribute for the instances of **PERSON**. That is, there is a subset of **PERSON** whose instances have a restricted range of values for **Birthdate**. Consequently, there could also be classes of people with ages greater than one, four, thirty nine, and so on. Without some guidelines or restrictions on the formation of subclasses, an explosion of "classes" of dubious value can be formed. Although this observation may seem obvious, it is one not previously made in discussions of subclass definition.

In order to provide guidelines for deciding whether to define a subclass, it is useful to first draw a distinction between a set of objects and the *extension* of a class. A set of objects can be purely arbitrary (e.g., user controllable subclasses in SDM), while the extension of a class is determined by a common set of attributes (i.e., *intensionally*). Thus, there is an asymmetry in that, while the extension of any class is a set of objects, a set of objects does not necessarily constitute the extension of a class. The view of classes in SDM is extensional in that a class is a set of objects.

A foundation for criteria for defining specialized classes is provided by the basic assumptions of classification theory. The classical theory of concepts and associated theories of semantic memory suggest that specialized concepts are different from more general ones by abstracting extra knowledge (e.g., Keil, 1979; Smith, 1978). This was codified in Chapter 4 as the ***principle of non-redundancy***, and is understood to mean that *the specialization permits additional questions to be answered in relation to the more general concept.* In other words, knowing that an instance belongs to a category implies certain knowledge that may not be specified when the instance is classified, and which cannot be understood simply by knowing that the instance belongs to a more general concept.

To illustrate this notion of abstracting additional knowledge, suppose that EMPLOYEE is a specialization of **PERSON** for which one of the added attributes is **Department**, whose value indicates the department an employee works in. In this case, it makes sense to ask questions such as "Which department does the employee *John* work in?" Clearly, this question might have different answers for different **EMPLOYEEs**. This is not, in general, a sensible question to ask of **PERSONs**, since not all persons are employees, and the question has no meaning for those who are not. On the other hand, suppose that a specialized concept is defined to contain only the instances of a more general concept having a certain value or range of values for one or more attributes. For example, one might think of **ADULT** as the specialization of **PERSON** for which the value of **Age** is greater than 18 years. By itself, this permits no additional questions to be answered with respect to the more general concept of **PERSON**, since the age of a person can be determined without reference to a "specialization" such as **ADULT**.

Nevertheless, most people would agree that a concept such as **ADULT** is quite useful. However, according to the assumption stated earlier the usefulness stems, not from the knowledge that the age of such individuals is greater than 18 years (or some other arbitrary number), but from the fact that society bestows on these persons additional properties. Hence, if *John* is an instance of ADULT, it makes sense (depending on the society) to ask questions such as "Does *John* have a driver's license?", "Which party does *John* intend to vote for in the upcoming election?", "To whom is *John* married?", and so on[83]. In other words, the essence of adulthood is that some persons possess added properties with respect to other persons. While the instances of ADULT do have a restricted range of values for the attribute **Age** (or **Birthdate**), this alone does not permit additional questions to be answered (or inferences made).

To consider a more complex example, suppose that males and females in some country retire at different ages. To define a subclass of **PERSON** called **RETIRED_PERSON**, one might specify a compound condition such as "**Sex**=Male and **Age**≥65 or **Sex**=Female and **Age**≥60". Although this condition is a sufficient test to identify a person as retired, it misses the fundamental meaning of "retiredness". Retired people differ from people in general since they possess additional properties (e.g., **Company_pension**), and may not be permitted to possess properties that some other persons possess (e.g., **Employer**).

These examples show that there are complementary roles of added properties and constraints on property values in the definition of subclasses. The characterization of class membership by values of an attribute may play an important role for humans when it comes to classifying new instances in the absence of complete information. In fact, values of one attribute may often be the reason for acquiring certain other attributes (as in ADULT). Hence, this may serve as a useful *identification procedure* (Armstrong et al., 1983). For example, an individual can be classified as a SENIOR_CITIZEN provided that we know that the person was born prior to a certain date. In addition, we can infer that the person possesses the added properties of SENIOR_CITIZEN (e.g., **Government_pension**) even if the values of these additional

[83]In fact, there may be some difficulty in defining a minimum common age for which all these properties are defined, since minimum legal driving, voting, and marrying ages may not be identical.

properties are unknown at the time of classification. In fact, knowledge about a person's birth date may motivate an effort to determine the values of the added attributes. To further illustrate the issue, consider a class called **ORDER**, and the set of "all orders valued over $500". This set of objects does not automatically constitute a class. However, orders meeting this condition may be subject to discounts or special payment terms. These additional attributes can be used to define a subclass. Furthermore, the range of values of the **Amount** attribute for members of the subclass is restricted to the subset of values greater than $500. This identifies an order as belonging to the subclass.

The discussion earlier indicated that subclass definition has been handled in largely an arbitrary fashion in existing modelling approaches. It will now be shown that MIMIC imposes a clear minimal criterion (i.e., a necessary condition) for defining subclasses that is consistent with abstracting added knowledge in the manner just described and, therefore, is consistent with the model as a formalization of the classical theory of concepts. In addition, further criteria for a good class structure are reviewed.

In MIMIC, a class structure **CS** contains a set of classes $\{C^1,...,C^K\}$[84]. Each class is intensionally defined as a set of structural/relational attributes, and satisfies the condition that $\mathbf{C}^k$ does not equal the union of the attributes of any other subset of classes in **CS**. It follows from this that a subclass must possess at least one additional structural or relational attribute with respect to union of the sets of structural and relational attributes of its superclasses (Theorem 7, Chapter 4). That is, letting **P**, **P′** denote sets of attributes a class $C' = P'$ is a subclass of $\mathbf{C} = P$ *only if* $P \subset P'$ (Definition 41, Chapter 4). *It is these additional attributes which, according to the model, permit a subclass to be defined.* This restriction is not enforced in other treatments of class specialization. In addition, this minimal criterion is consistent with the earlier description of what specialization means. That is, a subclass always provides the basis for answering additional questions beyond those which can be answered with respect to all superclasses.

[84]The "*" notation (e.g., CS^*) used in the previous chapter to distinguish the cognitive and object levels has been dropped for simplicity, since the focus in this chapter is on one level only - the object level.

MIMIC's view of specialization can prevent a needless proliferation of classes. Although many classes can still be formed from (random) permutations of attributes, the upper bound will be far smaller than that possible under the looser conditions contained in modelling approaches such as SDM. In addition, subclasses possess additional meaning when defined by added attributes, in that the class definitions abstract further commonality of a subset of the extension of one or more superclasses.

A practical methodological implication for systems analysis (conceptual modelling) emerges from this view. Attempts by users to specify specialized categories of things based on, say, values of attributes[85], can be followed up by asking "why?". Such probing would be aimed at eliciting additional structural, relational, and behavioral attributes associated with the subcategories. Inability to do so should lead to a re-examination of the need for the specialization. This constitutes a clear guideline for modelling which is derived directly from the foundations of the model. Furthermore, if a class is claimed to be a specialization of two or more others, the analyst should identify what additional attributes distinguish it with respect to the union of the attributes of all superclasses. If none can be identified, the subclass is not a good abstraction.

A Case Against Behavior-based Subclasses

The MIMIC model does not permit the definition of subclasses based solely on added behavioral attributes. Recall that a class **C** = **P**, where **P** is a set of structural/relational attributes, can also be specified **C** = <P,B>, where **B** contains all behavioral attributes constraining changes to any attribute in **P**. On first glance, one might argue that for a class **C=<P,B>**, a subclass **C′=<P,B′>** might well be defined, possessing only additional behavioral attributes (i.e., $\mathbf{B}\subset\mathbf{B}'$).

However, according to the MIMIC formalism, the extension of a class is determined completely

[85]This applies also to attempts to specify categories based on other criteria, such as set operations (intersection/union/difference) on two or more existing categories.

by the intersection of the domains of its structural and relational attributes, as are the behavioral attributes which constrain changes to the active functions of any structural/relational attribute. Consequently, two classes cannot differ only on the basis of behavioral attributes. Even if behavioral constraints were not "implied" by the set of structural/relational attributes defining a class, since they are defined on domains of Cartesian products of structural and relational attributes (Definition 37, Chapter 4), *the addition of behavioral attributes could not restrict the extension of a class.* Since a basic outcome of class specialization is that a subclass contains strictly fewer members than each of its superclasses (Theorem 8, Chapter 4), behavioral attributes alone are not a basis for class specialization.

This result is counter-intuitive. For example, one might argue that **MANAGER** is a subclass of **EMPLOYEE** for which changes to the value of attribute **Salary** can exceed 10% (presumably for other employees it cannot)[86]. However, the model prescribes that such a class can be defined *only if* there are additional structural and/or relational attributes (e.g., **Manages** other employees). Thus, while added behavioral attributes may accompany a subclass definition[87], they are not sufficient for defining a subclass.

In a similar vein, one might argue that **EMPLOYEE** can be specialized as follows. **EMPLOYEEs** with **Salary** < \$30000 receive 3% raises annually, whereas all other employees receive no raises (as part of a pay equity plan). This appears to specialize **EMPLOYEE** into two mutually exclusive subclasses based on value of **Salary**, with only behavior differing between the two classes. On closer examination, however, it is clear that the information contained in such a distinction is captured by a single behavioral attribute - one which permits increases in salary (i.e., changes in the active function of **Salary** which result in increases for one or more **EMPLOYEEs**) only for employees with Salary<30000, given any active

[86]This could be interpreted to mean that **EMPLOYEE** has a behavioral attribute **b**:Salary$\rightarrow \wp$(Salary), while **MANAGER** has a behavioral attribute **b′**:Salary$\rightarrow \wp$(Salary).

[87]This could mean that **MANAGER** possesses the behavioral attribute **b′**:Salary$\otimes$Manages$\rightarrow \wp$(Salary$\otimes$Manages).

function of **Salary**. Note that it is the ***state*** of **EMPLOYEEs** with respect to **Salary** which determines whether a raise may be given. Furthermore, all questions of interest about *allowed **behavior** with respect to **Salary*** can be answered by knowing the ***state*** of an EMPLOYEE with respect to **Salary.**

In MIMIC, a behavioral attribute specifies allowed events given active functions of structural/relational attributes. Consequently, the restriction on raises can be modelled as a behavioral attribute, **b**:Salary$\rightarrow \wp$(Salary). That is, if $\mathbf{s_1}$ is the active function of **Salary**, then an event $<s_1,s_2>$ is allowed (i.e., $s_2 \in b(s_1)$) only if, for each $e \in$ EMPLOYEE, $s_1(e)<30000 \Rightarrow s_2(e)=1.03*s_1(e)$ and $s_1(e) \geq 30000 \Rightarrow s_2(e)=s_1(e)$. This attribute captures differences in behavior based on the current state of the instances of a class. Since the distinction is naturally captured by modelling **Salary** as a set of functions, this constitutes a useful additional benefit of defining structural and relational attributes as sets of functions, rather than single functions.

In this way, a precise meaning is given to what otherwise might be a loosely- or even ill-defined phrase such as "behavioral differences". The general notion of having different behavior is imprecise since it may mean either that allowed changes of state of objects in a class differ depending on the current state of objects, or that objects behave differently by virtue of possessing different structural and relational attributes. In MIMIC, differences in behavior based on the current state of instances do not constitute different behavioral attributes, thereby providing a more precise view of behavioral differences.

The MIMIC model suggests that structural and relational attributes are more fundamental than behavioral attributes as a basis for classification and class specialization, since behavioral constraints are only implicitly contained in class definitions. This conclusion emerges from the chosen formalism (using the cognitive assumption that subclasses contain more information), and is not recognized in existing theories of concepts (e.g., see Smith and Medin, 1981; Lakoff, 1987). In these theories, the notion of classification by common behavior is not clearly specified, perhaps due to a lack of formalization in specifying what properties are. In MIMIC, classes may be specialized only by considering additional structural and relational attributes. Since this restriction emerges from the formalism chosen (and not from an assumption of classical concept theory), it may be that another formalism (specifically, a different definition of behavior) might permit subclasses to be defined based solely on behavioral differences.

The inadequacy of behavioral attributes for class specialization again offers a clear modelling guideline or prescription. Specifically, if a user informally describes a specialized category which is characterized only by added behavioral attributes, the analyst should try to identify further structural/relational attributes possessed by each subset of objects which "behaves differently". Such probing would be aimed at eliciting additional structural and relational attributes which motivate the specialization. If further attributes do not exist, the differences in behavior resulting from different states of instances of a class can be modelled as a behavioral attribute on the existing class (as in the example earlier of allowed salary increments being based on current salary).

This constitutes a useful mechanism to aid in defining a class structure based on a cognitive model. However, the model contains several other necessary conditions for a class structure to be considered "good".

The first is that a potential class abstracts the set of all attributes common to its instances (*principle of maximal abstraction*). This means, for instance, that a class **EMPLOYEE** is not defined to contain only the attribute **Works_in_department**, although this may be a condition true only of **EMPLOYEE**s. The point is that people who are employees also possess numerous other attributes, such as **Birthdate** and **Supervisor**. Some of these are possessed by virtue of the fact that **EMPLOYEE**s are **PERSON**s (i.e., inheritance), but others, such as **Supervisor**, may simply have the same or a larger domain as **Works_in_department**. The principle of maximal abstraction states that all such attributes are to be included in a class definition. This point does not appear to be considered in other conceptual models.

The second condition is that a class has a non-empty extension (*principle of abstraction from instances*). Although this has not been explicitly considered by other models, this may simply be because it is such an obvious consideration. It is mentioned here merely to emphasize that classes are defined to reflect the premise that concepts are abstracted from instances.

The third criterion deals with the nature of a class structure for representing some "universe" of knowledge (*principle of completeness*). Specifically, all relevant properties which characterize the universe of knowledge and, therefore, which are part of any concept structure on that knowledge, should be contained in the class structure representing that knowledge. This means that every property is represented by an attribute attached to some class.

These criteria are interesting because they allow for multiple *views* of the world: that is, multiple class structures which satisfy the other principles (this is *principle of variety*). This is discussed further in the next section.

To summarize, some necessary criteria for defining a class structure have been provided, with particular attention paid to a necessary condition for defining subclasses. That is, a subclass must possess at least one structural/relational attribute beyond the union of the attributes of its superclasses. This leads to a clear modelling guideline for analysts in eliciting a classification structure for a domain: namely, *search for added structural/relational attributes* if a user specifies subclasses based on:

a) values of an attribute of an existing class,

b) intersection (union, difference) of the extension of two or more existing classes,

c) added behavioral attributes.

This recommendation indicates the importance of structural and relational attributes for class specialization. While knowledge about things includes structure, relationships, and behavior, the last element alone is not a basis for developing category refinements and, if present, must be accompanied by further structural or relational attributes.

In the next section, the contribution of the model to resolving the debate over single versus multiple inheritance is discussed.

5.3 ON CLASS STRUCTURE - HIERARCHY VERSUS LATTICE

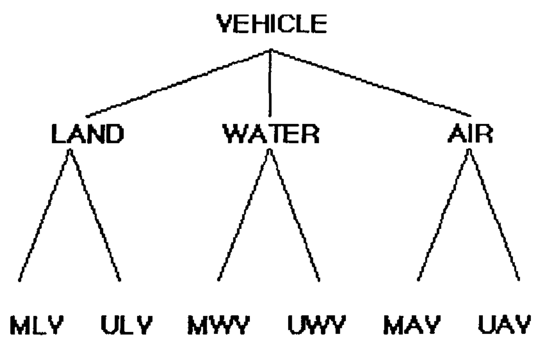
There are varying positions in both the AI and object-oriented literatures about whether models and systems should support a *hierarchical* class structure, in which a class has a single immediate superclass, or a more general *lattice* structure, in which a class may have several immediate superclasses (e.g., Goldberg and Robson, 1989; Stefik and Bobrow, 1986; Fikes and Kehler, 1985). An important consequence of this choice is the nature of the inheritance of attributes from superclasses to subclasses. Specifically, a hierarchical structure supports only single inheritance, while a lattice structure supports multiple inheritance.

The issue of a hierarchical versus lattice class structure has important ramifications for conceptual modelling, since it may affect the number and nature of classes used to model a domain. In this section, the problem is examined and the MIMIC model is used to justify the "naturalness" of a lattice structure for organizing classes. First, though, two examples from the programming language domain are used to illustrate differing approaches to class specialization.

In Smalltalk (Goldberg and Robson, 1989), classes are organized in a strict hierarchy. That is, each subclass has a single immediate superclass. Inheritance is single. In such a system, the class structure of Figure 5-1 might be used to define a group of subclasses of **VEHICLE**, where **Medium** and **Motorization** are important criteria for distinguishing classes[88].

[88]Note that the subclasses must also possess additional structural/relational attributes, and that the characterization by **Medium** and **Motorization** is merely shorthand for this. In addition, this is only intended to be an illustrative example, and the class lattice could well be organized differently.

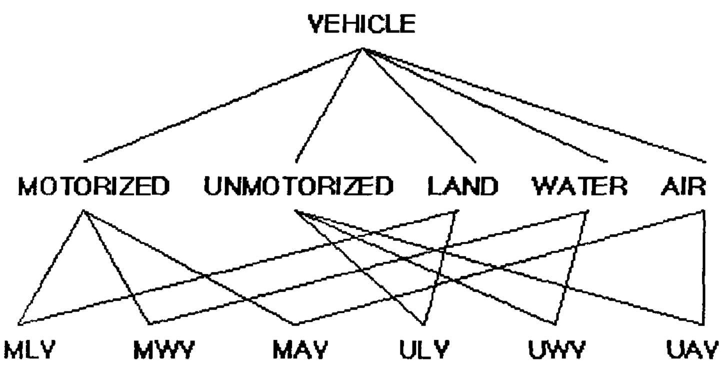
Figure 5-1: A Hierarchical Class Structure



Here, it is not clear why **Medium** (i.e., land, water, air) is used to derive the first "level" of specialization, and **Motorization** (i.e., motorized, unmotorized) to derive the second, since these dimensions are orthogonal. The structure above implies a decision about "importance" that appears to be unjustified: namely, that **Medium** is more "fundamental" than **Motorization**. In addition, there are no explicit classes of **MOTORIZED** and **UNMOTORIZED** vehicles, although these might well be of interest for an application.

By contrast, a system such as LOOPS (Stefik and Bobrow, 1985), classes are organized in a lattice. Under this approach, the example above might have the class structure of Figure 5-2:

Figure 5-2: A Lattice Class Structure



Unlike the previous example, this recognizes a class of all **MOTORIZED** vehicles (as well as **UNMOTORIZED**)[89]. This cannot be done in a strict hierarchical system without choosing **Motorization** over **Medium** as the primary basis for subclass definition, in which case the classes of **LAND, AIR**, and **WATER** vehicles are lost. Consequently, a lattice appears to be a more flexible mechanism for organizing classes. Further indication of this in the context of MIMIC is given later.

The same issue has been raised in the semantic and object-oriented data model areas (e.g., Banerjee et al., 1987; Fishman et al., 1987; Hammer and McLeod, 1981). Most models support a class lattice with multiple inheritance.

The approach taken in dealing with classification in MIMIC is derived from the *principle of abstraction from instances* - the notion that concepts are defined based on shared properties of instances. However, there is no reason why the sharing of properties must be hierarchical (Smith, 1978; Quillian, 1968). In line with the theory of concepts in which the model is grounded, the basic guideline governing the specialization of classes in MIMIC is that a subclass must possess added attributes which permit additional questions to be answered about objects. Since classes are defined as collections of attributes, attributes can be combined in different ways to yield different class structures, subject to the criteria discussed in the previous section. Furthermore, there is nothing to prevent a class from being derived from two (or more) existing classes provided that it possesses added attributes with respect to those of all superclasses. This is a point which is not explicitly addressed in theories of concepts and semantic memory, but emerges from the attribute-based definition of class in MIMIC.

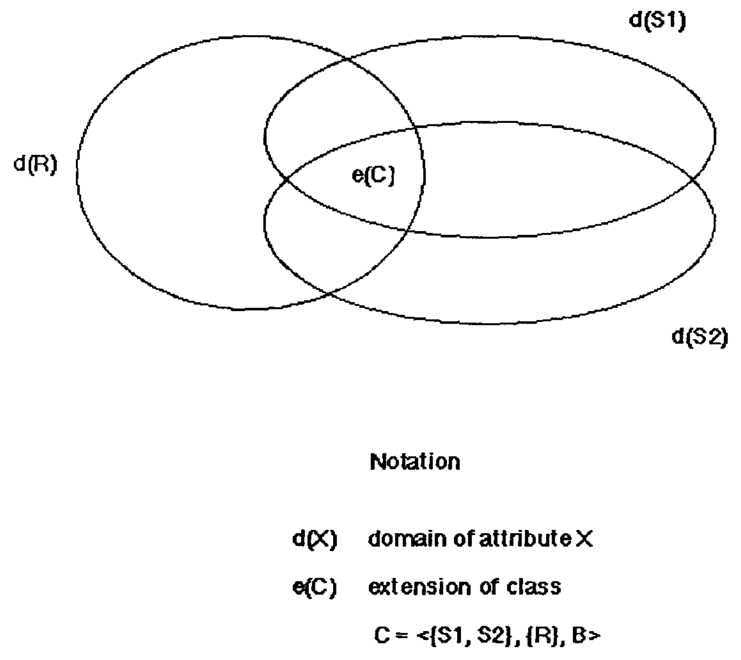
MIMIC defines a class as a set, **P**, of structural and relational attributes. If class **C′** = P′ (or <P′,B′>, recognizing the associated behavioral constraints) is a subclass of **C** = P (or <P,B>), then $P \subset P'$ and $B \subseteq B'$, and **C′** is said to ***inherit*** the attributes of **C**. That is, instances of **C′** possess (i.e., are in the domain of) all the attributes of **C** plus some additional ones. Also, an instance of **C′** is an instance of **C**.

[89]These classes may or may not be disjoint.

Now, let $\mathbf{C}''$ = P'' (or $<P'',B''>$ denote a distinct subclass of **C**, (i.e. $P \subseteq (P' \cap P'') \subset (P' \cup P'')$. Then, by the *principle of non-redundancy* a class $\mathbf{M}$ = $<P^M,B^M>$ is a subclass of both $\mathbf{C}'$ and $\mathbf{C}''$ if and only if $P' \cup P'' \subset P^M$ (implying $B' \cup B'' \subseteq B^M$). That is, **M** contains all the attributes of $\mathbf{C}'$ <u>and</u> $\mathbf{C}''$, plus some additional ones, and is said to *inherit multiply* from $\mathbf{C}'$ and $\mathbf{C}''$.

This example illustrates multiple inheritance from an *intensional* point of view. However, multiple inheritance also has an *extensional* interpretation. Each attribute is defined over a domain of objects. An object will typically be in the domain of many structural and relational attributes. By considering various combinations of attributes, multiple class structures can be formed. The extension of a class is determined by the intersection of the domains of its structural and relational attributes, as shown in Figure 5-3.

Figure 5-3: Extension of a Class



Now, for any two classes $\mathbf{C}^1$ and $\mathbf{C}^2$ with extensions $e(C^1)$ and $e(C^2)$, respectively, there is a (possibly empty) set of objects, $e(M) = e(C^1) \cap e(C^2)$. If $e(M)$ is non-empty, it may denote the extension of a class **M** only if there exists at least one structural or relational attribute $\mathbf{P^M}$ with domain $\mathbf{d(P^M)}$, such that $\mathbf{P^M}$ is not an attribute of either $\mathbf{C^1}$ or $\mathbf{C^2}$ and $e(M) \subseteq d(P^M)$. Thus, class specialization is purely attribute-based.

From the model, objects may be interpreted as having "roles" by their membership in several classes (Richardson and Schwarz, 1991; Pernici, 1990; Sciore, 1989). An object may simultaneously belong to several classes which are not in a single specialization path[90] (i.e., no class is a subclass of any of the others in question). For example, a vehicle is both a **MOTORIZED** and a **LAND** vehicle. This allows one to consider only the attributes which are relevant for a particular application. The model thereby supports various "views" of an object corresponding to its membership in several classes.

There is also a second way in which MIMIC supports various views of an object. The *principle of variety* states that, for a given universe of objects and attributes, there may be many class structures which satisfy the other principles of conceptualization. For example, different individuals (or departments in an organization) may divide the same world (objects and attributes) into distinct class structures best suited to their needs. Hence, the model supports, not only different views of an object <u>within</u> a class structure, but also different class structures for a given set of objects and attributes.

Returning to the example of Figure 5-2, it is now clear that there is extra information in this class structure with respect to that of Figure 5-1. The sets of classes {LAND, WATER, AIR} and {MOTORIZED, UNMOTORIZED} exhaust (and possibly partition) VEHICLE along different dimensions. Consequently, any subclass beyond this level <u>must</u> be a subclass of at least (and possibly exactly) one class from {LAND, WATER, AIR} and at least (possibly exactly) one from {MOTORIZED, UNMOTORIZED}. If either of these sets of classes partitions VEHICLE, a class at the third level is a

[90]A *specialization path* is a collection of IS-A edges linking class nodes A,B,...,K such that A IS-A B IS-A ... IS-A K. An object always belongs to each of the classes in a specialization path.

subclass of exactly one class from that set. This is captured in MIMIC by virtue of the fact that if, say, {MOTORIZED, UNMOTORIZED} partitions VEHICLE, then the intersection of the shared domain of the added attributes defining MOTORIZED with the shared domain of the added attributes defining UNMOTORIZED is the *empty set.* Therefore, any subclass of MOTORIZED cannot also be a subclass of UNMOTORIZED (and vice versa) since MOTORIZED and UNMOTORIZED share no instances.

This approach to class specialization (and multiple inheritance) gives rise to another potential guideline for modelling, which may be termed an *instance and attribute search* strategy. The analyst can work with the user(s) to identify sets of objects and the structural and relational attributes which apply to them. For example, specific objects could be compared to ascertain their common attributes. A provisional decision could be made to view these attributes as the definition of a class. Subsequently, further instances could be classified and, if necessary, attributes added or removed from the provisional class definition. At that point, behavioral attributes can be determined based on the structural and relational attributes defining a class. This may, in turn, uncover structural/relational attributes which were originally missed. Such a strategy could complement one of first specifying classes and then eliciting attributes for these classes. Motivating this guideline is an assumption that shared attributes are the fundamental basis for classification and can support different class structures, which may change over time as needed. As a result, identifying attributes first may be more "stable" than identifying classes first.

The class structures which are possible depend on which of two assumptions is made about the added attributes of subclasses in a multiple inheritance context. One assumption is that a subclass possesses added attributes with respect to the attributes of each of its superclasses. The more restrictive assumption, and the one enforced by MIMIC, is that a subclass must possess added attributes with respect to the union of the attributes of all its superclasses (principle of non-redundancy). The consequences of these assumptions in terms of their impact on the possible organization of a class lattice are now examined.

The issue here is best illustrated with a simple example. Consider a "world" of objects, **O**, and the structural/relational attributes $\mathbf{P^1}$, $\mathbf{P^2}$, and $\mathbf{P^3}$ which are possessed by any of the objects in **O**. Assume that the potential classes are:

$C^1 = \{P^1\}$
$C^2 = \{P^2\}$
$C^3 = \{P^3\}$
$C^4 = \{P^1, P^2\}$
$C^5 = \{P^1, P^3\}$
$C^6 = \{P^2, P^3\}$
$C^7 = \{P^1, P^2, P^3\}$

Under the first assumption, any combination of these classes is allowed. However, under the second, only certain class structures are possible. One class structure over **O** is $CS^A=\{C^1,C^2,C^3\}$: that is, classes containing only a single attribute. Given this class structure, no subclasses can be defined, since no combination of $\mathbf{P^1}$, $\mathbf{P^2}$, and $\mathbf{P^3}$ permits any additional questions to be answered with respect to the initial classes. However, if, for example, $\mathbf{C^1}$ and $\mathbf{C^2}$ are the only classes defined to contain a single property, then the subclasses $\mathbf{C^5}$ (an immediate subclass of $\mathbf{C^1}$) and $\mathbf{C^6}$ (an immediate subclass of $\mathbf{C^2}$) may be included, since each contains an added attribute with respect to its single superclass. This means that $CS^B=\{C^1,C^2,C^5,C^6\}$ is another class structure over **O**. Alternatively, $\mathbf{C^7}$ may be defined as a subclass of both $\mathbf{C^1}$ and $\mathbf{C^2}$, since it has an additional attribute with respect to the union of the attributes of $\mathbf{C^1}$ and $\mathbf{C^2}$ (i.e., with respect to $\{P^1,P^2\}$). Therefore, $CS^C=\{C^1,C^2,C^7\}$ is yet a third class structure over **O**. The point is that a class structure limits the classes which satisfy the principles of *non-redundancy* and *completeness*.

This simple example indicates the dominant role of attributes and the notion of a class structure in determining how subclasses may be formed. Various class structures are possible, with the possible subclasses depending on the classes which are defined at the highest level of a structure (i.e., those with fewest attributes). No complete guidelines on the grouping of attributes to define classes can be offered here. However, reasonable guidelines and restrictions on arbitrariness have been suggested based on some fundamental assumptions about the nature of concepts and the importance of categorization (specifically, the sharing of attributes in a non-hierarchical manner).

In summary, MIMIC classes are formed as groupings of attributes. Class organization in the general case can be expressed as a directed acyclic graph. The model provides an attribute-based approach to class definition and specialization/inheritance.

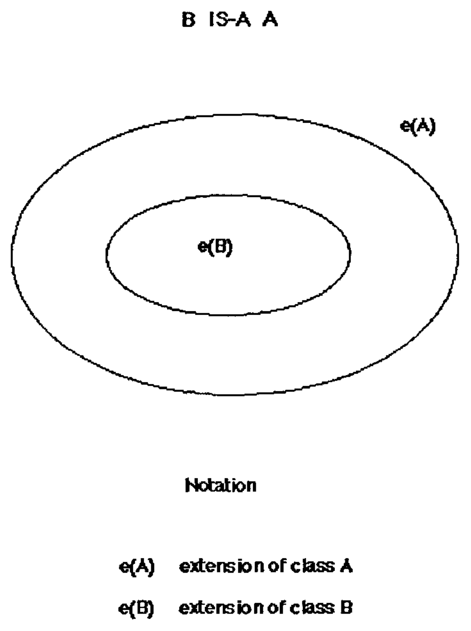
5.4 ON THE MEANINGS OF "ISA"

There is considerable ambiguity in research in the AI and database communities about the use of IS-A relationships (also called subset hierarchies, generalization hierarchies, etc.) to capture world knowledge. This is most evident in a paper by Brachman (1983), which describes a number of different ways in which the label IS-A has been used[91].

The general meaning of IS-A may be conveyed as follows. Let **e(A)**, **e(B)** designate the extension of classes **A** and **B**. Then, **B** *IS-A* **A** iff every member of **B** is also a member of A (e.g., Brachman, 1983; Mylopoulos et al., 1980). That is, **e(B)** is a proper subset of **e(A)**. Figure 5-4 illustrates this with a Venn diagram.

[91]This discussion focuses strictly on IS-A relationships between classes of objects and not on links between objects and classes. For example, statements such as "STUDENT *IS-A* PERSON" are of interest here, while those such as "*John IS-A* STUDENT" are not. The latter deal with the possession of attributes by objects.

Figure 5-4: Extensional Meaning of IS-A



Beyond this, the single term IS-A, as used in conceptual and semantic data modelling (e.g., Mylopoulos et al., 1980; Hammer and McLeod, 1981) to indicate that the extension of one class is a strict subset of the extension of another class, is *semantically overloaded*, since one symbol is used to describe several kinds of subclass linkages with important differences in meaning, as shown next.

In MIMIC, classes are defined based on shared attributes. In addition, it is recognized that some properties may be acquired and lost over the lifetime of an object. This fact is used to derive four refined meanings of IS-A based on the temporal or permanent nature of the additional attributes defining a subclass[92].

[92]Although the representation level (i.e., MIMIC model) is being discussed here, everything that is said applies also to the conceptual level due to the mapping between the levels discussed in Chapter 4.

The cognitive premise on which the proposed refinements to IS-A are based is that instances (representations of individual things in the world) may *acquire* and *lose* properties through domain expansion and contraction (Definitions 14 and 15, Chapter 4). This is reflected directly in MIMIC in that objects may acquire and lose attributes, thereby moving between classes. However, not all attributes may be acquired/lost. Four cases exhaust the possibilities in this regard.

First, membership in a (sub)class may be *always*. This means that an object, when created, belongs to and remains in the class in question as long as it exists. An example is **PERSON**. No objects (e.g., from a superclass **ANIMAL**) become persons subsequent to their creation, and a person is always a person while in existence. In this case, the added structural/relational attributes are permanent[93]. That is, an object is assigned to (i.e., becomes a member of the domain of) these attributes at the time it is created, and continues to possess these attributes as long as it exists. **Height**, **Weight**, and **Birthdate**, are examples of permanent attributes of class **PERSON**.

Second, subclass membership may be *becoming*. This means that an object starts out in a superclass only, but eventually becomes (or may become) a member of a subclass and remains in that subclass as long as it continues to exist. An example is **PARENT**, since a **PERSON** is not a parent at birth, but if he/she becomes a parent, he/she remains so. Another example is **ADULT**[94]. In this case, an object in some class may acquire (i.e., enter the domain of) certain attributes at some point subsequent to its creation. However, once these attributes are possessed by the object, they are retained for the duration of its existence. For both **PARENT** and **ADULT**, an instance of **PERSON** becomes an instance of the subclass (i.e., **PARENT** or **ADULT**) if or when it acquires the attributes distinguishing the subclass

[93]This does not mean that these attributes are unchanging, since the values (active functions) may change over time.

[94]There is a slight difference between **PARENT** and **ADULT**, since all **PERSON**s become **ADULT**s, but not all **PERSON**s become **PARENT**s. In other words, all **PERSON**s must assume the added attributes of **ADULT**, but need not assume the added attributes (i.e., become) **PARENT**s.

(e.g., **Has_children** for **PARENT**). Once these attributes are acquired, however, they are retained permanently.

Third, subclass membership may be *ceasing*. This means that an object starts its existence belonging to a subclass, but at some point ceases to be a member of the subclass, and stays out (but in some superclass of that class) for the duration of its existence. An example is **CHILD**. Every **PERSON** is initially a **CHILD**. However, at some point the person ceases to be a child and never becomes a child again. In this case, an object is created with certain attributes, but loses some of these attributes at some point during its existence. Once these attributes are lost, they are not re-acquired. For example, every **PERSON** has the added structural/relational attributes of **CHILD** when born (e.g., **Legal_guardian**), but loses some of these (i.e., the attributes which distinguish **CHILD** from **PERSON**) at some point, thereby losing membership in the class **CHILD**.

Fourth, objects may *happen to be* in a particular subclass. This means that an object is created belonging to some class, but may belong and then not belong to the subclass in question several times during its existence. Examples include **CUSTOMER** and **EMPLOYEE**. **PERSON**s may move in and out of these classes several times over the course of their lives. In this case, a member of a superclass may acquire and lose the added attributes of the subclass (perhaps several times) during its existence. For example, **CUSTOMER** is based on a relational attribute **Customer_of** with **COMPANY**, and **EMPLOYEE** is based on a relational attribute **Employed-by** with **COMPANY**.

This analysis reinforces the view that a single notion of IS-A is inadequate to capture the various kinds of subclass derivations that are possible when the temporal nature of class membership is taken into account. The definition of subclasses in the MIMIC model is based on added structural/relational attributes and the assumption that attributes can be acquired and lost by objects. This provides a framework for explaining and understanding the semantics of four forms of subclasses, since the cases can be distinguished by the permanence of the added attributes of a subclass. It additionally recognizes that an object may move between classes over its lifetime, an issue not dealt with by many conceptual models[95]. However, the modelling mechanisms for structural and relational attributes do not inherently indicate to which of the four kinds an attribute belongs. Consequently, the specification of whether an attribute is permanent, becoming, and so on, must be captured externally.

A modelling guideline emerging from this analysis is that, when attributes are specified by a user, these should be determined to be permanent, acquirable only, losable only, or both acquirable and losable. This captures important information about whether objects may move between classes in a lattice.

5.5 ON ASSOCIATIONS - RELATIONAL ATTRIBUTE VERSUS COMPOSITE CLASS

An important element of knowledge about a thing involves the associations (relationships) the thing has with other things. In some cases, an association of two or more things may have properties of its own, in which case it may be more valuable to regard the association (or composition) as a separate thing. The issue is whether relationships with properties should be modelled differently from relationships without properties.

In the entity-relationship model (Chen, 1976), the relationship construct is used to model both associations with attributes of interest and those without attributes of interest. However, these two kinds of associations are not formally distinguished. In some cases, it is difficult to decide whether something should be modelled as an entity or as a relationship. Even if relationships with attributes are modelled as entities, information about the allowed changes in the values of attributes of these associations cannot be represented since the E-R model does not capture behavioral knowledge,.

[95]There are some additional considerations here, since the acquisition or loss of attributes is a kind of change. In particular, if these attributes are relational, acquisition of an attribute depends on the existence (or creation) of certain objects in other classes (e.g., acquiring **Parent_of** accompanies the creation of a new object of class **CHILD**). This is effectively a constraint on change, although not at the level of changes to the active functions of attributes. Incorporating such information is left as a subject for further work.

In most modelling approaches, no distinction is made between structural and relational attributes. All are modelled as binary associations between entities or objects (e.g., Mylopoulos et al., 1990; Verheijen and Van Bekkum, 1982; Abrial, 1974). Such models have no notion of composition in the sense of emergent attributes (Section 4.2.3.4). Instead, composition is often equated with *Cartesian aggregation* (Hull and King, 1987; Smith and Smith, 1977), in which an object is considered as composed of its attributes (e.g., **PERSON** is composed of attributes such as **Height** and **Weight**). Composition is recognized in some object-oriented programming languages such as Loops (e.g., Stefik and Bobrow, 1986), and in data models such as ORION (e.g., Banerjee et al., 1987). However, these do not provide clear guidelines as to whether or not certain knowledge should be modelled by composite objects, nor is it clear whether they enforce that a composite object must possess emergent attributes.

The cognitive assumption underlying MIMIC's treatment of composition is that part of knowledge includes knowing how things are linked to other things (Smith, 1978; Miller and Johnson-Laird, 1976). Furthermore, components can be considered either as distinct things or as *parts* of more complex things.

The basic mechanism for expressing associations among objects in MIMIC is the *relational attribute*. A *simple* relational attribute **R** is a set of functions mapping from a set of objects $\mathbf{T}^1$ to a subset of the power set of the Cartesian product of several other sets of objects $\mathbf{T}^2,...,\mathbf{T}^N$ ($N \geq 2$): that is, ($R=\{r_i | r_i : T^1 \rightarrow Q\}$), where $Q \subseteq \wp(T^2 \otimes ... \otimes T^N)$). Relational attributes link an object to one or more other objects without ascribing any added information to this association. However, the model also provides a mechanism for representing associations which can be viewed as objects in their own right. Such associations possess *emergent* (or composite) attributes: that is, attributes associated with the collection of objects in the association. *Composite objects* model these associations and composite object classes represent the kind of association.

A *composite object* is a concatenation of other objects of other classes, denoted $(t^1,t^2,...,t^N)$ $(t^i \in T^i)$, which possesses *emergent (or composite) structural/relational attributes*. The *emergent structural* *attributes* forming part of the definition of a composite class are of the form $S:H\rightarrow V$, where **V** denotes a set of values, **H** denotes a subset of $T^1\otimes...\otimes T^N$ and is a domain of composite objects, and $T^1,...,T^N$ are the extensions of the component classes of the composite. Similarly, the *emergent relational* attributes forming part of the definition of a composite class are of the form $R:H\rightarrow Q$, where $Q\subseteq T^{N+1}\otimes...\otimes T^M$ (i.e., **M** other sets of objects). The *emergent behavioral attributes* are of the form $b:P^1\otimes...\otimes P^K\rightarrow\wp(P^1\otimes...\otimes P^K)$, where P^k denotes the kth composite structural or relational attribute. These attributes are needed when there is some information or property that can be ascribed only to the association of two or more objects. An example of an emergent attribute is **Grade**, which characterizes neither a **STUDENT** nor a **COURSE** individually, but a certain student-course pair.

Although not dealt with in theories of concepts, it seems reasonable to assume that the components of a composite object are linked via a relational attribute. This is embodied in the following postulate.

Postulate 7:

In a composite class (object), **n** "complementary" n-ary relational attributes link the **n** component classes (objects).

That is, a composite of instances of $T^1,...,T^N$ is based on relational attributes such as $R:T^1\rightarrow\wp(T^2\otimes...\otimes T^N)$[96]. For example, an **ENROLMENT** composite is based on an **Enrolled** relational attribute of class **STUDENT** with class **SECTION**, as well as a complementary **Has_enrolled** attribute of class **SECTION** with class **STUDENT**.

The model adopts a further important assumption.

[96]For each of the other components, there is a similar relational attribute.

Assumption:

A given collection of objects may, at any time, form only *one composite object of a particular composite class* (i.e., the object is a surrogate or identifier of a composite thing) via a single relational attribute.

This assumption limits what may be modelled as a composite object. For example, there cannot be two composites with parts *John* and *UBC* in a composite based on the **Student** relational attribute of **STUDENT**[97]. Again, although this consequence is not explicit in the classification literature, it does have intuitive validity, particularly when physical objects are considered. For example, a given collection of parts may belong to only a single automobile (at any time). Interestingly, this assumption has also been adopted by some object-oriented data models which support composition (e.g., Banerjee et al., 1987).

The assumption is embodied in MIMIC through emergent structural and relational attributes. For example, if **Amount** is an emergent attribute of class **DEPOSIT**, it cannot be that Amount = {a:ACCOUNT⊗TELLER→POSITIVE_NUMBER}, since several deposits made be made on the same account by the same teller. In other words, the relationship is not functional if **DEPOSIT** has only the components ACCOUNT and **TELLER**. That is, if $\mathbf{c_1}$ and $\mathbf{t_2}$ are components of **DEPOSIT** which form, say, two deposit objects with amounts 25 and 50, it would be required that $a(c_1,t_2)=25$ <u>and</u> $a(c_1,t_2)=50$. Consequently, the formalism[98] enforces the assumption that a given collection of objects can form only one composite of a particular class (at any time). In order, then, to model deposits to a single account using the same teller, an additional component class is needed, the members of which vary for a single combination of account and teller. This recognizes that the deposits are, indeed, different objects: that is,

[97]However, there may be another composite of *John* and *UBC* which is, for example, based on the **Employed_by** relational attribute of **EMPLOYEE.**

[98]In particular, the way in which composite objects and attributes are defined.

they have a different existence.

In contravention of this assumption, one might argue that a composite class such as **DEPOSIT** may have parts **ACCOUNT** and **TELLER**, and that several composite objects (orders) may contain the same components (e.g., several deposits on the same account by the same teller). However, the position taken here is that this composition is incomplete. Each deposit has a distinct existence and occurs at a different time[99].

This view of composition has an implication for conceptual modelling activities. Each relational attribute identified in modelling a domain should be examined to determine whether there is any information about the association which is relevant. In such cases, a composite class is needed, with the classes involved in the relational attribute constituting its components. The additional information about the association can then be formalized as emergent structural/relational attributes. Subsequently, constraints on behavior with respect to these attributes can be identified.

To summarize, the MIMIC model supports a distinction between simple associations, modelled as relational attributes, and associations which can be viewed as objects with emergent attributes. This distinction depends on whether the domain of structural and relational attributes is a set of objects or the Cartesian product of several sets of objects. Associations with emergent attributes are modelled as composite objects. Thus, a semantic difference between the two kinds of associations - one involving attributes of the association and the other not - is captured. It can be argued that (almost) any association may have attributes (e.g., time at which the association began) and, hence, may be modelled as a composite object. However, for the purposes of many applications, such attributes may not be relevant.

[99]Consequently, one view is that the components of **DEPOSIT** should actually be **ACCOUNT**, **TELLER**, and **TIME**, where **TIME** is a class of objects whose extension is a set of time points. Philosophically, there may be problems in viewing time points as objects. Since there is no fundamental principle in theories of concepts to resolve this issue, the best that can be said here is that treating time points as objects may merely be a surrogate for the fact that the deposits have different existence.

Relational attributes allow simpler modelling of situations in which no relevant attributes are "attached" to an association.

5.6 ON THE IMPLICATIONS OF A UNIFORM FORMALISM

It is widely recognized that both *structure* and *behavior* are important for the complete modelling of knowledge about entities (e.g., Borgida, 1984; Brodie, 1984). Behavioral knowledge indicates how the states of things may change over time. This must, in one form or another, be incorporated along with state knowledge in information systems if they are to accurately track knowledge of the state of the things they represent (Wand and Weber, 1988).

In many conceptual modelling methods, behavior is modelled in a process-oriented way (and, in some cases, is not dealt with at all). For example, NIAM (Verheijen and Van Bekkum, 1982) captures behavior through information flows and processing. Object-oriented systems (languages and data models) encapsulate behavior with structure through methods. Methods implement mechanisms which describe how changes to instance variables occur and, implicitly, determine which changes may (not) occur. By contrast, neither the E-R model (Chen, 1976; Teorey et al., 1985) nor most semantic data models (see King, 1989) deal with behavior. An exception to the procedural treatment of behavior is found in Telos (Mylopoulos, 1991; Mylopoulos et al., 1990), which models behavior in terms of activities belonging to activity classes. Activities describe the "what" of change, specifying *pre- and post-conditions* which must be satisfied before and after a change. However, activities also contains *parts* which come closer to describing the "how" of change.

The issue which emerges here is that structure and behavior are usually modelled in quite different ways, increasing the complexity of models such as those mentioned. Since structure and behavior are both important elements of knowledge about things, it would be simpler to have a uniform approach for representing both. The notion of uniformity was first recognized in the Taxis model (Mylopoulos et al.,

1980) and plays a large part in Telos. However, the notion of uniformity discussed here is at a more abstract level, since MIMIC is completely non-procedural.

In the MIMIC model, both structural and relational attributes are defined as sets of functions (from objects to values or sets of things, respectively). This function-based approach to modelling knowledge is very simple. It allows the description of state by applying active functions to objects. *Furthermore, behavioral attributes are also modelled as functions.* The domain of a behavioral attribute is the Cartesian product of a number of structural and/or relational attributes, and the codomain is the power set of the domain. A behavioral attribute thereby specifies, for any given combination of the active functions of one or more structural/relational attributes, which events or changes can occur. Intuitively, this approach makes sense since structure, relationships, and behavior are all elements of knowledge about things. Furthermore, it is a parsimonious and implementation-independent approach to integrating static and behavioral knowledge. However, it is also restricted to describing the "what" of change and not the procedural details of "how" change is performed. In this regard, it is worth mentioning that this approach is somewhat related to the *λ-calculus* and *functional programming* (see Backus, 1978) in hiding the procedural details of change. According to Backus, a functional programming system "has a loosely-coupled state-transition semantics in which a state transition occurs only once in a major computation" (p. 619). These "major computations" appear analogous to the MIMIC notion of an *event* conforming to a behavioral attribute, since the model does not contain mechanisms for describing procedurally how the change is brought about.

The primary consequence of this view is that only knowledge about *allowed* behavior is represented. That is, behavioral attributes may be viewed as restrictions or laws which describe the changes an object may or may not undergo (by virtue of belonging to a class and, therefore, possessing the attributes defining that class) by restricting changes to the active functions of attributes of that class. Models specifying how changes occur (e.g., through control statements) commit to a particular view of the implementation of change (e.g., transactions in ACM/PCM and Taxis). MIMIC is more abstract than these models in that behavioral attributes can accommodate many concrete descriptions of how change occurs. The model thus allows for a greater degree of freedom in later stages of system design and implementation, as it does not impose a particular description of change mechanisms. That is, its constructs are completely free of implementation primitives.

It may be argued that this begs the question of representing the "how" (procedures) of change. Since, in an implemented system, programs execute the changes which can occur (i.e., implement behavioral attributes), a detailed description of how changes are to be performed is a prelude to implementation. However, the intent of MIMIC is to focus purely on the problem domain, rather than the implementation domain (cf. Mylopoulos, 1990). Consequently, the model does not specify how behavior is to be implemented. However, the functional approach to behavior does specify constraints which any implementation (of transaction mechanisms) must adhere to, without committing the analyst to a specific approach to the implementation. It is thus valuable as a preliminary modelling step.

In addition, it is worth noting that in discussions of methods, messages, and encapsulation in object-oriented programming languages, there is an emphasis on hiding the "how" of behavior and creating a protocol for communication between objects (e.g., message passing) that hides the implementation details of methods (e.g., Goldberg and Robson, 1989). MIMIC extends this idea uniformly (back) to the conceptual modelling phase of development by considering behavior only in terms of states before events and potential states after events (cf. Wand and Weber, 1988), without consideration of intervening processes.

In summary, several existing modelling methodologies recognize that both structure and behavior are important elements of knowledge to be captured during the conceptual modelling phase of systems development. However, these approaches often introduce different mechanisms for modelling behavior versus structure. Such mechanisms may, for example, use standard program control structures (sequence, iteration, and choice) that are closely linked to software (e.g., Brodie and Ridjanovic, 1984; Jackson, 1983). The MIMIC model, on the other hand, captures behavior through functions restricting changes in state. The uniform approach to modelling structure, relationships, and behavior is both intuitively appealing and simple: the former because a common approach is applied to static and dynamic knowledge, and the latter because it uses fewer modelling constructs. In practical terms, the model should be useful as it completely removes implementation or software considerations from conceptual modelling, thereby allowing flexibility in the later stages of designing transactions and programs to enforce behavioral constraints.

5.7 ON THE MEANING OF TIME

Time is important in conceptual modelling because knowledge about when certain changes occur *relative to others* is important to users. Specifically, there are at least two critical uses of temporal information. First, in representing knowledge about an application, it is often necessary to capture the *time at which an event occurs* (i.e., when a certain state is realized). For example, banks maintain records of when account transactions occur, when loan payments are made, and so on. Second, temporal knowledge is used to answer questions about *intervals over which states or conditions are true.* For example, the duration of account balances is essential knowledge in determining interest earned.

Accounting for time in conceptual models is an issue which has received some attention and which has been criticized for inadequacies (e.g., Richter, 1985; Schiel, 1985; Kung, 1983; Bubenko, 1980). The issues appear to be closely related to general questions about modelling temporal information in knowledge representation (e.g., Shoham, 1988; Allen, 1984; McDermott, 1982). However, this AI work focuses on *reasoning* about temporal matters and, consequently, tends to develop detailed temporal logics based on either time points or time intervals as primitives. Hence, it is not examined in detail here.

Problems in modelling time have been recognized. For example, McDermott (1982, p.101) states that "no one has ever dealt with time correctly in an AI program, and there is reason to believe that doing it would change everything." In the conceptual modelling context, Kung (1983, p.141) states that "[a]lthough there are a large number of conceptual models, most of them fall into what is called snapshot approaches", which do not adequately deal with time.

It is not the intent of this research to propose a complete model of time. However, since the ability to deal with temporal information is important to many IS applications and since knowledge of temporal relations is part of our knowledge of the world, some treatment of time in conceptual models is crucial. This section shows that MIMIC's treatment of behavior through events implicitly incorporates a simple treatment of time[100], and then discusses some consequences and contributions of that treatment and an extension of it.

According to Schiel (1985), there are two views of time: *absolute* and *relative*. The absolute view treats time as an infinite sequence of points[101] independent of things. The relative view claims that time has no meaning independent of things and events, as expressed succinctly by Shoham (1988, p.1):

> Of course, the passage of time is important only because *changes* are possible. In a world where no changes were possible - no viruses infecting blood systems, no electrical charges changing, no changes in program counters, not even changes in the position of the sun in the sky or the position of the hands on our wrist-watches ... the very concept of time would become meaningless.

This "time as change" perspective is implicitly adopted in the MIMIC model, but it is also argued that the former and latter views can be closely linked.

The basic view of time in MIMIC is derived from the primitive attribute constructs. A structural (respectively, relational) attribute is formalized as a set of functions from a domain of objects to a codomain containing values (respectively, sets of objects). A single *active* function from the set describes the state of the domain with respect to that attribute. The state of a set of objects possessing **m** structural

[100]Any conceptual model which deals with behavior necessarily includes a rudimentary notion of time, according to a view which sees time as change. Examples of conceptual models which explicitly deal with time include activity and behavior modelling (Kung and Solvberg, 1986), Telos (Mylopoulos, 1991; Mylopoulos et al., 1990), and the Set-function model (Bertziss, 1986).

[101]Time may also be construed in terms of *intervals* as primitives (Shoham, 1988; Allen, 1983).

and **n** relational attributes is an (m+n)-tuple containing **m+n** active functions. The notion of *change* is incorporated in the model through the replacement of active functions (of one or more attributes) with other functions from the same attribute set. Such a replacement is termed an *event*. An event changes the state of at least one object in the domain of each of the attributes affected. The model does not, at this level, deal with the spacing of events on some interval scale. Consequently, the only notion of time directly supported is that the passage of time is recognized by events, so that time intervals are not recognized.

This is consistent with the previously described view of the world (or knowledge of it) in which "time" has meaning only in that change occurs. MIMIC provides a natural mechanism for capturing this by modelling attributes as sets of functions, whereby an active function describes the state of a set of objects with respect to the attribute in question and may be replaced by another active function to reflect a change in knowledge of the state of the world. If an attribute has a single function, no change may occur. Naturally, time has no meaning with respect to such an attribute.

The model equates change in time with events. However, in general, events (in an information system or in the real world) are not equally spaced on any interval. For example, a bank may have **x** deposits to savings accounts in one day of operation and **y** in another day, where $x \neq y$. Furthermore, such events may be unequally spaced through the day. Since the spacing or intervals between events is often critical knowledge for planning or decision making[102], a mechanism to capture "distances" between events is important in a conceptual model. Consequently, a refinement to MIMIC is needed in order to recognize time intervals.

The notion of "unequally spaced" has meaning only in the sense that there is some standard or

[102]Consider a situation where customers perform deposits and withdrawals on bank accounts. The spacing between these events helps determine factors such as teller utilization and queue lengths. Consequently, this knowledge is critical in simulating different service options in order to decide on an appropriate level of customer service.

scale against which other events can be measured. Such a scale will hereafter be referred to as a *clock*. In the banking example above, deposits may be made at 9:14, 9:23, 10:01, and so on. For most people, time is measured by one or more physical (mechanical or electronic) clocks. Of course, such clocks generally provide a discrete (or continuous) approximation of continuous change based on the rotation of the earth. This approximation is adequate for day to day life, and certainly for business information systems needs. It is manifested in physical events such as the movement of hands on a clock face or changes to values on a digital display. In addition, people appear to have a notion of *absolute time* since we often question whether a specific physical clock is telling us the "correct" time.

Unfortunately, the theories of concepts discussed in Chapter 3 do not offer much insight into the concept of time and the nature of temporal knowledge. Furthermore, while the formalization of classical concept theory in Chapter 4 does offer the simple relative view of time based on events discussed above, this alone is inadequate to model the spacing of events. Consequently, a simple extension of the model is considered here which, while not directly based on cognitive research, illustrates one possible way of introducing an "absolute" notion of time which is uniform with respect to the constructs of the model already introduced and which allows the spacing of events on an interval scale to be measured.

Two assumptions underlie what follows. The first and most important is that there is a concept of CLOCK: that is, a standard by which other events can be measured. This assumption seems to be essential in models of time (Richter, 1985; Bunge, 1977). The second assumption is that the clock is in some sense "universal". This means that a given community of individuals can agree on a single common clock, against which other (individual) clocks can be calibrated. This assumption should not cause too much trouble given the frequency with which individuals compare "personal" time (as measured by, say, a watch) with "actual" time (as decreed by some authority).

These assumptions give rise to the definition of a class which will be called **CLOCK**[103], whose extension is a set of clocks. For simplicity, this class may be assumed to contain only a single object (i.e., the universal clock). **CLOCK** is defined in terms of a single structural attribute, along with a single associated behavioral attribute.

The structural attribute is **Time_value** = $\{tv_i | tv_i\text{:CLOCK} \rightarrow \text{NUMBER}\}$, where NUMBER is a finite[104] set of values (e.g., integers). That is, each function in **Time_value** is a single pair (if the class is assumed to contain a single object) whose first element is the clock object and second element is a value from the domain NUMBER. The behavioral attribute, **Change_time**, enforces the constraint:

$<tv_1,tv_2> := tv_2(c) = tv_1(c) + 1$ [105],

where **c** denotes the clock object. In other words, **Change_time**:Time_value→Time_value[106] such that for any i, Change_time(tv_i)=tv_{i+1}, where $tv_{i+1}(c)=tv_i(c)+1$. This class and the events on it are intended to model our knowledge of time as represented by an ordinary clock. An event might be the discrete movement of a second hand, or a change of numbers on a digital display. This kind of event provides a base to which more interesting events can be associated when the relative spacing between pairs of events (such as changes to account balances) is important. In these cases, the number of "clock tick" events, each designating an elapsed unit of time, between a pair of other events (e.g., two deposits on an account) provides a scale for measuring the distance between these events. Furthermore, the time at which an instance of a class (e.g., person, deposit) is created may be an attribute of the class, the value of which

[103]This class makes use of the constructs of the model (i.e., objects and attributes) and, therefore, is embedded in the model (actually, in any instance of the model) and not really an "extension" of it.

[104]For the purposes of this exposition, finiteness is assumed in order to impose a restricted time horizon and preclude a continuous notion of time. Given the domain of interest (i.e., business applications), this appears to be a reasonable restriction.

[105]See Appendix 1 for an explanation of this notation for behavioral attributes.

[106]The codomain of this attribute is a single element of the power set of **Time_value**: namely, the set **Time_value** itself.

is the state of the clock when the object is created.

The usefulness of this approach is that it defines a **CLOCK** as a mechanism for recognizing intervals between events. When the spacing between events is important, there is information about an event that is relevant knowledge. Consequently, *records of events* may be viewed as objects. This may appear at first to be strictly an information systems issue, since records of transactions are extremely important in IS applications for accounting and auditing purposes. However, there is a cognitive basis for modelling records of events as objects. Specifically, people abstract information about events which enables them to answer questions later. For example, a person making a deposit to a bank account[107] has a mental record of the event, and can later provide information such as the **time** and **amount** of the deposit. This record is thus an instance of a concept, **DEPOSIT**, which is an abstraction of the common attributes of cognitive records of deposit events.

In this case, the time of an event may be a structural attribute of its record, which is assigned the value of the state of the clock when the event object is created. Of course, the creation of an object as an instance of **DEPOSIT** accompanies an event (change in the active function) affecting the **Balance** attribute of class **ACCOUNT**, such that a particular instance of **ACCOUNT** has its value increased by the value of **Amount** of the newly created instance of **DEPOSIT**.

The constraints among these actions can be specified in more detail, although this will not be done here since it is more important at the implementation level. At the conceptual level, it is enough to know what the connection is. This involves enforcing a requirement that a given event (e.g., a change in the active function of **Balance** of class **ACCOUNT**) be accompanied by the creation of at least one instance of a specific event-record class (e.g., DEPOSIT, WITHDRAWAL, TRANSFER). Consequently, the fact that, for example, three **DEPOSIT** transactions (d_1, d_2, d_3) may not be equally spaced, can be captured by comparing the "differences" in the times of occurrence (e.g., value of **Time_of_occurrence** attribute)

[107]The deposit is simply an event, or change of state, of the balance of the account.

of consecutive deposit pairs. The term "difference" has meaning here since the value of the **Time_of_occurrence** is a number on an interval scale.

In order to better understand the limited treatment of time which MIMIC offers, it is useful to recognize that there are two possible focuses when representing temporal knowledge. The first deals with what is *presently true* (representing knowledge about the state of a real or potential world). As a conceptual model, MIMIC handles this representation through its mechanisms for describing object states, and constraints on changes of state. In fact, the central purpose of the model is to provide these mechanisms.

The second focus in modelling temporal knowledge deals with representing *past truths*. As discussed at the beginning of this section, there are two important uses of past truths. One is to provide answers to questions about when events occur. In the MIMIC framework, this can be captured by maintaining a record of the event as an object, one of the attributes of which takes on the value of the clock at the creation of the event record. The object continues to exist without changing state, and can always be referenced to answer questions about when the corresponding event occurred. The second use of past truths is to answer questions about intervals over which assertions or states are true. This use illustrates an important limitation of MIMIC's treatment of time. Since MIMIC's framework for dealing with time is not interval-based, it does not directly permit questions about time intervals (e.g., how long did an account balance remain within a certain range?) to be answered. However, records of events provide the basic information necessary to answer such questions, since intervals can be determined by comparing the times at which the relevant events occurred. Nevertheless, this requires an extension of the MIMIC model to include a reasoning mechanism, and is beyond the scope of this thesis.

To summarize, the intent of the preceding presentation is not to develop a comprehensive model of time, as would be necessary to perform detailed temporal reasoning (e.g., Shoham, 1988), since theories of concepts generally do not deal with how knowledge of things changes (is updated) over time. The more limited goal is to show that the fundamental constructs of MIMIC have a very basic model of time as change, captured by the notion of an event. This, in turn, can be enhanced by introducing a clock to recognize that events are not necessarily equally spaced (and also to answer questions related to the ordering of and distance between events). Thus, for this research, time is still viewed as change, but at the finer level of clock change and relative time becomes, with the clock as a standard, absolute.

5.8 ON THE OBJECT PARADIGM AND CONCEPTUAL MODELLING[108]

As Chapter 2 indicated, much attention has recently been focused on object-oriented approaches to systems analysis and design (e.g., Booch, 1991; Coad and Yourdon, 1991; Bailin, 1990; Henderson-Sellers and Edwards, 1990; Wirfs-Brock and Johnson, 1990). Of particular interest here is that the object paradigm has been applied to conceptual modelling and advocated as a natural approach for representing knowledge about a domain (Coad and Yourdon, 1991).

Notwithstanding the interest in object-based approaches, there has been widespread disagreement about the meaning of terms such as 'object' and 'object-oriented' (e.g., Nierstrasz, 1987; Stroustrup, 1987; Wegner, 1987; Stefik and Bobrow, 1986; Nygaard, 1986; Pascoe, 1986; Stoyan, 1984), particularly as applied to programming. Only the concept of encapsulation seems to be universally accepted. Classification and inheritance have also been generally embraced, but with notable exceptions (e.g., Lieberman, 1986). Other concepts, such as independence, homogeneity, message passing, composition and concurrency, have varying degrees of support.

The justifications offered for these characteristics tend to center around supporting software *reusability*, improving software *reliability*, and supporting *incremental development* (Meyer, 1989; Nierstrasz, 1987). At the same time, however, arguments have been made that many of the features of the

[108]This section is based on joint research with Yair Wand, as described in more detail in the paper "The Object Paradigm - Two for the Price of One?", *Proceedings of the First Workshop on Information Technologies and Systems (WITS'91)*, Cambridge, MA, December 1991, 308-319.

object paradigm support world modelling in a natural manner (e.g., Mylopoulos, 1990; Goldberg and Robson, 1989; Stefik and Bobrow, 1986; Birtwistle et al., 1973). Hence, there is an apparent confusion of what we (see Footnote 108) call *implementation* and *representation* advantages[109]. The position taken in this thesis is that, since conceptual modelling is strictly concerned with capturing knowledge about a domain of interest (on which an IS is based), *only the elements which support the development of good representations of a domain should be included in object-oriented conceptual models.* In the context of this research, this means that only the object constructs which support the direct representation of cognitive constructs should be included in a conceptual model. Consequently, the remainder of this section examines the degree to which a number of object characteristics are supported by MIMIC.

Classification/Instantiation

In most object-oriented systems, there is a clear distinction between *instance objects* and *class objects* (e.g., Goldberg and Robson, 1989; Banerjee et al., 1987; Stefik and Bobrow, 1986). Class objects are abstractions which describe a set of instances in terms of common characteristics (instance variables and methods). Classes are useful for creating instances and contain the code (methods) for manipulating the state (instance variables) of instances. The instances are have individual identity (Khoshafian and Copeland, 1986) and are intended to be in one-to-one correspondence with entities from some domain.

The MIMIC model clearly distinguishes *objects* from *classes.* Objects are in one-to-one correspondence with instances of concepts from some domain and possess attributes which represent knowledge about these instances. Classes, which are in one-to-one correspondence with concepts from that domain, are defined in terms of a collection of attributes, including a characterization of allowed behavior.

[109]In a similar vein, Mylopoulos (1990) states that "research on object-oriented databases should address world modelling or programming, but not both" (p.13).

Encapsulation

In most discussions of objects, class definitions encapsulate *statics* (e.g., instance variables) and *dynamics* (e.g., methods) (e.g., Nierstrasz, 1987). That is, a complete specification of a class includes definitions of its structure (hence, the structure of its instances), along with a description of how the states of instances may change. Encapsulation provides a well-defined, implementation-independent interface for accessing and manipulating object states.

In MIMIC, a class encapsulates *structural* and *relational* attributes which, in turn, imply a set of *behavioral* attributes. These jointly represent the three kinds of knowledge defining concepts. The behavioral attributes specify how the values of the structural and relational attributes may change over time for the objects which make up the extension of the class.

Class Specialization and Inheritance

In most object-oriented systems, there is support for defining some classes to be subclasses or specializations of others. The subclasses may possess added instance variables and methods, and often the methods of a subclass may override those of its superclass(es). There is no general agreement as to whether object-oriented systems should support *multiple* inheritance, in which a subclass has more than one superclass and, therefore, inherits instance variables and methods from each. In any event, specialization and inheritance support code reusability and reduced redundancy.

In MIMIC, a subclass *must* possess structural and/or relational attributes in addition to those of (each of) its superclass(es). Such added properties form the basis for defining new classes. In addition, the subclass may possess additional behavioral attributes. In the model, there is no notion of overriding the attributes of a superclass. Furthermore, the notion of multiple inheritance follows naturally from the way in which subclasses are defined, since a new class may possess the properties which define two or more existing classes.

Composition

The idea of composite objects and classes has been recognized in some object-oriented applications, particularly in the database area (e.g., Banerjee et al., 1987). Composite objects belong to composite classes. The latter are comprised of a number of component classes, and in addition, *may* possess certain emergent instance variables and methods. Composite objects may be stored and retrieved as a unit, thereby reducing access time.

In MIMIC, a composite class is defined in terms of (i.e., *must* possess) a number of emergent attributes. The domain of these attributes consists of composite objects, which are uniquely identified as a collection of component objects. Composite classes represent composite concepts, thereby reflecting the knowledge that some things are composed of simpler things.

Independence

In many object-based environments, methods are the only mechanisms by which changes to the state of objects can occur. That is, if there is no method to execute a particular change, such a change cannot be imposed, for example, by directly manipulating instance variables. Independence is supported by encapsulation and enhances software reliability.

In the MIMIC model, events are constrained by behavioral attributes. These attributes implicitly determine which changes can and cannot occur. Thus, representations are independent in the sense that allowed changes to state are embedded completely in behavioral attributes, although these attributes may span several classes.

Communication

At least two metaphors have been used in the object literature to describe communication. Message passing (Stefik and Bobrow, 1986; Robson and Goldberg, 1981) implies a direct request by one object to another. This is the most widely used approach described in object-oriented programming and databases. A second approach involves communication through a public blackboard, or workspace, which is checked by other objects in search of communications intended for (or applicable to) them (Tsichritzis et al., 1987).

Much of the discussion of communication in the object-oriented programming literature understandably focuses on implementation. However, communication can be viewed conceptually as a vehicle for supporting object independence. A communication mechanism provides an orderly protocol which prevents objects from changing state arbitrarily and, thereby, supports independence.

MIMIC includes only a very abstract notion of communication. Behavioral attributes constrain the allowed changes of state of objects. Any event (change in active function of one or more structural/relational attributes) must conform to all behavioral constraints which affect these attributes. The model distinguishes constraints on changes to (1) several attributes within a (non-composite) class, (2) attributes across several classes, and (3) composite attributes. This may be viewed as a communication mechanism in the abstract sense above: namely, in providing an orderly protocol which prevents object states from changing arbitrarily. There are no implications for a particular mechanism for communication, such as message passing.

Uniformity (Homogeneity)

Many applications of the object-oriented approach view every component of the system as an object. For example, the syntax of Smalltalk recognizes only the sending of messages to objects. This means that every possible value (e.g., a number) is an object and so are classes. However, the relentless pursuit of homogeneity leads to an obvious problem of infinite regress (Nierstrasz, 1987). Hence, the notion of an object's properties, in turn, being objects, must terminate at some level with primitive objects that do not have properties. Nevertheless, there is obvious value in thinking about the domain in terms of a single construct. When "everything is an object" and objects are independent, an application is expected to be well-behaved. If object behavior is completely determined by its own definition and state, and the behavior of the system is determined by communication among objects, it should be possible to predict the state of a system at any time, given a starting state (for all objects in the system) and a sequence of all subsequent communications. Hence, homogeneity is justified as a means of supporting object system reliability.

From a conceptual point of view, objects are viewed as representations of entities in the world (Coad and Yourdon, 1991, p.1; Digitalk, 1989 p.3). However, there is no compelling reason why object state variables and behavior, along with classes, should also be objects. MIMIC does not view classes or properties as objects, since they do not represent identifiable, independent things in the domain of interest. It seems that the disparity between the implementation and representation views regarding homogeneity stems from fundamentally different assumptions made by each. A system, once implemented, exists as a composite thing in the world, and further, as a composite thing. If knowledge about this thing and its components are modelled, they can be viewed as objects. It is important that these objects are not necessarily representations of any real thing in the application domain (unlike customers, students, accounts, and so on). Rather, they are implementation artifacts. In other words, from a representation point of view, an object represents a thing or an instance in reality, while from an implementation perspective, an object is a component of the implemented system[110].

To summarize, this section has compared the features of MIMIC with a variety of characteristics advocated by proponents of the object paradigm. There are many similarities, suggesting that object-based conceptual models can serve as "natural" vehicles for representing knowledge. However, there are also some important differences, indicating that the roots of object-orientation in programming languages may

[110] Of course, one implemented, the information system and its components become also part of reality. This issue is not pursued here.

hinder the development of object-based conceptual models if characteristics are adopted from programming and applied to modelling the subject matter of an IS without explicitly considering their value in representing knowledge about that subject matter.

5.9 SUMMARY

This chapter uses the constructs of the MIMIC model developed in Chapter 4 to examine several poorly understood issues in conceptual modelling, including *criteria for defining a class structure*, the *naturalness of lattice class structures*, the *meaning of ISA*, *composition*, the *meaning of time*, and the *representation value of object-oriented models*. In each of these areas, MIMIC's formalism and foundation in concept theory are used to examine the meaning of conceptual modelling constructs. In addition, a number of specific guidelines for modelling are suggested. The analysis indicates that MIMIC's grounding in a theory of categorization, along with its formalization, provides insights into several issues which are not well understood in conceptual models with a less clear theoretical and formal basis.

CHAPTER 6

A COMPARISON OF CONCEPTUAL MODELS

6.1 INTRODUCTION

In Chapter 5, the contributions of MIMIC to understanding and addressing several problems in the conceptual modelling area were examined. In this chapter, the MIMIC model is used as a foundation to evaluate the "cognitive content" of four quite different approaches to conceptual modelling: the Extended Entity-Relationship (EER) model (Teorey et al., 1986; Chen, 1976), Telos (Mylopoulos, 1991; Mylopoulos et al., 1990), NIAM (Verheijen and Van Bekkum, 1982), and Object-oriented Analysis (OOA) (Coad and Yourdon, 1991). The motivation for this evaluation is MIMIC's grounding in the classical theory of concepts. Since MIMIC is a conceptual model for directly representing knowledge about things, it can be used to determine the degree to which other conceptual models support the representation of cognitive constructs, according to the classical view of concepts. The specific objectives of this chapter are, first, to determine which constructs of the MIMIC model are supported in several conceptual modelling methods, and, second, to examine the significance of constructs found in other models, but not supported by MIMIC.

There is a wide range of systems development methodologies which address, to a greater or lesser degree, the conceptual modelling stage of IS development (e.g., see Bubenko, 1986; Brandt, 1983; Floyd, 1986; Olive, 1983 for comparisons of several). Performing a useful comparison with a large number of these would be impractical. Instead, a detailed evaluation of four models with quite different origins is performed here in an attempt to assess the degree to which each supports the representation of knowledge about a subject domain.

The four methods were chosen for different reasons. First, the EER model is a well-known and widely-used tool for conceptual database design. Although it has recognized problems (e.g., its inability to capture behavioral information), the EER model does serve as a well-understood starting point for a comparison of models. Second, Telos is explicitly intended for conceptual modelling (among other things), and is said to be based on knowledge representation principles (Mylopoulos et al., 1990). Since MIMIC explicitly adopts a knowledge representation approach based on concept theory, a comparison with Telos may provide insight on the consequences of adopting different foundations for representing knowledge about things. Third, NIAM is a well-known example of European research in the conceptual modelling area, addressing directly the notion of modelling an enterprise or slice of reality[111]. Since much of the impetus for conceptual modelling has come from the European research community (e.g. Nijssen, 1976; Schmid, 1977; Bubenko, 1980)[112], consideration of some of the results of these efforts is necessary for any evaluation of models to be representative of work that has been done. Finally, the object paradigm and object-oriented analysis have attracted tremendous attention in recent years, in part because of claims that such approaches allow "natural" representations of a problem domain to be constructed. A comparison of OOA with MIMIC may provide insight into the degree to which OOA may be called natural, according to the classical theory of concepts.

6.2 CRITERIA FOR COMPARISON

The comparison and evaluation of conceptual models and systems development methods is an extremely difficult task (Sol, 1983). Evidence of this is found in a series of IFIP conferences on the comparative review of information systems design methodologies (Olle et al., 1982; 1983; 1986), which together demonstrate a variety of approaches to comparing models. Among the impediments to performing a comparison are the variety of constructs and inconsistency of terminology among methods, the lack of

[111]NIAM also covers other aspects of the development process which are not considered here.

[112]These references capture but a small sample of conceptual modelling methods. Further reviews and comparisons are found in (Olle et al., 1982; 1983; 1986).

formalism in some models (which hinders reconciliation of constructs and terminology), and the diversity of scope of different approaches with respect to the phases of IS development addressed (e.g., analysis only, analysis and design, analysis through to implementation). Illustrating the first two difficulties, it is unclear whether terminology such as *entities* (Chen, 1976; Jackson, 1983) and *objects* (Verheijen and Van Bekkum, 1982; Mylopoulos et al., 1990) mean the same thing across models, particularly if these terms are not formally defined. Likewise, the similarity between *entity sets* in the EER model (Teorey et al., 1986) and *classes* in Telos (Mylopoulos et al., 1990) is difficult to assess. These difficulties are considered explicitly in the comparisons which follow.

Sol (1983, p. 4) suggests a number of approaches to evaluating methods, including:

1. *Comparison to an idealized methodology*

   This approach begs the questions of where the "ideal" is to be obtained and how such a judgment is to be made. If such an ideal existed, it seems that other methods, and hence a comparison, would not be needed.

2. *Accumulation of important features from several existing approaches to use for comparison*

   This presumes the "goodness" of existing methodologies and their features and relies on a subjective evaluation of the importance of the features selected.

3. *Definition of a meta-model or framework within which methods can be compared*

   Such a model may be useful in reconciling terminology and construct differences among methods being compared. However, the success of an evaluation will depend heavily on the quality and expressiveness of the meta-model. In addition, since "meta-model" may be closely related to "idealized method", this approach may encounter the same problem stated in point (1).

With respect to the second point, a number of attempts have been made to enumerate a set of features for comparing and evaluating modelling approaches. In general, the criteria tend to focus on the capability of models to capture *static knowledge*, *behavioral knowledge*, and *constraints* on states and behavior (Wand and Weber, 1989; Borgida, 1984)[113]. An example of such a checklist in the context of data modelling is given by Brodie (1984). However, his set of criteria includes a number of issues, such as exception handling and query facilities, which are related, not to modelling the domain, but to modelling a database. Since these are essentially implementation-oriented, they are not discussed here. Nevertheless, the emphasis on capturing structure, behavior, and the constraints on both, is appealing and influences the present evaluation.

This comparison draws from the first and third approaches above. The primary interest is in evaluating the mechanisms or constructs a model provides for *representing knowledge about domain entities* (i.e., things which constitute the subject matter of a proposed information system)[114]. Since MIMIC is explicitly derived from a theory of the structure and organization of knowledge about domain entities, it will be treated as a "benchmark" for comparing and evaluating the other models in terms of their support for cognitive constructs. Each model is assessed in terms of mechanisms provided for representing *existence*, *attributes* (structural, relational, and behavioral), *classes*, *class organization* (specialization), and *composition*. In addition, constructs of any model which are not supported by MIMIC are described and the significance of these differences analyzed.

Two caveats are mentioned before proceeding. First, most conceptual models do not precisely define their constructs and, furthermore, use widely varying terminology. As a result, it is often necessary to provide an interpretation of the meaning of some characteristic of a model in comparison to one of MIMIC. Second, the value of the comparison is bounded by the utility of classical concept theory. The

[113]This may constitute a meta-model in the sense described in point (3).

[114]Consequently, the analysis is restricted in two important senses. Since two of the models considered (Telos and NIAM) address other aspects of systems development, only the parts dealing with conceptual modelling are considered. In addition, no attention is given to guidelines for using a model, although such prescriptions are a crucial part of the success of any model.

limitations of the classical view as a theory of the structure and organization of knowledge (see Chapter 3) result in limitations for MIMIC. However, since all conceptual models examined here are concerned with representing knowledge about well-defined organizational domains (for which the classical theory of concepts provides suitable representation primitives), comparison to the MIMIC model is justified.

6.3 EXTENDED ENTITY-RELATIONSHIP MODEL (Teorey et al., 1986; Chen, 1976)

The entity-relationship (E-R) model was developed as a tool for logical data design which "adopts the more natural view that the world consists of entities and relationships" (Chen, 1976, p. 6). The main constructs of the model are *entities* (and entity sets)[115], *relationships* (and relationship sets), and *attributes*. The model has been extended (EER) to include hierarchical links between entity sets (generalization/specialization) and to better represent *n-ary* relationships (e.g., Teorey et al., 1986). This collection of constructs is used to capture structural knowledge about things in a domain of interest, primarily through a graphical technique which produces *entity-relationship diagrams*. In this section, the use of these constructs is discussed, illustrated through examples, and compared to constructs of MIMIC.

In the EER model, *entity sets* denote collections of entities, where an entity "exists in our minds" (Chen, 1976, p.11). The entity construct resembles[116] the object construct in MIMIC. Both entities and objects represent knowledge of the existence of things in some domain[117]. An example of an entity would be a symbol such as *John*, designating a particular human.

[115]This reflects Chen's original terminology. Most later discussions (e.g., Teorey et al., 1986) drop the term *set*, and use the term *entity* to describe either an individual or set of individuals, depending on context. However, it should be made clear that the model does distinguish instances from sets of instances.

[116]Throughout this discussion, terms such as "resembles" and "is similar to" are used (rather than, say, "corresponds to"), since it is generally not possible to determine a direct correspondence between constructs of different models.

[117]To be more precise, according to the original E-R model, an entity is a mental construct. However, any description of or information about an entity is a representation and, hence, the ER model practically deals with representations of mental constructs.

Attributes in EER are used to characterize or capture information about entities[118] by associating entities with values from some domain. An attribute is defined as a function from a set of entities to either a set of values or the Cartesian product of several such sets (Chen, 1976, p.12). Hence, attributes may be single- or multi-valued. Examples include **Weight** and **Birthdate**, which apply to entities in the set **PERSON**. Each entity in an entity set possesses a value for each attribute of the entity set at any time (e.g., Weight(John)=70). However, the notion of time and mechanisms for changes to values are not incorporated in EER[119]. An EER attribute is analogous to a structural attribute in MIMIC, with two important differences. First, a structural attribute is a *set of functions* (not a single function) from a domain of objects to a codomain of values (e.g., **Weight** = $\{w_i | w_i\text{:PERSON} \rightarrow \text{POSITIVE\_NUMBER}\}$). At any time, a single function from this set describes the actual state of knowledge. As shown in Chapter 5, time is understood completely with respect to such changes in the active functions of attributes. Consequently, MIMIC inherently captures the permanence of values for objects (e.g., Birthdate) with attributes that contain a single function, and for which time has no meaning. This distinction cannot be captured in EER. The second difference is that MIMIC requires that structural attributes be single-valued, whereas in EER, attributes may be multi-valued. The MIMIC treatment reflects a position that there is a fundamental difference between structure and relationships to other objects. While relationships may link an object to several others, an object may be linked to only one value via a structural attribute.

The third major construct in the EER model is the *relationship*. A relationship denotes an association between two or more entities of the same or different entity sets[120]. An example would be **CUSTOMER Holds_account ACCOUNT**. Relationships may be mandatory or optional, and binary or

[118]Attributes may also apply to relationships, as discussed later.

[119]If the EER model was extended further to accommodate change, a model of an attribute as a function might include time as a domain parameter, as in **Weight**:PERSON⊗TIME→POSITIVE_NUMBERS (Section 4.2.2.3.3).

[120]Relationships in EER are grouped into relationship sets.

n-ary. In addition, cardinality constraints enforce conditions on the connectivity of entities in a relationship (e.g., 1:1, 1:N, M:N). A **Holds_account** relationship is 1:N (from **CUSTOMER** to **ACCOUNT**) if a customer may hold several accounts, but each account is held by only a single customer.

Relationships in EER may reflect either of two distinct constructs in MIMIC. The first is the *relational attribute.* A relational attribute is a set of functions mapping from objects in one class to sets of objects from one or more object classes (e.g., **Holds_account** = $\{h_i | h_i\text{:CUSTOMER} \rightarrow Q\}$, where $Q \subseteq \wp(\text{ACCOUNT})$). As with structural attributes, one function from the set is active at any time, thereby providing a treatment of time that is absent in the EER model, including an implicit specification of the permanence of attribute values. Mandatory/optional attributes are captured by excluding/including, respectively, the null set from the codomain of the attribute. Since $R=\{r_i | r_i\text{:}T^1 \rightarrow Q\}$, where $Q \subseteq \wp(T^2 \otimes \ldots \otimes T^N)$, a relational attribute is labelled *binary* if N=2 (note that T^2 may be the same as T^1) and N-ary if N>2. Furthermore, the connectivity of a relational attribute may be constrained by appropriate restrictions on the codomain. For example, in modelling a society which does not allow polygamy, the codomain of the relational attribute **Married_to** is restricted to a subset of the power set of **PERSON** which contains only singletons. As for M:N (and 1:1) connectivity, since every relational attribute has an "inverse", both the attribute and its inverse may have a codomain containing only singletons (for 1:1 linkages), or a larger subset of the power set (for M:N linkages). In other words, the EER model deals with connectivity more explicitly than MIMIC, but the latter permits connectivity information to be captured implicitly through constraints on the codomain of a relational attribute.

Relationships in the EER model do not, however, always correspond to relational attributes of MIMIC since the former may possess attributes. In that case, they appear to be more similar to the notion of composite objects in MIMIC. Since the relationship construct is used to model both relationships with attributes and those without, it is semantically overloaded with respect to representing the elements of classification theory. MIMIC, however, clearly distinguishes the two cases. Relationships without attributes are captured simply through relational attributes. On the other hand, relationships possessing attributes are modelled as *composite objects*. Composite objects are objects and, therefore, possess their own attributes. In addition, each component in a composite is connected to the other components by a relational attribute. For example, **ENROLMENT** may be considered a composite class (having component classes **STUDENT** and **COURSE**) with emergent structural attribute **Grade**. The instances of this class are specific student-course combinations. The component classes are additionally linked by the relational attributes **Enrolled_in** (of **STUDENT**) and **Has_enrolled** (of **COURSE**). Since there is this duality between composites and relational attributes, it is not surprising that the EER model uses the relationship construct to represent two distinct, but related elements of knowledge.

MIMIC provides a basis for resolving the ambiguity in the EER model about whether to model a phenomenon as an entity or as a relationship (Teorey et al., 1986). Simply, a relational attribute which has emergent attributes with respect to the collection of objects involved in the relationship is modelled as a composite object (in EER terms, entity) whose components are the objects involved in the relational attribute. If there are no emergent attributes associated with the collection of objects involved, a relational attribute captures the pertinent knowledge.

In the EER model, entities are grouped into *entity sets*. An example of an entity set in EER would be **EMPLOYEE**, containing entities {*John, Jane, Mary* ...}. Although entities in an entity set possess the same attributes, the construct is primarily extensional, and does not fully reflect the intensional notion of a class in MIMIC. A class such as **EMPLOYEE** consists of a set, **P**, of structural and relational attributes, and an associated set, **B**, of behavioral attributes (i.e., EMPLOYEE = P, where P⇒B), where the attributes in **P** are defined on domains of objects (e.g., {*John, Jane, Mary* ...}, {*John, Paul, George* ...}, {*John, Jane, Frank* ...}). Thus, a class is more explicitly based on a notion of shared attributes than is an entity set. In addition, classes also have an extensional interpretation, given by the intersection of the domains of the attributes in **P**. Attributes appear to play a less important organizational role in EER descriptions than in MIMIC.

In MIMIC, classes are organized in a *class structure*, which imposes certain constraints on the classes it contains. These constraints are based on several cognitive principles, and constitute a set of minimal conditions for the "goodness" of a set of classes. EER imposes no such constraint on the definition of a collection of entity sets.

In line with the theory of concepts on which the model is based, MIMIC incorporates both structural and relational attributes as part of class definitions and, therefore, as part of the knowledge about the objects in these classes. This is in contrast with EER, in which relationships are not well-tied to entity set definitions. The presence or absence of relationships in an EER diagram does not appear to influence the extension of an entity set, whereas the presence or absence of relational attributes explicitly affects a class definition in MIMIC, and will generally affect the extension of that class.

As indicated above, a MIMIC class encapsulates structural and behavioral knowledge. Behavioral attributes specify allowed changes to objects, given any active functions for specified structural and relational attributes. Perhaps the most significant shortcoming of the EER model in representing knowledge about a domain is its inability to capture knowledge about how things change over time. Without a notion of behavior, the EER model is also unable to capture temporal knowledge.

A further construct of the EER model allows entity sets to be arranged in a hierarchy. Two cases are recognized. A *generalization* hierarchy partitions an entity set into a collection of mutually exclusive and collectively exhaustive subsets. An example would be **LOAN** and **ACCOUNT** as subclasses of **PRODUCT** in a banking environment in which these are the only products offered. A *subset* hierarchy produces potentially overlapping subsets of an entity set. An example would be **CUSTOMER** and **EMPLOYEE** as subclasses of **PERSON** in a context in which employees may be customers. In the discussion of Teorey et al. (1986), no mention is made of a more general *lattice* (versus hierarchy) of entity sets. Also, the model does not indicate that specializations must possess additional attributes or relationships.

The MIMIC model contains a more refined mechanism for capturing IS-A associations. Intensionally, a class is a *specialization* of another if it possesses one or more additional structural/relational attributes, whose shared domain is a subset of the extension of the existing class. Consequently, the specialized class has an extension which is a proper subset of that of the superclass. For example, **SAVING** may be a subclass of **ACCOUNT** with the added structural attribute **Interest_rate**. Due to the nature of class definition in terms of attributes, a class may be a specialization of several superclasses. This leads to a lattice structure for classes. For instance, **CHEQUING** may also be a subclass of **ACCOUNT** (with added attributes such as **Issued_cheques**), and **CHEQUABLE_SAVING** a subclass of both **CHEQUING** and **SAVING** with additional attributes with respect to both (such as **Minimum_balance**). If there are no added structural or relational attributes (e.g., if there are only differences in behavior between **CHEQUABLE_SAVING**, and **CHEQUING** or **SAVING** accounts), **CHEQUABLE_SAVING** would not be defined as a class in MIMIC. Instead, differences in behavior depending only on the state of an object (as opposed to the attributes it possesses) can be incorporated into the behavioral attributes of **CHEQUING** or **SAVING** as needed (Section 5.2).

In addition, the disjointness of subclasses is captured implicitly in MIMIC if the added attributes which distinguish two or more subclasses of an existing class (e.g., **LOAN** and **ACCOUNT** as subclasses of **PRODUCT**) have disjoint domains. The subclasses are collectively exhaustive if the union of the domains of the additional attributes is a superset of the extension of the superclass. This contrasts with the explicit mechanism provided in EER. Furthermore, MIMIC recognizes criteria on which objects may move between classes in a lattice (Section 5.4). EER does not provide a mechanism to recognize that entities may move between entity sets.

In sum, EER appears to lack the ability to express the following elements of concept-based knowledge. First, there are no mechanisms to capture behavioral knowledge or to deal with time. Second, there is an ambiguity with respect to modelling relational attributes and composites. Third, there is no intensional notion of a class. Fourth, there are no criteria by which a collection of classes may be judged "good" (i.e., no class structure). Finally, the semantics of specialization are less precise in EER than in MIMIC. Consequently, the EER model may be viewed as a "strict subset" of MIMIC. It does not appear to possess any representational elements which MIMIC doesn't, and does not capture some important elements of knowledge. MIMIC also serves as a higher level framework within which the capabilities and limitations of EER can better be understood in a well-defined way.

6.4 TELOS (Mylopoulos, 1991; Mylopoulos et al., 1990)

The Taxis project has produced several languages and tools to model large-scale, data-intensive applications from a knowledge representation perspective (Nixon, 1984 contains a collection of early papers on various aspects of the project). The project has evolved from a database description language called Taxis, through a non-procedural conceptual modelling language, RML (Requirements Modelling Language), to a more abstract knowledge representation language called Telos (Mylopoulos, 1991)[121]. There are many similarities among these languages, along with some important differences (see Section 2.2.1). In comparing this work to MIMIC, attention is focused on Telos, since this language is most recent and, more importantly, is explicitly intended for conceptual modelling.

One of the fundamental constructs in Telos is the *token*. Tokens are intended to be in direct correspondence with, and therefore surrogates of, things in a domain of interest[122] (Mylopoulos, 1991; Borgida, 1984; Greenspan and Mylopoulos, 1984). An example would be the symbol *John* designating

[121]In addition, CML (Conceptual Modelling Language) had a brief existence as either a refined RML or a preliminary Telos (Borgida et al., 1988).

[122]Using MIMIC terminology, it is more accurate to say that tokens represent *knowledge of* the existence of things in the world. However, this distinction is not explicitly made in descriptions of RML or Telos.

a specific person. In MIMIC, *objects* are surrogates which represent the existence of instances of concepts. Tokens and objects, therefore, have corresponding purposes in the respective languages.

The second fundamental construct in Telos is the *attribute*[123]. Attributes represent binary relationships between entities and, therefore, consist of binary relationships between tokens, such as [John, **Weight**, 70][124]. Attributes belong to attribute categories, such as characteristic and association. Characteristic properties are unchanging (e.g., Birthdate), whereas association properties may change over time (e.g., Address).

Attributes appear to substitute for two constructs in MIMIC: structural and relational attributes. Structural attributes link objects with values. Relational attributes link objects to sets of other objects. Values are not objects as they do not possess attributes. Both structural and relational attributes in MIMIC may be, in Telos terminology, characteristic or association. This is implicitly reflected by whether an attribute contains a single function (characteristic/unchanging) or more than one function (association/changing). Telos does not distinguish structural and relational attributes, since both objects (e.g., *John*) and values (e.g., 70 kg) in MIMIC are treated as tokens in Telos. Therefore, Telos does not capture a distinction between things (which possess properties) and values (which do not).

Telos tokens belong to *classes*. A class is defined in terms of the attributes which are applicable to all instances and is, therefore, an intension. In MIMIC, objects similarly belong to classes. A MIMIC class is an intension, or specification of conditions for membership, consisting of sets of structural, relational, and behavioral attributes. Behavioral attributes impose constraints on the allowed changes to structural and relational attributes for the objects possessing these attributes. Since Telos does not distinguish structural and relational attributes and, furthermore, does not encapsulate behavioral knowledge

[123]In Telos, all constructs are expressed as *propositions*. However, the discussion here deals with specific "kinds" of propositions, such as tokens and attributes.

[124]This proposition is an instance of the proposition [PERSON, **Weight**, WEIGHT]. Consequently, **Weight** may be viewed as a function from a class of tokens, to another class of tokens.

in class definitions[125], the correspondence between Telos and MIMIC notions of class is not precise. Furthermore, Telos does not provide criteria for evaluating a collection of classes as good. By contrast, the notion of class structure is integral to the definition of the class construct in MIMIC.

Classes in Telos are organized in a specialization/generalization lattice. Specialized classes refine one or more existing classes and may add attributes and/or restrict the range of values which attributes of the superclass(es) may assume. A specialized class inherits the attributes of the classes from which it is derived, and every instance of a class is an instance of each of its superclasses. An example would be **PERSON** with attributes (**Name, Address, Birthdate,** ...) and a subclass **CUSTOMER** which inherits **Name, Address,** and **Birthdate**, and may possess other attributes, such as **Holds_account**. Although it appears that tokens may move between classes in Telos, the model does not provide a taxonomy of "kinds" of subclasses for understanding when movements between classes may or may not occur (cf. Section 5.4).

Classes in MIMIC are also organized in a lattice. However, the definition of class structure ensures that a subclass can only be defined by adding attributes to an existing class. For the example above, the subclass **CUSTOMER** must possess at least one additional structural/relational attribute such as the relational attribute **Holds_account**, linking **CUSTOMER** to **ACCOUNT**. As in Telos, a MIMIC subclass may have several superclasses. Furthermore, the extension of a subclass is a proper subset of the extension of each of its superclasses. In addition, if the subclass has more than one superclass (e.g., **CUSTOMER_EMPLOYEE** as a subclass of both **CUSTOMER** and **EMPLOYEE**), it must possess at least one structural or relational attribute with respect to the union of the structural and relational attributes of all superclasses (Section 5.3). Telos does not appear to enforce this constraint.

An important construct of MIMIC which does not appear to be supported by Telos is *composition.* While a Telos object is an aggregation of attributes whose values are, in turn, objects which possess

[125]Behavior is treated separately in Telos, as discussed later.

further attributes, this collection of objects is not treated as a whole, which is linked by some relational attribute and possesses *emergent* attributes.

Earlier it was pointed out that Telos does not encapsulate behavioral attributes in class definitions. Instead, a class/instance framework is used to deal with change[126]. Knowledge about change (behavior) is expressed in terms of *activities* which belong to classes. Activity classes are a mechanism for describing change, including the addition of tokens to and the removal of tokens from classes, as well as changes to the values of properties of tokens. Activity classes are defined in terms of *inputs* and *outputs* of the activity (e.g., **PERSON** and **CUSTOMER** for an activity in which a person becomes a customer), *pre-* and *post-conditions* which must be satisfied before and after the activity occurs (e.g., the potential customer must be in class **PERSON** before the activity and must have the property **Holds_account** after the activity), and *parts* which describe the actual changes at a general level (e.g., open an account). Parts describe the "what" rather than the "how" of change. Each activity is an instance of an activity class. Activity classes are arranged in a specialization hierarchy with inheritance. For example, **OPEN_CHEQUING_ACCOUNT** may be an activity class specializing the class **OPEN_ACCOUNT** with new pre-conditions (e.g., customer must already hold a savings account) and parts (e.g., issue cheques to accounts).

In contrast, MIMIC integrates behavior as part of the description of knowledge about things through *behavioral attributes*. A behavioral attribute specifies the changes which are permitted to the active function of one or more structural/relational attributes given any combination of active functions for these attributes. At first glance, there seems to be little connection between behavioral attributes and activities. However, since pre- and post-conditions may be used to describe states necessary for an activity

[126]Telos has been referred to as *object-oriented* (Mylopoulos et al., 1990). However, the framework does not include the most commonly used indicator of object orientation: *encapsulation* (Nierstrasz, 1987), since structural and behavioral knowledge about classes of entities are handled through separate mechanisms.

(pre-conditions) and constraints on the states which may result from an activity (post-conditions), they are analogous to the input to (i.e., a function) and output from (i.e., a set of functions) a behavioral attribute function in MIMIC. The parts of an activity also describe, in a non-procedural way, the nature of the changes which occur. Nevertheless, MIMIC behavioral attributes are more abstract than Telos activities.

Closely related to behavior is the treatment of time in each model. The model of time in Telos is based on time intervals (Allen, 1983). A time interval is associated with each proposition (token or attribute) indicating a duration for which the proposition holds. MIMIC handles time only implicitly through the occurrence of events. That is, the passage of time is recognized by events, and behavioral attributes impose constraints on which events are allowed. By introducing a "clock" object, the time of events relative to a standard enables comparisons of the distance between events to be made, and for the duration of states to be determined (Section 5.7).

One important feature of Telos is not included in the MIMIC model. Specifically, Telos classes may also be instances of other classes - the latter termed *metaclasses*. This "uniformity" is one of the most emphasized features of the model (Mylopoulos et al., 1990). For example, a metaclass **PERSON_CLASS** may include instances, such as **PERSON, EMPLOYEE** and **CUSTOMER**, which are classes. This allows, for example, statistical and aggregate information to be attached to classes (e.g., **Average_age** of **EMPLOYEE, CUSTOMER,** and other instances of **PERSON_CLASS**)[127]. There is deliberately no counterpart to this feature in MIMIC, since the model is intended only for representing knowledge about the subject matter of an information system. Meta-classes represent "knowledge about knowledge" about the domain. Considering classes as objects (and, therefore, members of metaclasses) is appropriate only if what is being modelled includes knowledge about knowledge. In other words, MIMIC holds that concepts and instances of concepts differ, and need to be appropriately distinguished in the model by

[127]At least some of the information that is captured by the properties of metaclasses is implicitly carried by the objects of a class. For example, **Average_age** is implied by (and can be computed given) the ages of each instance of a class.

differentiating classes from objects. This important difference between the models may be due to the larger scope of Telos, which is intended not only for modelling the subject domain (e.g., knowledge about entities in an organization), but for the information system, usage, and development domains (Mylopoulos et al., 1990).

To summarize, Telos supports many of the elements of the cognitive model of MIMIC. However, a few features are not supported and Telos contains one very significant departure from the MIMIC model. Like MIMIC, Telos supports the representation a domain in terms of instances which belong to classes that are defined intensionally. Similarly, it supports class specialization and inheritance. In addition, both structural and behavioral knowledge can be modelled, although in a manner different from that of MIMIC. Furthermore, the importance of time is recognized.

Telos does not, however, introduce the notion of class structure - a collection of classes which satisfy certain cognitive principles. In addition, it does not use a uniform, function-based approach to the description of structure, relationships, and behavior. Structural and behavioral knowledge is represented in separate class hierarchies, and structural and relational attributes are not distinguished. Telos also adopts a quite different approach than MIMIC for representing behavior, using pre-conditions, parts, and post-conditions. Consequently, behavior modelling is more detailed than in MIMIC, in which behavior is specified only in terms of which events may occur, given a particular state (with respect to one or more structural/relational attributes). In addition, Telos does not have a mechanism to model composite things, which consist of parts and possess emergent properties.

The most significant difference between the models is that Telos stresses uniformity in the use of propositions and multiple levels of classification. Telos views classes as objects and introduces metaclasses to abstract the commonality of classes. MIMIC does not support this (although it could be extended to do so)[128] because of the stance that the primary domain to be modelled does not include knowledge about knowledge and, therefore, classes are to be treated differently from objects.

6.5 NIAM (Verheijen and Van Bekkum, 1982)

NIAM is a language (primarily graphical) for representing knowledge about a domain, termed the *object system*. Knowledge is captured in terms of *information structure* and *information flow*, corresponding roughly to static and dynamic knowledge. In this section, the features of NIAM are examined to determine the model's degree of support for representing knowledge constructs.

The simplest construct in NIAM is the *object*. Objects are of two kinds: non-lexical and lexical. A non-lexical object is a thing in the domain of interest (e.g. the person *john_smith*)[129,130]. A lexical object is a string which can be spoken and refers to a thing in the domain of interest (e.g., "John Smith"). Both non-lexical and lexical objects are classified into *types*. A *non-lexical object type* (NOLOT) is a set of non-lexical objects (e.g., PERSON). A *lexical object type* (LOT) is a set of lexical objects (e.g., PERSON_NAME).

The non-lexical/lexical object distinction does not correspond well to the object construct of MIMIC. Some non-lexical objects, such as symbols denoting persons, would be modelled as objects in MIMIC. On the other hand, other non-lexical objects, such as symbols designating dates, would be modelled as values in MIMIC. Furthermore, lexical objects, such as strings designating names, also correspond to MIMIC values. In short, NIAM does not clearly support the MIMIC object construct. To

[128]For example, classes could constitute the domain of other structural and relational attributes. Collections of attribute with such domains could then form metaclasses.

[129]Actually, a non-lexical object in a NIAM description of a domain is a representation of a thing, and not the thing itself.

[130]In this discussion, non-lexical objects are indicated by words in italics to distinguish them from lexical objects, which are enclosed in double quotes.

serve as a more direct model of knowledge, non-lexical objects must be used only to represent instances of concepts (thereby corresponding to MIMIC objects) and lexical objects must be used only to represent values.

The essence of the information structure component of NIAM is a collection of LOTs and NOLOTs linked by *bridge* and *idea* types. A bridge type denotes a binary association between a NOLOT and a LOT (e.g., **PERSON Has_name PERSON_NAME**). An idea type denotes a binary association between two NOLOTs (e.g., **EMPLOYEE Employed_by EMPLOYER**). Bridges and ideas are members of these types (e.g., *john_smith* **Has_name** "John Smith", *john_smith* **Employed_by** *ubc*).

If lexical and non-lexical objects (and types) are constrained to represent values and instances of concepts, respectively, bridge types can be viewed as serving a similar purpose as structural attributes in MIMIC, since a bridge type links a NOLOT with a LOT. Idea types may be viewed as similar to binary relational attributes since an idea type links two NOLOTs. However, *n-ary* relational attributes, linking objects of one class with objects of several other classes, are not supported by NIAM.

NIAM does not fully support the MIMIC class construct, since a NOLOT is substantially different from a MIMIC class. A NOLOT is a set of non-lexical objects (e.g., **EMPLOYEE** = {*John Mary Jane*}), and is not defined intensionally. Even if the collection of idea and bridge types involving a NOLOT is taken as a definition of the NOLOT (which is not done in NIAM), behavioral knowledge is not encapsulated, but is dealt with using other mechanisms (which are discussed later). Furthermore, NIAM does not provide a mechanism for evaluating the quality of a collection of NOLOTs.

NIAM does support subtype links between NOLOTs. Subtypes inherit idea and bridge types of supertypes and may (not must) be involved in additional bridge and idea types. It is unclear, however, whether a NIAM subtype can have multiple supertypes. Furthermore, NIAM does not seem to account for the movement of objects between types. Subtype links serve a similar function as subclasses in MIMIC. However, the MIMIC subclass construct has clearer semantics, in that a subclass must possess added attributes and its extension is a strict subset of that of (each of) its superclass(es).

NIAM does not support a notion of composite NOLOTs having emergent properties and, therefore, does not have a mechanism corresponding to composition in the MIMIC model.

Several kinds of constraints on information structure diagrams are introduced in NIAM. *Identifier constraints* are similar to cardinality constraints of the ER model. They determine whether a member of one type can be linked to one or more members of another type via a bridge or idea. The MIMIC model captures this information in two ways. First, a structural attribute always links an object to a single value. A relational attribute may link an object to several sets of one or more objects (from the same or different classes). If the codomain is restricted to a set of singletons, a 1-1 mapping is implicitly determined. For example, the codomain of a relational attribute, **Married_to**, can be constrained to contain only singletons. Other conditions can equally well be handled (e.g., pairs, singletons and pairs, and so on). For example, the codomain of **Child_of** can be constrained to contain only pairs.

Subset constraints impose the condition that the objects involved in one idea or bridge type are a subset of those involved in another. For example, if **PAPER Presented_by AUTHOR** and **PAPER Written_by AUTHOR**, then for any paper, its presenters must be a subset of its authors. MIMIC does not have a mechanism to explicitly enforce this, since it constitutes knowledge about knowledge about the subject matter. In other words, knowledge about the subject matter contains specific details about the authors and presenters of papers. That this conforms to the constraint "All presenters of a paper are authors of that paper" is an abstraction from the data, which can be recognized from a MIMIC representation, but is not enforced as an explicit constraint of the model.

Equality constraints impose the condition that the population of objects in one binary association is the same as that of another involving the same two object types. For example, if **Starts_on** and **Ends_on** associate **CONFERENCE**s with **DATE**s, a given conference must have both (or neither). This is not explicitly dealt with in MIMIC, but any two (or more) attributes (e.g., structural) must have values (i.e., the model does not deal with incomplete information).

Uniqueness constraints deal with a combination of role occurrences which uniquely identify a non-lexical object. For example, a customer is uniquely identified by any account which s/he holds, if an account is held by no more than one customer. This is related to the concept of a key in relational databases. In MIMIC, an object in a class is uniquely identified by a surrogate. It may also be that some combination of values of attributes also distinguishes an object from all others, but this is not relevant to the purpose of the model, since implementation considerations are beyond its scope.

Disjoint constraints are used to express the knowledge that two subtypes of an object type are mutually exclusive (e.g., subtypes **ACCOUNT** and **LOAN** of **CUSTOMER**). In MIMIC, two subclasses of a class are disjoint if the intersection of the common domain of the additional attributes of one with the common domain of the additional attributes of the other is an empty set. That is, the restriction is imposed (i.e., can be recognized and enforced in a representation) implicitly by the extension of classes and not by any explicit rule, since this again constitutes knowledge about knowledge.

Total role constraints enforce that every object of a certain type participates in a given role in a binary relationship. Essentially this is similar to a prohibition of null values for an attribute or relationship. In MIMIC, the class definitions are such that no member of a class has a null (or not applicable) value for a structural attribute[131]. Null values are, in general, permitted for relational attributes (e.g., **Married_to**), but this can be constrained for particular relational attributes by omitting the null set from the codomain of a relational attribute if all objects in a class are to be linked to other objects via the attribute (e.g., **Child_of**).

The interesting thing about most of these constraints, and their treatment in NIAM versus MIMIC, is that they reflect knowledge about knowledge about specific things. Like Telos, NIAM captures some

[131]Otherwise, either the attribute is not truly part of the class definition or the object in question does not belong in the class. Values may be unknown, but MIMIC does not address representing incomplete knowledge.

metaknowledge by adding explicit constraints. However, MIMIC explicitly deals only with representing base knowledge about things in a domain (where the domain does not include knowledge about knowledge). Constraints such as those supported by NIAM are, however, recognized implicitly, in that they can be satisfied in a specific representation. For example, a problem description for a bank contains the information that accounts and loans are non-overlapping, since the shared domain of the extra attributes of each class is disjoint with that of the other (e.g., accounts do not have the attribute **Term**, and loans do not have the attribute **Overdraft**).

A second component of the NIAM model deals with change. This is handled in a process-oriented manner through *functions* and *information flows*. Information flows contain information on the objects of an information structure diagram. Functions transform information flows and, consequently, change the state of a system. An information flow diagram contains a number of functions, an information base (which is based on an information structure diagram), and a number of information flows (streams of data).

Although it is impossible to formally compare this view of behavior to the way in which MIMIC deals with change, it is clear that the NIAM model is much closer to traditional data processing (implementation or software-oriented) views of information systems. Information flow diagrams are very similar to the data flow diagrams of structured analysis (DeMarco, 1979)[132]. The view of change in MIMIC, on the other hand, is completely non-procedural. Behavioral attributes describe the events which may occur, with no attention to how changes are carried out. The behavioral component of NIAM is perhaps more useful as a system design tool than a knowledge modelling tool, since NIAM functions and information bases have direct counterparts in software (programs and files).

In sum, NIAM support for the representation of most cognitive constructs (as modelled by

[132]This can be seen by comparing *information base* with *data store*, *function* with *transform*, *information flow* with *data flow*, *environment* with *source/sink*.

MIMIC) appears to be lacking. First, there is no direct counterpart to the MIMIC object. Second, there is no distinction between structural and relational attributes. Third, behavior is modelled procedurally. Fourth, the notion of a class as an intension encapsulating structure, relationships, and behavior, is missing. Fifth, the issue of the "quality" of a set of NOLOTs is ignored. Sixth, the notion of specialization is not as precise as that of MIMIC. Finally, composition is not supported.

6.6 OBJECT-ORIENTED ANALYSIS (Coad and Yourdon, 1991)

Object-oriented analysis (OOA) has emerged as a modelling technique said to be based on principles guiding the way people organize knowledge (Booch, 1991; Coad and Yourdon, 1991). Unfortunately, there is not a standard approach to object-oriented modelling. In this section, the practical method advocated by Coad and Yourdon (1991) is analyzed in terms of its support for the concept theory based constructs of MIMIC.

The basic construct in OOA is the *object*, which is a collection of attributes and services[133]. Objects correspond to things in the subject domain, including "structures, other systems, devices, things or events remembered, roles played, operational procedures, sites, and organizational units" (p. 60). Without going into the details of each, it should nonetheless be clear that the object construct is sufficiently general to represent both physical things (e.g., structures, devices) and abstract entities (e.g., events remembered, organizational units). Thus, an OOA object is very similar to a MIMIC object. However, MIMIC uses objects to explicitly represent the existence of instances of concepts as distinct from the properties possessed by these instances. OOA does not appear to make this distinction.

Objects in OOA possess *attributes*. According to Coad and Yourdon, "an attribute is some data (state information) for which an Object in a Class has its own value" (1991, p.119). Attributes are single-

[133]*Service* is a synonym for method or activity, and is a mechanism for describing behavior (changes to attribute values), as discussed later.

valued, and resemble structural attributes in MIMIC. However, unlike structural attributes, OOA neither formally defines the construct nor specifies whether values are objects.

OOA also introduces a notion called *instance connection*, which consists of a mapping (along with cardinalities) from an object to other objects. This construct is analogous to the ***relational attribute*** of MIMIC, which maps objects to sets of other objects. However, relational attributes may be n-ary (i.e., connect attributes from **n** sets), whereas instance connections are strictly binary. In addition, the importance of instance connections to the definition of classes in OOA is not specified.

Behavior is modelled in OOA by what is called a *service.* A service describes a mechanism by which objects change state. The description of services is highly procedural (algorithmic). Hence, unlike the abstract descriptions of allowed change that constitute MIMIC behavioral attributes, services describe <u>how</u> objects change state (not simply which changes may occur).

Attributes and services together are used to define *classes* in OOA. A class may be viewed as a template, or abstraction of a set of objects sharing a collection of attributes and services. As in MIMIC, a class is intensionally defined, has an associated extension, and encapsulates structure and behavior. However, part of the class description includes instructions for creating new instances. This use of classes to create instances is borrowed from Smalltalk (Goldberg and Robson, 1989), and is very much an implementation consideration. Furthermore, OOA does not develop a notion comparable to the MIMIC class structure, for assessing the quality of a collection of classes.

OOA also introduces two *structures*[134] for organizing classes. The ***generalization-specialization*** structure captures IS-A linkages between classes. A specialized class inherits the attributes and services of the more general classes from which it descends. Thus, a lattice of classes is supported. This notion is similar to the specialization construct in MIMIC. However, MIMIC offers different criteria for defining subclasses. OOA generally requires that a specialized class add attributes <u>or</u> services, <u>except</u> when it

[134]This term is not used in the same sense as *class structure* in MIMIC.

specializes two or more existing classes. As discussed in Chapter 5, MIMIC does not allow specialization based solely on added behavioral attributes. Furthermore, a subclass derived from two or more existing classes not only inherits attributes of both, it must possess additional ones. Finally, MIMIC considers conditions under which objects may move between superclasses and subclasses, while OOA does not.

The final construct in OOA is the *whole-part* structure[135]. A whole object (or class) is composed of several parts. OOA allows for optional and mandatory parts. Three variations - assembly/parts, container/contents, and collection/members - are used. It is not clear whether the whole must possess emergent attributes and services (although the examples used suggest this). The whole-part construct clearly parallels the composite class (or object) in MIMIC. However, MIMIC provides a precise construct, defined in terms of emergent attributes, and further requires that the components be linked by relational attributes.

In sum, OOA provides some support for almost all of the cognitive constructs represented in MIMIC. The notable omission is the notion of a class structure which meets certain conditions of goodness. However, the constructs provided are not precisely defined in OOA, and some MIMIC constructs, such as relational attributes, specialization, and composition, capture more knowledge than their counterparts in OOA. Furthermore, the treatment of behavior in MIMIC is more abstract (i.e., only in terms of allowed state changes) than in OOA, since the latter introduces algorithmic specification of how changes occur. A consequence of the precision of MIMIC is that time can be handled formally, while the subject is not treated at all in OOA.

MIMIC may be viewed as a higher-level, formally defined companion of OOA, giving precise meaning to constructs that are only casually introduced in OOA. This is not a criticism of OOA. On the contrary, a casual description is probably very useful for a practical methodology that is intended to be

[135]OOA also discusses a term called *subjects* as a mechanism for organizing a large model into segments. However, this does not add to the knowledge content of a representation, but merely its presentation, and is not discussed further here.

directly used in building domain descriptions.

6.7 SUMMARY

This chapter has used MIMIC to evaluate the "cognitive content" of four other conceptual models. The examination highlights the difficulty of reconciling terminology and construct differences - a prerequisite for comparing and evaluating different approaches. Direct comparison was, in some cases, impossible, and otherwise required assumptions to be made about the meaning of constructs in the other models.

Nevertheless, the basic conclusion of the analysis is that EER, Telos, NIAM, and OOA *support to varying degrees* the elements of concept theory on which MIMIC is based. Both EER and NIAM have significant gaps in representing important knowledge. EER fails to incorporate behavioral modelling, while NIAM does not distinguish structural and relational knowledge, does not encapsulate behavior with structure, has only a weak notion of specialization, and does not handle composition. Neither EER nor NIAM deals with time. Telos captures many of the important elements of concept theory, but does not appear to support either encapsulation or composition. Furthermore, it introduces metaclasses to model knowledge about knowledge, a task beyond the intended scope of MIMIC[136]. OOA deals to some extent with each of the concept theory-based constructs of MIMIC, but in a less precise manner. As a consequence of the lack of formal treatment in OOA, it fails to capture some important semantics that are reflected in MIMIC, such as the meaning of class specialization, and the nature of time. In short, the comparison demonstrates that MIMIC may be viewed as a framework, or *metamodel* within which the similarities and differences of other models can be better understood.

Perhaps the most telling limitation of the models surveyed here is that none consider the *quality of a class structure*. One of the primary contributions of MIMIC is the use of several basic "principles of

---

[136]NIAM also represents knowledge about knowledge, but to a much lesser degree.

conceptualization" to derive a set of minimal conditions for a given collection of (potential) classes to be "good". The conditions which define *potential class* and *class structure* allow knowledge about a domain to be represented in an effective (complete) and efficient (non-redundant) manner, while recognizing that many organizations of classes for a given domain of knowledge may satisfy these conditions.

Finally, MIMIC provides a simple and implicit view of time that does not require the introduction of additional constructs or a complex logic of time. This is not the case for any of the other models considered, since Telos introduces a detailed temporal logic, while the other models examined do not deal with time. While the simplicity may limit the expressiveness of the model for dealing with time, it does focus on and operationalize the single notion of time as change (specifically, as change in active functions of structural and relational attributes).

CHAPTER 7

CONCLUSIONS AND FURTHER RESEARCH

7.1 THESIS SUMMARY

Conceptual modelling is a critical early activity in information systems development, during which users' knowledge about the things which constitute the subject matter of a proposed information system is represented in an implementation-independent manner. Unfortunately, many existing systems analysis methods (and conceptual models) are not explicitly based on theories about the ways in which people structure and organize knowledge about things. Instead, many contain constructs which are more suitable for modelling the implementation or software domain. This has contributed to problems of incomprehensibility, redundancy, and duplication of effort in systems development.

This thesis helps fill the conceptual modelling "gap" by developing a conceptual information systems model based on a theory of *concepts* (or classification). More specifically, the work attempts to answer the following general research questions (Chapter 1):

1. ***Can a theory of concepts serve as a foundation for defining a uniform model of knowledge that supports direct representation across several kinds of IS applications?***
2. ***Does this model offer insights and guidelines for conceptual modelling relative to other conceptual models?***

These questions were, in turn, operationalized in terms of three basic research objectives:

1. *Define the model,*
2. *Demonstrate the model's value as a tool for exploring open problems in conceptual modelling,*
3. *Evaluate the model relative to several existing conceptual modelling approaches.*

To achieve these objectives, a theory of concepts - the so-called *classical* view - was first described in detail. Five fundamental constructs - instance, property, concept, specialization, and composition - were identified (Chapter 3) and defined formally (Chapter 4). This formalism, along with the objective of providing direct representation, provided the basis for defining a set of corresponding conceptual modelling constructs, resulting in the MIMIC model (Chapter 4). Use of these constructs on a hypothetical, but representative, problem domain served to highlight how the model may be used and its uniformity in representing knowledge for several kinds of applications (Appendix 1). Subsequently, the contributions of the model to resolving several open problems in conceptual modelling were explored (Chapter 5). This was followed by evaluating the constructs of four existing conceptual models with respect to those of MIMIC. The comparison indicated the model's generality and representation power and provided a basis for understanding and comparing the constructs of other models (Chapter 6).

7.2 CONTRIBUTIONS

This research has been directed toward developing a classification theory based, formal conceptual model. Hence, the most significant contributions are theoretical. However, the work has also yielded a number of practical contributions, primarily in the form of implications for conceptual modelling activities and methodologies.

7.2.1 Theoretical Contributions

As discussed in Chapter 2, previous conceptual modelling research and practice has produced many methodologies with varying "ad hoc" constructs which have not been adequately motivated or demonstrated to be useful. The primary contribution of this work is the development of MIMIC - a uniform, formal model of knowledge about things[137]. Unlike many existing models, MIMIC is *explicitly* and *formally* based on a theory of the structure and organization of knowledge about entities, and is intended for modelling knowledge about the subject matter of an information system. As a result, a number of insights into conceptual modelling semantics are gained.

The model introduces the notion of *potential class*, based on the principles of *abstraction from instances* and *maximal abstraction*, and of *class structure* - a collection of potential classes satisfying the principles of *non-redundancy* and *completeness.* A class structure is a complete representation some universe of knowledge, in which no class is simply a combination of other classes in the structure. Different views of the world can be supported by defining different class structures on a given set of objects and attributes. An "asymmetry" between structural/relational versus behavioral attributes is identified. The former provide the basis for defining classes, while allowed changes are implied by the behavioral attributes . The naturalness of a lattice class organization is explained, and the notion of *roles* within a class structure is supported by the membership of objects in several classes. The meaning of IS-A relationships is refined in terms of the temporal or permanent nature of attributes. Guidelines are given for modelling phenomena as either relational attributes or as composites. In addition, the merits of uniformity in modelling structure and behavior are discussed. Furthermore, the model's position on the meaning of time, and a simple approach to modelling time are explored.

An additional theoretical contribution comes in the form of insights into the ongoing debate over

[137] In order to develop this model, the constructs of classical concept theory had first to be formalized. In the process, a number of potential refinements to the theory emerged. The notion of a lattice of classes is a natural consequence of the formalization of properties, but is not traditionally associated with the classical view. The definitions of potential concept and concept structure use a few basic assumptions about the purposes of classification to contribute to an understanding of why certain concepts are formed instead of others. The notion of class structure further supports alternate conceptualizations of the world. The inability of the classical view to support alternate world views has been a criticism recently levelled against the theory (Lakoff, 1987). These insights are viewed as contributions to the reference discipline, but are not discussed further here since they go beyond the focus of the thesis.

the meaning of *objects*. Existing research has provided diverse, and sometimes conflicting, views on what the object paradigm is or should be. MIMIC is offered as an "object model", and may contribute to resolving the debate by highlighting that traditional justifications of the elements of the object paradigm have mixed *implementation* and *representation* advantages. MIMIC provides a model of objects that is based solely on the objective of directly representing knowledge about things.

A third theoretical contribution, which also has practical implications, emerges from the demonstration of the model's applicability by using it to uniformly represent knowledge for three kinds of applications (transaction processing, reasoning, and simulation) involving the same things. The example demonstrates that a single modelling framework can be used to develop integrated representations: that is, representations in which a single thing is modelled by one object which participates in several classes of applications. This suggests that reductions in redundancy and duplication of development effort can be achieved using the model.

Finally, the model provides a foundation for a program of future research. Although this might not generally be considered a contribution of a piece of research, in this case the model clearly identifies a number of important related research issues, and provides a clear foundation for being able to address them properly. This is considered further in the last section.

7.2.2 Practical Contributions

An important practical contribution of this research emerges from the insights afforded by the model. A number of guidelines for conceptual modelling are provided which may form the basis for a methodology for eliciting knowledge. For example, a specialization of an existing class must possess additional structural or relational attributes. Therefore, attempts to define subclasses on the basis of either values of attributes or behavioral attributes can be caught and corrected. In addition, identification of instances and attributes may be a useful complement to a more traditional strategy of identifying classes, since the latter may be more volatile. Furthermore, refinements in the meaning of IS-A relationships allow more knowledge to be captured in defining specialized classes. Finally, relational attributes can serve as a signal to look for information about the association, leading to the definition of composite classes. These modelling prescriptions may prove useful in developing representations which are more easily understood by users[138].

A second practical contribution is that MIMIC serves as a metamodel against which other conceptual models can be compared (cf. Sol, 1983). This contributes to an understanding of the strengths and weaknesses of other methodologies from a cognitive point of view, and thereby, can support the selection of a methodology for a given application.

7.3 LIMITATIONS

The boundaries of this research are clearly marked by two factors. The first is the scope of the theory of concepts on which the model is based, and the second is the focus of the model on representing knowledge at the complete exclusion of considerations related to the implementation of software and data. These factors impose a number of limitations on the work.

First, by focusing on concept theory, other kinds of "knowledge" or cognitive structures, such as beliefs and goals, are ignored. Consequently, concept theory does not provide a complete theory of knowledge and the model developed is, in its current form, not applicable as a general knowledge representation language. For example, the model does not recognize incompleteness or ambiguity[139].

[138]Clearly, it remains as future research to demonstrate or refute this hypothesis. This is discussed in the last section.

[139]Consequently, the model is not useful in explaining how people make default assumptions and reason, given uncertain knowledge.

While MIMIC can be extended to accommodate uncertainty[140], the decision not to do so was based on the fact that the chosen theory of concepts did not warrant its inclusion. Within the scope of applications of interest, this limitation does not impair the intended usefulness of the model.

Second, MIMIC does not capture *metaknowledge* (knowledge about knowledge). By contrast with models such as Telos (Mylopoulos, 1991; Mylopoulos et al., 1990), MIMIC supports only the representation of what can be called *base* knowledge about things in a domain. Again, the model could be extended to capture metaknowledge[141]: however, such an extension could not be justified using concept theory. Furthermore, since the domain of interest is the subject matter of a proposed information system (e.g., an organization), the inclusion of metaknowledge would not contribute to directly representing knowledge about the *things* in that domain.

Third, the model is far removed from implementation concerns. The most obvious manifestation of this is the completely non-procedural treatment of behavioral properties. Consequently, it is not clear how the functional description of behavioral constraints should be transformed to a procedural design and implementation. Similarly, the explicit implementation of structural and relational attributes as "large" sets of functions is not practical. However, these issues open the door to one path of further research involving a direct implementation of the model, as discussed in the next section.

Finally, the model is strongly shaped by the particular theory of concepts in which it is grounded. A model based on the quite different prototype- or exemplar-based theories would differ in important ways from MIMIC and might, for example, lead to different conceptual modelling guidelines. Since such models have not yet been proposed, it remains to be seen how they would compare with MIMIC.

---

[140]As an illustration, structural properties could be defined to map from domains of objects to codomains containing value-certainty pairs. For example, the property **Weight** could be defined as $\{w_i | w_i\text{:PERSON} \rightarrow \text{WEIGHT} \otimes \text{CERTAINTY}\}$, where CERTAINTY = $\{0, \ldots, 100\}$ and indicates the degree of certainty (or belief) one has about a fact.

[141]For example, classes could be treated as objects which, in turn, possess attributes, and so on.

7.4 DIRECTIONS FOR FUTURE RESEARCH

One of the contributions of this thesis is the provision of a formal, grounded basis for a significant program of future research which builds on the model and the results obtained from it. Three methodologically different, but complementary, research directions are mentioned here.

First, the model provides a vehicle for testing the much-vaunted "naturalness" of the object paradigm. It has been widely claimed that object-oriented approaches more naturally reflect the way people structure and organize knowledge about the world. However, the lack of agreement on what is meant by the term "object" prevents appropriate tests of naturalness. MIMIC is proposed as an "object model" which provides a small set of constructs for representing knowledge according to a specific theory of concepts. Hence, people's ability to understand (or create) representations using these constructs can be evaluated relative to existing conceptual modelling approaches. This work can be done in either an experimental or field setting.

Second, a direct implementation of the MIMIC model may help to reduce the "semantic gap" between conceptual modelling and implementation constructs. Traditionally, systems analysis methodologies have been heavily influenced by software constructs, such as procedural programming languages and data structures. In contrast, a direct implementation of the model would provide implementation primitives based on cognitive constructs. This may offer advantages such as greater ease of learning and using a system. In particular, an implementation may facilitate the creation of end-user developed applications if users are able to build applications using constructs which parallel the way they organize knowledge about things in the domain. Appendix 1 hints at directions for implementation by informally describing a number of operations to build and maintain a representation.

Finally, the model suggests a number of guidelines for performing conceptual modelling: however, these do not yet constitute a coherent methodology or comprehensive set of instructions for using MIMIC to represent knowledge. Such a methodology will be crucial for the success transfer of the model to practical IS development applications. For example, although the notion of class structure provides minimal criteria evaluating the quality of a collection of potential classes, much work remains to be done to provide further criteria for choosing among competing class structures. Similarly, further research is needed to determine a practical way of expressing behavioral knowledge.

BIBLIOGRAPHY

Abbott, R., "Knowledge Abstraction", *Communications of the ACM*, 30(8), August 1987, 664-671.

Abrial, J., "Data Semantics", in J. Klimbie and K. Koffeman (eds.), *Data Management Systems*, Amsterdam: North-Holland, 1974, 1-59.

Allen, J., "Maintaining Knowledge About Temporal Intervals", *Communications of the ACM*, 26(11), 1983, 832-843.

Allen, J., "Towards a General Theory of Action and Time", *Artificial Intelligence*, 23, 1984, 123-154.

Armstrong, S., L. Gleitman and H. Gleitman, "What Some Concepts Might Not Be", *Cognition*, 13, 1983, 263-308.

Backus, J., "Can Programming be Liberated from the von Neumann Style?", A Functional Style and its Algebra", *Communications of the ACM*, 21(8), 1978, 613-641.

Bailin, S., "An Object-oriented Requirements Specification Method", *Communications of the ACM*, 32(5), May 1989, 618-623.

Banerjee, J., H.-T. Chou, J. Garza, W. Kim, D. Woelk, N. Ballou, and H.-J. Kim, "Data Model Issues for Object-oriented Applications", *ACM Transactions on Office Information Systems*, 5(1), January 1987, 3-26.

Barsalou, L., "Context-independent and Context-dependent Information in Concepts", *Memory and Cognition*, 10:1, 1982, 82-93.

Berztiss, A., "The Set-Function Approach to Conceptual Modelling", in (Olle et al., 1986), 107-144.

Best, J., *Cognitive Psychology*, St. Paul, MN: West Publishing, 1986.

Birtwistle, G., O. Dahl, B. Myhrhaug and K. Nygaard, *SIMULA Begin*, Philadelphia:Auerbach Press, 1973.

Bobrow, D. and T. Winograd, "An Overview of KRL, A Knowledge Representation Language", *Cognitive Science*, 1(1), 1977, 3-46.

Booch, G., *Object-oriented Design with Applications*, Redwood City, CA: Benjamin/Cummings, 1991.

Borgida, A., "Features of Languages for the Development of Information Systems at the Conceptual Level", in (Nixon, 1984), 21-57.

Borgida, A., S. Greenspan, and J. Mylopoulos, "Knowledge Representation as the Basis for Requirements Specification", *Computer*, April 1985, 82-90.

Borgida, A., M. Jarke, J. Mylopoulos, J. Schmidt, and Y. Vassiliou, "The Software Development Process as a Knowledge Base Management System", in J. Schmidt and C. Thanos (eds.), *Knowledge Base Management Principles*, Springer-Verlag, 1988.

Bourne, L., B. Ekstrand and R. Dominowski, *The Psychology of Thinking*, Englewood Cliffs: Prentice-Hall, 1976.

Bracchi, G., P. Paolini, and G. Pelagatti, "Binary Logical Associations in Data Modelling", in (Nijssen, 1976), 125-148.

Brachman, K. and J. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science*, 9, 1985, 171-216.

Brachman, R., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks", *Computer*, October 1983, 30-36.

Brandt, I., "A Comparative Study of Information System Design Methodologies", in (Olle et al., 1983), 9-36.

Bretl, R., D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, H. Williams and M. Williams, "The GemStone Data Management System", in (Kim and Lochovsky, 1989), 283-308.

Brodie, M., "On the Development of Data Models", in (Brodie et al., 1984), 19-47.

Brodie, M. and E. Silva, "Active and Passive Component Modelling: ACM/PCM", in (Olle et al., 1982), 41-91.

Brodie, M. and D. Ridjanovic, "On the Design and Specification of Database Transactions", in (Brodie et al., 1984), 277-306.

Brodie, M., D. Ridjanovic, and E. Silva, "On a Framework for Information Systems Design Methodologies", in (Olle et al., 1983), 231-242.

Brodie, M., J. Mylopoulos and J. Schmidt (eds.), *On Conceptual Modelling*, New York: Springer-Verlag, 1984.

Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer*, April 1987, 10-19.

Bruner, J., J. Goodnow and G. Austin, *A Study of Thinking*, Wiley: New York, 1956.

Bubenko, J., "Information System Methodologies - A Research View", in (Olle et al., 1986), 289-318.

Bubenko, J., "Information Modelling in the Context of System Development", in S. Lavington (ed.), *Information Processing 80*, Amsterdam: North-Holland, 1980, 395-411.

Bunge, M., *Treatise on Basic Philosophy (Volume 3), Ontology I: The Furniture of the World*, Boston: Reidel, 1977.

Bunge, M., *Treatise on Basic Philosophy (Volume 4), Ontology II: A World of Systems*, Boston: Reidel, 1979.

Chen, P., "The Entity-Relationship Model: Toward a Unified Model of Data", *ACM Transactions on Database Systems*, 1(1), March 1976, 9-36.

Coad, P. and E. Yourdon, *Object-oriented Analysis*, Englewood Cliffs, NJ: Prentice-Hall, 1991.

Collins, A. and E. Loftus, "A Spreading-Activation Theory of Semantic Processing", *Psychological Review*, 82(6) 1975, 407-428.

Crutchfield, R., "The Creative Process", in M. Bloomberg (ed.), *Creativity: Theory and Research*, New Haven, CT: College and University Press, 1973, 54-74.

Davis, G. and M. Olson, *Management Information Systems: Conceptual Foundations, Structure, and Development*, New York: McGraw-Hill, 1985.

DeMarco, T., *Structured Analysis and System Specification*, New York: Yourdon Press, 1979.

Digitalk, *Smalltalk/V Mac Object-oriented Programming System*, Los Angeles: Digitalk, Inc., 1989.

Fikes, R. and T, Kehler, "The Role of Frame-based Representation in Reasoning", *Communications of the ACM*, 28(9), September 1985, 904-920.

Fishman, D., D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahmoud, M. Neimat, T. Ryan and W. Shan, "Iris: An Object-oriented Database Management System", *ACM Transactions on Office Information Systems*, 5(1), January 1987, 48-69.

Floyd, C., "A Comparative Evaluation of System Development Methods", in (Olle et al., 1986), 19-54.

Gane, C. and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, Englewood Cliffs, NJ: Prentice-Hall, 1979.

Glass, A. and K. Holyoak, "Categorization", chapter 5 in *Cognition*, New York: Random House, 1986, 149-180.

Greenspan, S. and J. Mylopoulos, "A Knowledge Representation Approach to Software Engineering: The Taxis Project", in (Nixon, 1984), 1-12.

Goldberg, A., *Smalltalk-80: The Interactive Programming Environment*, Reading, MA: Addison-Wesley, 1984.

Goldberg, A. and D. Robson, *Smalltalk-80: The Language*, Reading, MA: Addison-Wesley, 1989.

Gustafson, M., T. Carlsson, and J. Bubenko, "A Declarative Approach to Conceptual Information Modelling", in (Olle et al., 1982), 93-142.

Hammer, R. and D. McLeod, "Database Description with SDM: A Semantic Database Model", *ACM Transactions on Database Systems*, 6(3), September 1981, 351-386.

Haugeland, J. (ed.), *Mind Design*, Cambridge, MA: MIT Press, 1981.

Henderson-Sellers, B. and J. Edwards, "The Object-oriented Systems Life Cycle", *Communications of the ACM*, 33(9), September 1990, 142-159.

Hornick, M. and S. Zdonik, "A Shared, Segmented Memory System for an Object-oriented Database", *ACM Transactions on Office Information Systems*, 5(1), January 1987, 70-95.

Hull, R. and R. King, "Semantic Database Modelling: Survey, Applications, and Research Issues", *ACM Computing Surveys*, 19(3), September 1987, 201-260.

Jackson, M., *Systems Development*, London: Prentice-Hall International, 1983.

Keil, F., *Conceptual and Semantic Development*, Cambridge, MA: Harvard University Press, 1979.

Keil, F., "The Acquisition of Natural Kinds and Artifact Terms", in W. Demopoulos and A. Marras (eds.), *Language, Learning, and Concept Acquisition*, Norwood, NJ: Ablex, 1986, 133-153.

Kent, W., *Data and Reality*, New York: Elsevier, 1978.

Kent, W., "Limitations of Record-based Information Models", *ACM Transactions on Database Systems*, 4(1), March 1979, 107-131.

Kerckhoffs, E., G. VanSkeenkiste and B. Zeigler (eds.), *AI Applied to Simulation*, San Diego: SCS Simulation Series, 18(1), 1986.

Kerschberg, L. (ed.), *Expert Database Systems: Proceedings from the First International Conference*, Menlo Park, CA: Benjamin/Cummings, 1986.

Khoshafian, S. and G. Copeland, "Object Identity", in (Meyrowitz, 1986), 406-416.

Kim, W. and F. Lochovsky (eds.), *Object-oriented Concepts, Databases and Applications*, Reading, MA: Addison-Wesley, 1989.

King, R., "My Cat is Object-oriented", in (Lochovsky and Kim, 1989), 23-31.

King, R. and D. McLeod, "A Unified Model and Methodology for Conceptual Database Design", in (Brodie et al., 1984), 313-327.

Kung, C., "An Analysis of Three Conceptual Models with Time Perspective", in (Olle et al., 1983), 141-167.

Kung, C. and A. Solvberg, "Activity Modelling and Behavior Modelling", in (Olle et al., 1986), 145-171.

Lakoff, G., *Women, Fire and Dangerous Things: What Categories Reveal About the Mind*, Chicago: University of Chicago Press, 1987.

Lieberman, H., "Using Prototypical Objects to Implement Shared Behavior in Object-oriented Systems", in (Meyrowitz, 1986), 214-223.

Liskov, B and J. Guttag, *Abstraction and Specification in Program Development*, New York: McGraw-Hill, 1986.

London, K., *The People Side of Systems*, New York: McGraw-Hill, 1976.

Lyngbaek, P. and W. Kent, "A Data Modelling Methodology for the Design and Implementation of Information Systems", in K. Dittrich and U. Dayal (eds.), *Proceedings of the 1986 International Workshop on Object-oriented Database Systems*, Pacific Grove, CA: IEEE Press, 6-17.

Mattessich, R., "Social and Physical Reality in Accounting: Onion Model vs. Pygmalion Syndrome", Working Paper, Faculty of Commerce and Business Administration, The University of British Columbia, 1989.

McCloskey, M. and S. Glucksberg, "Natural Categories: Well-defined or Fuzzy Sets?", *Memory and Cognition*, 6(1978), 462-472.

McDermott, D., "A Temporal Logic for Reasoning About Processes and Plans", *Cognitive Science*, 6, 1982, 101-155.

McNeile, A., "Jackson System Development", in (Olle et al., 1986), 225-246.

Medin, D. and E. Smith, "Concepts and Concept Formation", *Annual Review of Psychology*, 35, 1984, 113-138.

Meyer, B., "Eiffel: An Introduction", TR-EI-3/6I, Interactive Software Engineering, Inc., July 1989.

Meyer, D., "On the Representation and Retrieval of Stored Semantic Information", *Cognitive Psychology*, 1, 1970, 242-300.

Meyrowitz, N. (ed.), *OOPSLA'86 Conference Proceedings, Sigplan Notices*, 21(9), September 1986.

Meyrowitz, N. (ed.), *OOPSLA'87 Conference Proceedings, Sigplan Notices*, 22(12), December 1987.

Meyrowitz, N. (ed.), *OOPSLA'88 Conference Proceedings, Sigplan Notices*, 23(11), November 1988.

Miller, G. and P. Johnson-Laird, *Language and Perception*, Cambridge, MA: Harvard University Press, 1976.

Minsky, M. "A Framework for Representing Knowledge", in P. Winston (ed.), *The Psychology of Computer Vision*, New York: McGraw-Hill, 1975, 211-275.

Mylopoulos, J., "Object-orientation and Knowledge Representation", in D. Meersman and W. Kent (eds.), *Object-Oriented Databases: Analysis, Design, and Construction*, Amsterdam: North-Holland, 1990.

Mylopoulos, J., "Conceptual Modelling and Telos", to appear in P. Loucopoulos and R. Zicari (eds.), *Conceptual Modelling, Databases, and CASE: An Integrated View of Information Systems Development*, New York: McGraw-Hill, 1991.

Mylopoulos, J., P. Bernstein and H. Wong, "A Language Facility for Designing Data Intensive Applications", *ACM Transactions on Database Systems*, 5(12), June 1980, 185-207.

Mylopoulos, J., A. Borgida, M. Jarke, and M. Koubarakis, "Telos: A Language for Representing Knowledge about Information Systems", *ACM Transactions on Information Systems*, 8(4), October 1990, 325-362.

Newell, A. and H. Simon, *Human Problem Solving*, Englewood Cliffs: Prentice-Hall, 1972.

Newell, A. and H. Simon, "Computer Science as Empirical Enquiry: Symbols and Search", *Communications of the ACM*, 19, March 1976, 113-126.

Nierstrasz, O., "What is the 'Object' in Object-oriented Programming?", in D. Tsichritzis (ed.), *Objects and Things*, Centre Universitaire d'Informatique, University of Geneva, March 1987, 1-13.

Nierstrasz, O., "A Survey of Object-oriented Concepts", in (Kim and Lochovsky, 1989), 3-21.

Nijssen, G., "A Gross Architecture for the Next Generation Database Management Systems", in G. Nijssen (ed.), *Modelling in Database Management Systems*, North-Holland, 1976, 1-24.

Nixon, B. (ed.), *Taxis'84: Selected Papers*, Technical Report CSRG-160, University of Toronto, June 1984.

Nygaard, K., "Basic Concepts in Object-oriented Programming", *SIGPLAN Notices*, 21(10), October 1986, 128-132.

Olive, A., "Analysis of Conceptual and Logical Models in Information Systems", in (Olle et al., 1983), 63-85.

Olle, T., H. Sol and A. Verrijn-Stuart (eds.), *Information Systems Design Methodologies: A Comparative Review*, Amsterdam: North-Holland, 1982.

Olle, T., H. Sol and C. Tully (eds.), *Information Systems Design Methodologies: A Feature Analysis*, Amsterdam: North-Holland, 1983.

Olle, T., H. Sol and A. Verrijn-Stuart (eds.), *Information Systems Design Methodologies: Improving the Practice*, Amsterdam: North-Holland, 1986.

Osherson, D. and E. Smith, "On the Adequacy of Prototype Theory as a Theory of Concepts", *Cognition*, 9, 1981, 35-58.

Pascoe, G., "Elements of Object-oriented Programming", *Byte*, August 1986, 139-159.

Peckham, J. and F. Maryanski, "Semantic Data Models", *ACM Computing Surveys*, 20(3), September 1988, 153-189.

Pernici, B., "Objects with Roles", in F. Lochovsky and R. Allen (eds.), *Conference on Information Systems*, Cambridge, MA, April 1990, 205-215.

Posner, M. and S. Keele, "On the Genesis of Abstract Ideas", *Journal of Experimental Psychology*, 77(3), 1968, 353-363.

Pressman, R., *Software Engineering: A Practitioner's Approach*, New York: McGraw-Hill, 1987.

Quillian, R., "Semantic Memory", in M. Minsky (ed.), *Semantic Information Processing*, Cambridge, MA: MIT Press, 1968.

Reed, S., "Pattern Recognition and Categorization", *Cognitive Psychology*, 3, 1972, 382-407.

Reiter, R., "Towards a Logical Reconstruction of Relational Database Theory", in (Brodie et al., 1984), 191-233.

Rentsch, T., "Object-oriented Programming", *Sigplan Notices*, 17(9), September 1982, 51-57.

Richardson, J. and P. Schwarz, "Aspects: Extending Objects to Support Multiple, Independent Roles", in J. Clifford and R. King (eds.), *Proceedings of the 1991 International Conference on Management of Data, SIGMOD RECORD*, 20(2), June 1991, 298-307.

Richter, G., "Clocks and their use for Time Modelling", in A. Sernadas, J. Bubenko, and A. Olive (eds.), *Information Systems: Theoretical and Formal Aspects*, Amsterdam: North-Holland, 1985, 49-66.

Robson, D., "Object-oriented Software Systems", *Byte*, 6(8), August 1981, 74-86.

Robson, D. and A. Goldberg, "The Smalltalk-80 System", *Byte*, 6(8), August 1981, 36-48.

Rosch, E., "On the Internal Structure of Perceptual and Semantic Categories", in T. Moore (ed.), *Cognitive Development and the Acquisition of Language*, New York: Academic Press, 1973, 111-144.

Rosch, E., "Principles of Categorization", in E. Rosch and B. Lloyd (eds.), *Cognition and Categorization*, Hillsdale, NJ: Erlbaum, 1978, 27-48.

Rosch, E. and C. Mervis, "Family Resemblances: Studies in the Internal Structure of Categories", *Cognitive Psychology*, 7, 1975, 573-605.

Roth, E. and E. Shoben, "The Effect of Context on the Structure of Categories", *Cognitive Psychology*, 15, 1983, 346-378.

Schank, R. and R. Abelson, *Scripts, Plans, Goals and Understanding*, Hillsdale, NJ: Erlbaum, 1971.

Schiel, U., "The Time Dimension in Information Systems", in (Sernadas et al., 1985), 67-76.

Schmid, H., "An Analysis of Some Constructs for Conceptual Models", in G. Nijssen (ed.), *Architecture and Models in Data Base Management*, Amsterdam: North-Holland, 1977, 119-148.

Schriver, B. and P. Wegner (eds.), *Research Directions in Object-oriented Programming*, Cambridge, MA: MIT Press, 1987.

Sciore, E., "Object Specialization", *ACM Transactions on Information Systems*, 7(2), April 1989, 103-122.

Shipman, D., "The Functional Data Model and the Data Language DAPLEX", *ACM Transactions on Database Systems*, 6(1), March 1981, 140-173

Shoham, Y., *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*, Cambridge, MA: MIT Press, 1988.

Sibley, E., "The Evolution of Approaches to Information Systems Design Methodology", in (Olle et al., 1986), 1-17.

Sigplan Notices, Special Issue on the Object-oriented Programming Workshop at IBM Yorktown Heights, 21(10), October 1986.

Simon, H., *Models of Thought*, New Haven, CT: Yale University Press, 1979.

Smith, E., "Theories of Semantic Memory", in W.K. Estes (ed.), *Handbook of Learning and Cognitive Processes*, Vol. 6, Hillsdale, NJ: Erlbaum, 1978, 1-56.

Smith, E., "Concepts and Thoughts", in R. Sternberg and E. Smith (eds.), *The Psychology of Human Thought*, Cambridge, MA: Cambridge University Press, 1988.

Smith, E. and D. Medin, *Categories and Concepts*, Cambridge, MA: Harvard University Press, 1981.

Smith, E., E. Shoben and L. Rips, "Structure and Process in Semantic Memory: A Featural Model for Semantic Decisions", *Psychological Review*, 81, 1974, 214-241.

Smith, J. and D. Smith, "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, 2(1), June 1977, 105-133.

Sol, H., "A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations", in (Olle et al., 1983), 1-7.

Stefik, M. and D. Bobrow, "Object-oriented Programming: Themes and Variations", *AI Magazine*, 6(4), Winter 1986, 40-62.

Stoyan, H., "What is an 'Object-oriented' Programming Language?", in J. Fry, R. Panko and R. Sprague (eds.), *Proceedings of the Seventeenth Annual Hawaii International Conference on Systems Sciences, Volume I*, 1984, 152-162.

Stroustrup, B., "What is 'Object-oriented Programming'?", in J. Bezevin, J.-M. Hullot, P. Cointe and H. Lieberman (eds.), *ECOOP'87 Proceedings, Lecture Notes in Computer Science*, 276, New York: Springer-Verlag, 1987, 51-70.

Su, S., "SAM*: A Semantic Association Model for Corporate and Scientific Statistical Databases", *Information Science*, 29, 1983, 151-199.

Takagaki, K., *A Formalism for Object-based Information Systems Development*, Unpublished Ph.D. Dissertation, Faculty of Commerce and Business Administration, The University of British Columbia, Vancouver, Canada, March 1990.

Takagaki, K. and Y. Wand, "An Object-oriented Information Systems Model Based on Ontology", in F. Van Assche, B. Moulin, and C. Rolland (eds.), *Proceedings of the IFIP 8.1 Working Conference on the Object Oriented Approach in Information Systems*, Quebec City, Canada, 1991, 275-296.

Teorey, T., Y. Yang and J. Fry, "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", *ACM Computing Surveys*, 18(2), June 1986.

Tsichritzis, D. and F. Lochovsky, *Data Models*, Englewood Cliffs: Prentice_hall, 1982.

Tversky, A. and D. Kahneman, "Judgment Under Uncertainty: Heuristics and Biases", *Science*, 185, 1974, 1124-1131.

Verheijen, G. and J. Van Bekkum, "NIAM: An Information Analysis Method", in (Olle et al., 1982), 536-589.

Wand, Y., "A Proposal for a Formal Model of Objects", Chapter 21 in (Kim and Lochovsky, 1989), 537-559.

Wand, Y. and R. Weber, "An Ontological Analysis of Some Fundamental Information Systems Concepts", *Proceedings of the Ninth International Conference on Information Systems*, Minneapolis, December 1988, 213-225.

Wand, Y. and R. Weber, "An Ontological Evaluation of Systems Analysis and Design Methods", in E. Falkenberg and P. Lindgren (eds.), *Information Systems Concepts: An In-depth Analysis*, Amsterdam: North-Holland, 1989, 79-107.

Wasserman, A., P. Freedman and M. Morcella, "Characteristics of Software Development Methodologies", in (Olle et al., 1983), 37-62.

Wegner, P., "Dimensions of Object-based Language Design", in (Meyrowitz, 1987), 168-182.

Winograd, T. and F. Flores, *Understanding Computers and Cognition*, Reading, MA: Addison-Wesley, 1987.

Wirfs-Brock, R. and R. Johnson, "Surveying Current Research in Object-oriented Design", *Communications of the ACM*, 33(9), September 1990, 104-124.

Wittgenstein, L., *Philosophical Investigations*, translated by G.E.M. Anscombe, New York: MacMillan, 1958.

APPENDIX 1

MODELLING AN EXAMPLE WITH MIMIC

A.1 INTRODUCTION

This appendix presents a small example which demonstrates the constructs of MIMIC. Elements of the problem are introduced gradually to illustrate different components of the model in succession. The example is taken from the banking industry and considers the activities of a branch of a commercial bank in the provision of some services to individual (as opposed to corporate) customers. The description is not intended to be complete, or necessarily representative of any specific bank. Instead, the aim is to show how knowledge about important entities, such as customers and accounts, is represented using MIMIC, and to highlight some of the important activities in maintaining knowledge about these entities. In addition to considering *transaction processing* applications, support for *simulation* and *reasoning* activities is briefly illustrated in order to demonstrate the usefulness of the model across several kinds of applications.

In order to construct the example and illustrate the features of MIMIC, it is necessary to first define a number of operations to create instances of the basic constructs of the model (e.g., objects, attributes, classes). These will be used to create and maintain a representation of the problem domain. For ease of reference, the next section restates the constructs of MIMIC. Section A.3 then describes the operations needed to create a specific representation. Section A.4 demonstrates how the operations are used to develop a representation of the banking example. Finally, Section A.5 describes how the model uniformly supports representations across several kinds of applications.

A.2 REVIEW OF MIMIC CONSTRUCTS

MIMIC consists of a number of constructs which correspond to elements of the classical theory of concepts. The fundamental constructs in MIMIC are *objects* and *attributes*. An ***object*** is a surrogate or symbol designating the existence of an instance of a concept. Attributes are divided into three kinds. A (simple)[142] ***structural attribute*** represents a (simple) structural property, and consists of a set of functions mapping from a common set of objects to a common set of values. A (simple) ***n-ary relational attribute*** represents a (simple) n-ary relational property, and consists of a set of functions mapping from a common set of objects to the power set of the Cartesian product of *n-1* other sets of objects. A ***behavioral attribute*** represents a behavioral property and consists of a function whose domain is (the cross-product of) one (or more) structural/relational attributes and codomain is the power set of the domain. Behavioral attributes specify allowed ***events*** (changes in the ***active functions*** of the structural/relational attributes involved). Events cause the state of a representation to change to reflect changes in the state of knowledge about the domain. MIMIC further recognizes that objects may acquire and lose attributes through the notions of ***domain expansion*** and ***domain contraction***, in which objects are added to and removed from, respectively, the domain of a structural or relational attribute.

The model also provides for the organization of knowledge through classes. A ***potential class*** represents a potential concept and consists of a set of structural/relational attributes with a non-empty common domain. A ***class structure*** is a collection of potential classes for which every attribute of interest

[142]The model also defines *composite* attributes.

belongs to at least one class, and in which no class is simply the union of the attributes of other classes in the structure.

Given these modelling mechanisms, the objective of this appendix is to show how they can be used to develop a representation of knowledge about a specific problem. In order to do this, however, it is necessary to first define a number of operations which enable objects, attributes, classes, and so on, to be constructed and which allow the state of a description to change in accordance with changes in the state of knowledge about the application.

A.3 OPERATIONS TO CONSTRUCT A REPRESENTATION

The MIMIC model consists of a number of constructs for describing the state of knowledge about an application, and for constraining how states may change. However, in order to represent a specific instance (e.g., the banking example mentioned in the introduction), operations are needed to create objects, attributes, and classes - and to allow states to be modified. In what follows, the operations are described by first giving a simple syntax, then listing the conditions that must hold in order for the operation to occur (pre-conditions), and finally describing the consequences of the operation (effects).

Since the objective of this exposition is simply to illustrate the use of MIMIC, the operations are defined in a somewhat informal manner. Consequently, although they provide a basic foundation for a future implementation of the model, they are not described with the object of providing a detailed blueprint for implementation. Furthermore, the focus here is only on mechanisms for constructing a representation and allowing its state to change. An implementation would obviously have to also provide additional mechanisms for querying a representation to extract information from it.

A.3.1 Creating Objects[143]

The object is the basic element in a representation created using MIMIC. Objects are simply surrogates which designate the existence of instances of concepts. In constructing an example, objects must be created to represent the instances of concepts that are of interest. The operation to do this is called **MAKEOBJECT**. Let $\underline{O}$ designate the set of objects in the representation (i.e, that have already been created: initially, $\underline{O} = \{\}$), and **o** designate an arbitrary symbol.

Syntax

MAKEOBJECT(o)

Pre-condition

$o \notin \underline{O}$ (the object has not already been created)

[143]The model also uses the notion of *values*. Since values are based on notions such as characters and numbers, they can be assumed to be supported in any environment. Hence, it adds nothing to the discussion here to consider operations for creating values.

Effect

$\underline{O} \leftarrow \underline{O} \cup \{o\}$ (the object is added to the existing set of objects)

This operation can be used to create both simple and composite objects, since there is no restriction on the nature of the surrogates. Specifically, a surrogate in $\underline{O}$ could be a concatenation of other elements of $\underline{O}$, in which case, it would be a composite object. However, to keep the discussion more manageable, the notions of composite objects, attributes, and classes will not be considered in detail.

Note that a more complex operation could be considered here, in which objects are classified as they are created (e.g., MAKEOBJECT(o,C), where C designates the name of a class, but not necessarily the only one, to which the object is assigned). This presupposes that all objects will be classified when created and that classes are defined before objects are created. However, rather than explicitly treat classification as part of the MAKEOBJECT operation, it will be considered through the assignment of attributes to objects and the creation of classes.

A.3.2 Creating Attributes

The creation of objects simply reflects knowledge of the existence of things. However, it is equally important to be able to describe properties of things. Therefore, operations are needed to create attributes. These are divided into operations for creating *structural*, *relational*, and *behavioral* attributes.

The operation to create a structural attribute is called **MAKESTRUC**. The purpose of the operation is to create a table designating (the active function of) a structural attribute[144]. Let $\underline{S}^U$ and $\underline{S}^C$ designate the sets of (the active functions of) all unchangeable and changeable (respectively) structural attributes that have been created (initially $\underline{S}^U=\{\}$, $\underline{S}^C=\{\}$), and $\underline{O}$ designate the set of all objects created. Let $s = \{<o_i,v_i>_{i=1,\dots,I}\}$ denote a set of I object-value pairs[145], and **type** denote a variable which can assume one of two values: $\boldsymbol{u}$ (for unchangeable) and $\boldsymbol{c}$ (for changeable)[146].

Syntax

MAKESTRUC(s,type)

---

[144]Since a structural attribute is defined in MIMIC as a *set* of functions, more is needed before the meaning of a *changeable* structural attribute (whose size is greater than one) is fully operationalized. This is handled later in the discussion of behavior. In other words, the set of functions defining a structural attribute is not enumerated when the attribute is created using the MAKESTRUC operation, unless the attribute is *unchangeable* (e.g., Birthdate).

[145]The active function of an attribute is referred to using a lower case letter (s), while the upper case equivalent (S) will be used to refer to the attribute more generally as a set of functions.

[146]The distinction between changeable and unchangeable attributes is made explicit here since an operation to construct an attribute creates only the active function. Unless the attributes are subdivided into those which may and may not change, there is no immediate way of determining whether an attributes values may vary over time. The distinction was not needed in developing the model in Chapter 4 since, at that level, the "changeability" of an attribute is implicit in the number of functions in the set.

Preconditions

$o_i \in \underline{O}$, $i=1,\ldots,I$ (each first element of the pairs in s is an existing object)

$o_i \neq o_j$, $\forall$ $i,j \in I$, $i \neq j$ (no object appears in two pairs, so that s is a function)

Effect

If type=u, $\underline{S}^U \leftarrow \underline{S}^U \cup \{s\}$ (function is added to the existing set of unchangeable structural attributes)

If type=c, $\underline{S}^C \leftarrow \underline{S}^C \cup \{s\}$ (function is added to the existing set of active functions of changeable attributes)

In this operation, the (active) function is expressed as a set of pairs, where *the o_i's jointly constitute the domain of the attribute.* The **MAKESTRUC** operation provides for the creation of simple or composite structural attributes.

The operation to create a relational attribute is called **MAKEREL**. The purpose of the operation is to create a table designating the active function of a relational attribute (see Footnote 144). Let $\underline{\mathbf{R}}^U$ and $\underline{\mathbf{R}}^C$ designate the sets of (the active functions of) all unchangeable and changeable (respectively) relational attributes that have been created (initially $\underline{R}^U=\{\}$, $\underline{R}^C=\{\}$), and $\underline{\mathbf{O}}$ designate the set of all objects created. Let **r** = $\{<o^1_i,o^2_i>_{i=1,\ldots,I}\}$ a set of I object-object pairs[147], and **type** denote a variable which can assume one of two values: ***u*** (for unchangeable) and ***c*** (for changeable).

Syntax

MAKEREL(r,type)

Precondition

$o^1_i,o^2_i \in \underline{O}$, $i=1,\ldots,I$ (the objects involved have already been created)

$o^1_i \neq o^1_j$, $\forall$ $i,j \in I$, $i \neq j$ (no object appears as the first element in two pairs, so that **r** is a function)

Effect

If type=u, $\underline{R}^U \leftarrow \underline{R}^U \cup \{r\}$ (function is added to the existing set of unchangeable relational attributes)

If type=c, $\underline{R}^C \leftarrow \underline{R}^C \cup \{r\}$ (function is added to the existing set of active functions of changeable attributes)

As already noted, the operations do not allow the meaning of structural and relational attributes to be fully captured, since attributes are formally defined in MIMIC as sets of functions. However, these sets can be characterized implicitly by providing an operation for defining behavioral attributes. This operation is called **MAKEBEH**. Behavioral attributes constrain events (changes in active function), and will be expressed here using sentences in first-order predicate logic. Let **F** denote a formula of the form:

---

[147]This applies to binary attributes. For n-ary relational attributes, the second element of each pair is a set of *n-1* objects.

F: $<p_1,p_2>$:= *a sentence in first order logic,*

where $\mathbf{p_1}$ and $\mathbf{p_2}$ denote functions in an attribute **P**. This interpreted to mean that a direct transition in the active function of **P** from $\mathbf{p_1}$ to $\mathbf{p_2}$ is permitted only if the sentence to the right of ":=" is true when evaluated[148]. Let $\underline{F}$ denote the set of formulas that have been created (initially $\underline{F} = \{\}$).

As an illustration, suppose that an attribute **Age** is defined as the set of functions, **Age** = $\{a_1,\ldots,a_K\}$. The behavioral constraint may be expressed as:

F: $<a_i,a_j>$:= $\forall\ o \in O^1,\ a_j(o)= a_i(o)+1$ (where $O^1 \subseteq \underline{O}$).

This means that an event $<a_i,a_j>$ ($i,j \in \{1,\ldots,K\}$, $i \neq j$) can occur only if **F** is true when evaluated using functions $\mathbf{a_i}$ and $\mathbf{a_j}$ (i.e., ages change in increments of one unit). In **F**, $\mathbf{a_i}$ and $\mathbf{a_j}$ are variables for which the specific functions involved in an event (e.g., a_2, a_8) are substituted in determining if the event is permissible (events are discussed in more detail in Section A.3.4).

Syntax

MAKEBEH(F)

Precondition

Each term in F refers to one or more structural/relational attributes (so that the truth of the sentence is determined by the active functions of these structural and relational attributes)[149]

Effect

$\underline{F} \leftarrow \underline{F} \cup \{F\}$

The **MAKEBEH** operation establishes **F** as a logical constraint (or pre-condition) on events. That is, no event may occur if it violates **F**. Note that the operation to create F's does not decide whether the sentence is a "good" description of knowledge about behavior. That is the job of the analyst.

A behavioral attribute is formally defined in MIMIC as a function. The operationalization above can be given such a functional interpretation. Specifically, if $\mathbf{p_i}$ and $\mathbf{p_j}$ denote functions in a structural/relational attribute $P=\{p_1,p_2,\ldots,p_K\}$, $p_j \in b(p_i)$ (or $<p_i,p_j>$ is possible) if and only if **F** is true when evaluated with respect to a specific $p_i,p_j \in P$ (where **b** denotes the behavioral attribute function). In other words, **F** implicitly describes (but does not enumerate) the function **b** that constitutes a behavioral attribute.

[148]This does not mean that an indirect transition from $\mathbf{p_1}$ to $\mathbf{p_2}$ is prohibited (e.g., $p_1 \rightarrow p_3 \rightarrow p_5 \rightarrow p_2$).

[149]This is left somewhat vague. The meaning becomes clearer when specific examples are considered in Section A.4.

A.3.3 Creating Classes

Thus far, operations have been defined to create objects and attributes. The MIMIC model also provides for the organization of knowledge through classes. Consequently, an operation is needed to construct classes by aggregating attributes. This operation is called **MAKECLASS**. Its purpose is to establish a collection of attributes as a class. Moreover, the operation explicitly places a new class in an existing class lattice (which is expressed as a graph).

In the following, $\underline{S}=\underline{S}^U \cup \underline{S}^C$ and $\underline{R}=\underline{R}^U \cup \underline{R}^C$. Let $\underline{C}$ designate the existing set of classes (initially $\underline{C}=\{\}$), $p^1,...,p^N$ denote the active functions of structural/relational attributes $\mathbf{P^1,...,P^N}$, and $\mathbf{G} = \{<C^1_i, C^2_i>_{i=1,...,I}\}$ denote a class lattice (initially G={}). **G** is interpreted to mean that $\mathbf{C^1_I}$ is a subclass of C^2_i (i=1,...,I). Let **dom(p^n)** denote the domain of attribute $\mathbf{p^n}$ (i.e., the set of first elements of the pairs in p^n)[150].

Syntax

MAKECLASS($C^*,p^1,...,p^N$) ($\mathbf{C^*}$ is a symbol designating the new class name)

Preconditions

$p^1,...,p^N$ exist: that is, each $\mathbf{p^n}$ belongs to $\underline{S}$ or $\underline{R}$ (the sets containing the active functions of existing structural and relational attributes, respectively)

$dom(p^1) \cap ... \cap dom(p^N) \neq \emptyset$ (the class has a non-empty extension)

$\nexists\, p^*, p^* \in \underline{S} \cup \underline{R}$ and $p^* \notin \{p^1,...,p^N\}$, such that $dom(p^*) \supseteq dom(p^1) \cap ... \cap dom(p^N)$ (the class contains all attributes possessed by all members)

$\nexists$ a subset of classes $\{C^1,...,C^K\}$ of $\underline{C}$ such that $\cup_{k=1,...,K} C^k = \{p^1,...,p^N\}$ (the class is not simply the aggregation of attributes of some subset of existing classes)

Effects

$C^* \leftarrow \{p^1,...,p^N\}$

$\underline{C} \leftarrow \underline{C} \cup \{C^*\}$

For each $C \in \underline{C}$ such that $C \subset C^*$, $<C^*,C>$ is added to **G**

For each $C \in \underline{C}$ such that $C^* \subset C$, $<C,C^*>$ is added to **G**

This operation both creates classes and adds them explicitly to the class lattice. Since the latter is implied when is a class is defined anyway, it need not be listed as an effect, but is included here to

[150]Some liberty is taken here in reference to attribute names and symbols designating the active functions of the attributes (P's versus p's). Hence, **dom(p^n)** could equally well be written **dom(P^n)**. The MAKECLASS operation applies to the attributes, which initially are "known" only in terms of the active functions.

improve clarity.

Each class created using the **MAKECLASS** operation is a potential class in the sense used in Chapter 4. In addition, the constraint that the collection of classes constitute a class structure can be imposed by requiring that the operation be repeated until all attributes that are defined are assigned to at least one class. This exposition assumes that all attributes are defined before any classes are. In practice, there may be some iteration between defining objects, attributes, and classes. In that case, an operation may be needed to add newly defined attributes to a class. Since that issue depends on the approach taken in analyzing a problem, it is not pursued here.

In practice, it will also be necessary to define operations to "destroy" objects or remove them from the representation. Since the interest here is only in showing how a representation may be created, such operations are not discussed.

A.3.4 Supporting Change

A representation, once created, is useful only if it changes to reflect changes in the state of knowledge about the domain. Two kinds of change are relevant here. First, objects may acquire and lose attributes. Consequently, objects will have to be added to the domain of attributes. The operation to achieve this is called **DOMEXP**. Likewise, objects will have to be removed from the domain of attributes. The operation to achieve this is called **DOMCON**. The second kind of change involves changes of state that occur when objects take on new values for structural or relational attributes. These changes will be effected through **MAKEVENT** operations.

Domain expansion is operationalized as follows. Let **p** denote the active function of a structural or relational attribute **P**, $\underline{\mathbf{P}}$ denote the set of (the active functions of) all existing structural/relational attributes, and **A** denote a non-empty set of *I* pairs $\{<o_i,x_i>_{i=1,...,I}\}$, where the $\mathbf{x_i}$'s are either values (if **p** belongs to a structural attribute) or sets of objects (if **p** belongs to a relational attribute). The DOMEXP operation adds *I* pairs to the active function of an attribute.

Syntax

DOMEXP(p,A)

Pre-conditions

$p \in \underline{P}$

$o_i \in \underline{O}$, i=1,...,I

The $\mathbf{o_i}$'s do not belong to any pairs in **p**

Effect

$p \leftarrow p \cup A$

Using this framework, an object could join a class by a series of **DOMEXP** operations, one for each attribute defining the class which was not previously possessed by the object (e.g., a PERSON may become an EMPLOYEE by acquiring the added attributes which define EMPLOYEE).

The **DOMCON** operation has the opposite effect of **DOMEXP**. It removes one or more pairs from the active function of an attribute, reflecting that the objects involved lose the attribute in question. Let **p** denote the active function of a structural/relational attribute **P** (i.e., **p** is a set of object-value pairs), **dom(p)** denote the domain of **p**, and **A** a non-empty set of *I* objects $\{o_1,...,o_I\}$.

Syntax

DOMCON(p,A)

Pre-condition

$o_i \in dom(p)$, i=1,...,I

Effect

$p = p-\{<o_i,x_i>_{i=1,...,I}\}$ (the $\mathbf{x_i}$'s are values or sets of objects associated with the objects being removed)

Next, the notion of an event is operationalized to provide a mechanism for changing the state of a representation. The operation **MAKEVENT** is defined as a state change generator. Unlike operations to create objects, attributes, and classes, **MAKEVENT** is not used in creating an initial representation, but is used to maintain the state of the representation as changes occur in the domain.

In MIMIC, an event changes the active function of one *or more* structural/relational attributes. For simplicity, the operation is characterized here in terms of one attribute only. More complex changes can be constructed using a series of these simpler **MAKEVENT** operations. Let **p** designate the active function of an existing structural/relational attribute **P**, **dom(p)** denote the domain of **p**, $\underline{\mathbf{P}}^{\mathbf{C}}$ denote the set of all changeable structural/relational attributes, and **E** a set of *I* pairs $\{<o_i,x_i>_{i=1,...,I}\}$, where the $\mathbf{x_i}$'s are either values (if **p** belongs to a structural attribute) or sets of objects (if **p** belongs to a relational attribute). Let **p′** designate the function formed by replacing each pair in **p** which has the same first element as a pair in **E** (i.e., an o_i) with the corresponding pair in **E** (i.e., $<o_i,x_i>$).

Syntax

MAKEVENT(p,E)

Preconditions

$o_i \in dom(p)$, i=1,...,I

$p \in \underline{P}^C$ (the attribute is changeable)

For each $F \in \underline{F}$ (i.e., each behavioral attribute), **F** is true when **p** and **p′** are "substituted" into **F**

Effect

p′ replaces **p** as the active function of **P**

To illustrate this, suppose $p = \{<o_1,v_1>,<o_2,v_2>\}$ and $E = \{<o_2,v_3>\}$. The operation MAKEVENT(P,E) produces:

$p' = \{<o_1,v_1>,<o_2,v_3>\}$

which becomes the new active function of **P**. The event can then be designated <p,p'>.

Using this notion of event, it is now possible to see how the logical sentence **F**, which expresses a behavioral attribute, implicitly describes the set of pairs (i.e., the function) that is a behavioral attribute, and also determines the set of functions that constitutes a structural or relational attribute. In effect, **F** determines a set of pairs $\{<p,\{p_j\}_{j\in J_i}>_{i\in I}\}$ (where $j=1,\ldots,|J_i|$ for each **i**). The set of pairs is a function (where **p** and the $\mathbf{p_j}$'s denote active functions before and after a **MAKEVENT** operation, respectively). This means that an event <p,p'> can occur only if there is a pair <p,{...,p',...}> in the functional equivalent of **F**.

The set of functions constituting the attribute **P**, then, is the set of all functions which appear (again, this is implicit) in **F** (in the above case, $p_1,\ldots,p_{|I|}$).

This concludes the description of the basic operations needed to create and maintain a representation. Note that the operations are intended to be used at the object level first, while the "schema" (consisting of attributes and classes) is created later (since attributes presuppose objects and classes presuppose attributes). The next section shows how these are used to create a representation of knowledge about things in a banking environment, which provides the basis for several kinds of information systems. In building the example, objects are first created, then attributes and classes.

A.4 CONSTRUCTING THE EXAMPLE

A.4.1 Overview

The subject matter of the representation constructed here is a bank. Attention is focused on the objects of interest, such as customers, employees, accounts, and loans. In addition, the effect of events on the state of objects is demonstrated by looking at changes that result from withdrawals from accounts. The exposition here is partial in two senses. First, only enough detail is given to illustrate how the operations defined in Section A.3 are used to create a representation of domain knowledge. Second, only a few instances are considered when, in reality, a bank would have many more.

A.4.2 Objects

Objects are created to represent the existence of instances of concepts. The major concepts of interest here include persons, customers, employees, various kinds of accounts, and loans. These terms are used very loosely at this point so that the reader gains an informal idea of the "kinds" of objects that are being created. While users might, in practice, identify classes of objects before indicating specific instances, the presentation here will begin by creating objects and attributes since classes can only be defined when these are determined. Definitions of classes (in terms of the attributes they possess) are given in Section A.4.4, and give formality to terms such as *customer* and *savings account.*

On examining this hypothetical subject matter, a number of instances are identified. These include the persons *John, Mary, Jane, Bill, Sally, Frank.* Objects to represent instances are created through a series of **MAKEOBJECT** operations. Initially, $\underline{O}=\{\}$ (the set of objects is empty). From this starting point

MAKEOBJECT(*John*) results in $\underline{O}$ = {*John*}[151],
MAKEOBJECT(*Mary*) results in $\underline{O}$ = {*John, Mary*},

and so on. After repeating the operation four more times

$\underline{O}$ = {*John, Mary, Jane, Bill, Sally, Frank*}.

Other objects to be created represent instances of accounts and loans. The objects are created through repeated **MAKEOBJECT** operations. Eventually, the set of objects is as follows:

$\underline{O}$ = {*John,Mary,Jane,Bill,Frank,S100,S101,S102,S103,S104,C200,C201,L301,L302,L303*}

Note that no mention has been made of the attributes possessed by the objects, and only very intuitive reference to "classes" such as person. In practice, the analyst would probably acquire information about classes, attributes, and objects in an iterative and interactive manner.

A.4.3 Attributes

The objects of interest each possess a number of attributes. Once the objects are created, it is possible to create the attributes they possess. For example, a **Name** attribute is constructed as follows. Let **Name** denote the set

{<*John*,'John Doe'>, <*Mary*,'Mary Williams'>, <*Jane*,'Jane Doe'>, <*Bill*, 'Bill Wilson'>, <*Sally*, 'Sally Smith'>, <*Frank*, 'Frank Stokes'>}.

Name is created as a changeable structural attribute using the **MAKESTRUC** operation. Initially, $\underline{S}^C$={} (the set of active functions of changeable structural attributes is initially empty). Performing the operation

MAKESTRUC(Name,c) results in $\underline{S}^C$ = {Name}.

The operation is repeated to create a number of other structural attributes: **Address, Birthdate, Experience, Balance, Interest_rate_paid, Term, Principal,** and so on. Each time the operation invoked, a new function is added to either the set $\underline{S}^C$ or $\underline{S}^U$. For example, **Birthdate** is added to $\underline{S}^U$. Initial active functions of several of the attributes listed above are shown next using a tabular representation[152]. Performing the **MAKESTRUC** operation on each of these functions produces

$\underline{S}^C$ = {Name, Address, Experience, Balance, Interest_rate_paid, ...},
$\underline{S}^U$ = {Birthdate, Principal, Term, ...}.

[151]Throughout this example, objects will be designated by italicized words.

[152]These active functions will, in the case of changeable attributes, be replaced by new ones as events occur (see Section A.4.6).

Address

John	123 Main Street
Mary	456 Walnut Crescent
Jane	123 Main Street
Bill	456 Walnut Crescent
Sally	789 Crabtree Road
Frank	159 Maple Drive

Experience

John	7
Mary	25
Sally	2

Balance

S100	500
S101	100
S102	250
S103	1000
S104	10000
C200	10
C201	150

Birthdate

John	19 09 1950
Mary	03 12 1963
Jane	15 05 1948
Bill	16 09 1960
Sally	21 04 1965
Frank	27 06 1970

Note that these attributes do not all share the same domain. This is relevant to the definition of classes that is given later.

Objects also possess relational attributes linking them to other objects. For example, a **Spouse** attribute is constructed using:

MAKEREL(Spouse,c), where

Spouse = {*<John, Jane>*, *<Mary, Bill>*, *<Jane, John>*, *<Bill, Mary>*, *<Sally, Nil>*, *<Frank, Nil>*}.

Since initially $\underline{R}^c$={ } and $\underline{R}^u$={ }, MAKEREL(Spouse) results in $\underline{R}^c$ = {Spouse}. The **MAKEREL** operation can be repeated to create the active functions of other relational attributes, including **Holds**, **Handles**, and **Held_by**. These are listed below in tabular form.

Holds

John	{S100, C200 L301}
Mary	{S101, S102}
Bill	{S103, C201}
Sally	{L302}
Frank	{S104, L303}

Handles

John	{S101, S102, S103, S104, C201}
Mary	{L301, L302, L303}
Sally	{S100, S101, S102, S103, S104, C200, C201}

Held_by

S100	John
S101	Mary
S102	Mary
S103	Bill
S104	Frank
C200	John
C201	Bill
L301	John
L302	Sally
L303	Frank

Repeating the **MAKESTRUC** operation for these functions results in

$\underline{R}^C$ = {Spouse, Holds, Handles, ...}, and
$\underline{R}^U$ = {Held_by,...}.

There are also a number of behavioral constraints, associated with the structural/relational attributes listed above, which restrict allowed changes in active functions. Some restrictions represent logically necessary constraints (e.g., changes to years of experience must follow a pattern of increments by one). Others may be viewed as corresponding to "business rules" of the organization. That is, they reflect conventions by which the bank conducts its activity (e.g., maximum withdrawals without prior notice on savings accounts).

Behavioral attributes are created using the **MAKEBEH** operation. Let $\underline{F}$ denote the set of restrictions that have been created. Initially, $\underline{F}=\{\}$. Behavioral constraints are created by invoking **MAKEBEH.**

In order to identify behavioral attributes, it is useful to look at structural/relational attributes, and the restrictions on changes to their active functions. For example, the attributes **Name** and **Address** have no general restrictions. That is, it is assumed that any string (of a "reasonable" length) may serve as the value of **Name(o)** or **Address(o)** for an object $o \in O$, and that changes of **Name** and **Address** are unrestricted. On the other hand, **Birthdate** contains a single function. Hence, there are no behavioral attributes, as the notion of a change in the active function is undefined.

There is, however, an assumed restriction on changes to the active function of **Spouse.** A person who has a spouse will, after an event involving a change in the active function of **Spouse,** either have the same spouse or have no spouse. That is, a married person cannot have another spouse without first being in a state of having no spouse.

This restriction is expressed more precisely as a constraint on changes to the active function of **Spouse.** Letting:

s_1 designate the active function of **Spouse,**
s_2 designate another function of **Spouse** (s_1 and s_2 are variable here), and
dom(Spouse) designate the domain of **Spouse,**

the restriction is:

$$F_1: \langle s_1,s_2 \rangle := \forall\ x,y \in \text{dom(Spouse)}\ (x \neq y),\ \ s_1(x) = y \Rightarrow s_2(x) = y \vee s_2(x) = \text{Nil}.$$

The behavioral constraint is created by invoking the operation

MAKEBEH(F_1).

Since initially $\underline{F}=\{\}$ (the set of behavioral constraints is empty), MAKEBEH(F_1) results in $\underline{F} = \{F_1\}$.

If an event $\langle s_1,s_2 \rangle$ (a change in the active function of **Spouse** brought on by invoking a MAKEVENT operation as described in Section A.4.6) satisfies this disjunctive condition, it is a permitted event.

Similarly, for the structural attribute **Experience,** there is the obvious restriction that changes in years of experience for an employee occur in increments of 1. The constraints will not be listed for all attributes here. However, as a further illustration, a constraint on changes to the structural attribute **Balance** is detailed. Assume that a customer may withdraw no more than the greater of $1,000 (or the account balance, if less than \$1000) or 50% of the account balance *of a savings account* in a single transaction. Although such a constraint may appear somewhat artificial, it does resemble commonly used restrictions on withdrawals made without advanced notice. Letting:

$\mathbf{b_1}$ denote the active function of **Balance,**
$\mathbf{b_2}$ denote another function of **Balance,** and
dom(Balance) designate the domain of **Balance,**

the constraint is:

F_2: $<b_1,b_2> := \forall\ x \in \text{dom(Balance)},\ b_2(x) \geq \min\{\max\{b_1(x)\text{-}1000,0\},0.5*b_1(x)\}$.

The constraint is imposed by performing MAKEBEH(F_2), which results in $\underline{F} = \{F_1,F_2\}$. The operation is repeated for each of the behavioral constraints identified in the analysis, resulting in a set of restrictions $\underline{F} = \{F_1, F_2,...\}$.

A.4.4 Classes

Once all structural, relational, and behavioral attributes of interest have been created, groupings of attributes can be formed to define classes. To this point in the example, only vague reference has been made to the fact that objects are of various kinds (customers, accounts, etc.). The MAKECLASS operation introduced in Section A.3 allows a class to be formally defined as a set of attributes.

The first class defined here is PERSON. Let $\underline{C}$ denote the set of classes and **G** the class lattice. Initially, $\underline{C}=\{\}$ and G={}. PERSON is determined to consist of the attributes **Name, Address, Birthdate,** and **Spouse.** The operation to define the class is:

MAKECLASS(PERSON, Name, Address, Birthdate, Spouse).

The results are that PERSON = {Name, Address, Birthdate, Spouse} and $\underline{C}$ = {PERSON}. Nothing is added to **G** at this point (the lattice is simply a single node). Next, the class CUSTOMER is created by invoking

MAKECLASS(CUSTOMER, Name, Address, Birthdate, Spouse, Holds),

resulting in

CUSTOMER = {Name, Address, Birthdate, Spouse, Holds}
$\underline{C}$ = {PERSON, CUSTOMER}
G = {<CUSTOMER,PERSON>}.

Next, the class EMPLOYEE is defined by invoking

MAKECLASS(EMPLOYEE, Name, Address, Birthdate, Spouse, Experience, Handles}

resulting in

EMPLOYEE = {Name, Address, Birthdate, Spouse, Experience, Handles}
$\underline{C}$ = {PERSON, CUSTOMER, EMPLOYEE}
G = {<CUSTOMER,PERSON>, <EMPLOYEE, PERSON>}.

Note that if the operation MAKECLASS(CUSTEMP, Name, Address, Birthdate, Spouse, Holds, Experience, Handles) is attempted in order to define a class CUSTEMP as a subclass of both CUSTOMER and EMPLOYEE, it violates the precondition (of **MAKECLASS**) that a class cannot simply be the union of the attributes of existing classes. Therefore, the operation would not be performed unless additional attributes were specified as parameters.

The operation can be repeated to create the remaining classes of interest, including PRODUCT, LOAN, ACCOUNT, SAVING, CHEQUING, and so on. For a quick summary, the operations to create these classes are shown below, and the final contents of $\underline{C}$ and **G** are indicated. Some attributes which are listed were not described among the attributes mentioned earlier in explaining attribute creation.

MAKECLASS(PRODUCT,Held_by)
MAKECLASS(ACCOUNT,Held_by,Balance)
MAKECLASS(CHEQUING,Held_by,Balance,Overdraft,Issued_cheques)
MAKECLASS(SAVING,Held_by,Balance,Interest_rate_paid)
MAKECLASS(LOAN,Held_by,Interest_rate _charged,Term,Principal,Remaining_balance)

When these operations are performed, the result is:

$\underline{C}$ = {PERSON,CUSTOMER,EMPLOYEE,PRODUCT,ACCOUNT,CHEQUING,SAVING,LOAN}
$\underline{G}$ = {<CUSTOMER,PERSON>, <EMPLOYEE,PERSON>, <ACCOUNT,PRODUCT>, <LOAN,PRODUCT>, <CHEQUING,ACCOUNT>,<SAVING,ACCOUNT>}[153].

A.4.5 Domain Changes

The representation thus far is static. To be useful, it must change to reflect changing knowledge. One way that knowledge changes is through the recognition and classification of new instances. The **MAKEOBJECT** operation allows new objects to be created, while the **DOMEXP** operation allows new objects to be added to the domain of existing attributes. For example, suppose that an object *Dave* is created using **MAKEOBJECT**, and that the object is required to possess the attributes **Name, Address, Birthdate, Spouse, Holds**. If the object *Dave* is to be assigned the value 'Dave Parsons' for the attribute **Name**, the attribute is updated by invoking

DOMEXP(Name,{<*Dave*, 'Dave Parsons'>}.

As a result of this operation, the **Name** attribute is:

[153]The lattice here is actually two disjoint lattices, since no persons are products.

Name

John	John Doe
Mary	Mary Williams
Jane	Jane Doe
Bill	Bill Williams
Sally	Sally Smith
Frank	Frank Stokes
Dave	**Dave Parsons**

The **DOMEXP** operation can be repeated for each of the other attributes possessed by *Dave*, adding a row to the active functions of each. When *Dave* is added to the domain of these attributes, the object is implicitly classified as a CUSTOMER (and a PERSON) since it possesses the attributes defining both classes.

Similarly, objects may be removed from the representation to reflect that instances no longer exist from the point of view of the application. For example, if *Sally* pays off the loan she holds, she may no longer be considered a CUSTOMER. The operation

DOMCON(Holds, {*Sally*})

removes the pair <*Sally*, {*L302*}> from the active function of **Holds**, resulting in

Holds

John	{S100, C200 L301}
Mary	{S101, S102}
Bill	{S103, C201}
Frank	{S104, L303}

Note that this means that *Sally* is no longer a CUSTOMER, but remains in the representation as a PERSON. If, for example, the person Sally dies, further **DOMCON** operations would be required to remove the object *Sally* from the domain of the attributes **Name, Address, Birthdate, Spouse.**

A.4.6 Events

A banking environment produces a variety of changes that must be tracked. For example, there are changes to CUSTOMER attributes such as **Address** and **Holds**. A significant group of changes in values are those to account balances brought on by deposits to and withdrawals from accounts. For example, suppose that the active function of the attribute **Balance** (of ACCOUNT) is:

b_1

S100	500
S101	100
S102	250
S103	1000
S104	10000
C200	10
C201	150

Any changes to the active function must satisfy the behavioral constraint given earlier:

F_2: $<b_1,b_2> := \forall\ x \in \text{dom(Balance)},\ b_2(x) \geq \min\{\max\{b_1(x)-1000,0\},0.5*b_1(x)\}$.

Changes are brought about by invoking the **MAKEVENT** operation. Two examples are considered here. In the first, the event is permitted as it does not violate $\mathbf{F_2}$. In the second, the event is denied as it violates $\mathbf{F_2}$.

Example 1

A withdrawal of $350 is attempted from account *S100* (which has a current balance of $500). Using the notation introduced in Section A.3, E = {<*S100*,150>}. The operation

MAKEVENT(b_1,E)

is invoked in order to create the event. However, in order for the event to be created, it must satisfy

F_2: $<b_1,b_2> := \forall\ x \in \text{SAVING},\ b_2(x) \geq \min\{\max\{b_1(x)-1000,0\},0.5*b_1(x)\}$,

where $\mathbf{b_1}$ is the table listed above and $\mathbf{b_2}$ is the function produced by replacing the row <*S100*, 500> in $\mathbf{b_1}$ with <*S100*, 150>. Since this condition is satisfied if the **MAKEVENT** operation succeeds, the operation takes place and $\mathbf{b_2}$ becomes the active function of **Balance**.

Example 2

A withdrawal of $6000 is requested from account *S104*, so that E = {<*S104*,4000>}. The operation MAKEVENT(b_1,E) is invoked in order to create the event. However, since $\mathbf{b_2}$ is the function produced by replacing the row <*S104*,10000> in $\mathbf{b_1}$ with <*S104*, 4000>, the change would violate $\mathbf{F_2}$. Consequently, a precondition of the **MAKEVENT** operation is not met and the operation does not succeed.

In Chapter 5 (Section 5.7), it was mentioned that records of events can be considered as objects. This means that when a **MAKEVENT** operation takes place, an additional effect can be specified. This is to create a new object. Subsequently, a number of **DOMEXP** operations can be triggered to add the object to the domains of the attributes it possesses. These attributes will, in turn, have been created using **MAKESTRUC** and **MAKEREL** operations. Similarly, sets of these attributes can be specified using **MAKECLASS** operations to create event record classes. For example, the operations

```
MAKESTRUC(Time,u)
MAKESTRUC(Amount,u)
MAKEREL(From_account,u)
MAKEREL(By_teller,u)
```

can be invoked to create attributes which allow representation of the time and amount of a withdrawal, the account from which the withdrawal is made, and the teller who processes the withdrawal. The second parameter takes the value **u** since attributes of event records are unchanging. Initially, each of these attributes may contain no pairs. Pairs are added as withdrawals occur (i.e., as **MAKEVENT** operations occur). The class WITHDRAWAL may be formed by invoking the operation

```
MAKECLASS(WITHDRAWAL,Time,Amount,From_account,By_teller).
```

In the same manner, other classes of event records can be formed for kinds of other events (e.g., deposits, transfers, loan payments).

A.5 UNIFORMITY OF THE MODEL

This section demonstrates that a representation created using MIMIC can be used for at least three kinds of applications: *transaction processing*, *reasoning*, and *simulation*. The differences among the three are argued to be in the nature of the changes that occur. In transaction processing, the objects, attributes, and instances in a representation reflect instances representing things that exist, and domain changes and events occur in accordance with knowledge about changes in the real world. This would be the case when events such as deposits and withdrawals reflect actual actions of customers, as assumed in the descriptions of the previous section. In reasoning (or expert system) applications, a common objective is to determine whether a sequence of events that leads to a specified state is possible given the behavioral constraints contained in the representation. In simulation, the interest is in examining the effect of hypothetical events on certain parameters of a real or hypothetical system. In other words, the three kinds of applications differ in terms of the nature of events (real/possible/hypothetical). In addition, they differ in terms of the manner in which operations to create events occur (Wand and Weber, 1988). Transaction events are triggered by input from a user interface. Reasoning events are generated by a software-based inference engine which tries to determine whether certain combinations of events are possible. Simulation events are generated by a software-based mechanism which invokes changes according to some random or pre-specified temporal pattern.

The objective of transaction processing is to accurately track the state of knowledge about the domain of interest by recording actual changes in the state of things in the domain. Events such as the withdrawals discussed in the previous section support such tracking activity. Hence, use of MIMIC to support transaction processing is not considered further. The remainder of this section is devoted to showing how the constructs of the MIMIC model can support reasoning and simulation activities in the banking example introduced earlier.

A potential reasoning application for the example used here might involve a customer's application for a loan. Consider expanding the original description of the domain as follows. A loan application may be considered an object, and is created using the operation

MAKEOBJECT(*L_a_1*),

where ***L_a_1*** is a surrogate (identifier) designating a specific loan application. The **MAKESTRUC** and **MAKEREL** operations are used to create the attributes **Amount, Status,** and **Applicant** (which may initially be empty sets). The operation

MAKECLASS(LOAN_APPLICANT, Amount, Status, Applicant)

creates the class LOAN_APPLICANT. The attribute **Status** is assumed to have the codomain {accepted, rejected, undetermined}, reflecting three possible status values of an application. In order to classify the object ***L_a_1*** as a loan application, the **DOMEXP** operation is invoked three times to add the object to the active function of each of the three attributes defining the class. This example assumes that, whenever a new application (*L_a_n*) is created, the pair <*L_a_n*,undetermined> is added to the active function of **Status** using a **DOMEXP** operation. The assumption plays an important role in triggering the reasoning process, as it can trigger an examination of loan eligibility criteria expressed as behavioral constraints.

In a similar manner, a class LOAN_APPLICANT can be defined as a subclass of CUSTOMER, with additional attributes that include **Capability, Commitment, Risk,** and **Application.** In this example, the first three of these attributes are each assumed to have the codomain {high, moderate, low, undetermined}. When a loan application is made, a **DOMEXP** operation is performed on each of these three attributes to add the customer (i.e., the value of Applicant(*L_a_n*)) to the class LOAN_APPLICANT.

The role of the reasoning mechanism (the details of which are not of concern here) might be to regard the value 'undetermined' as unstable, and to identify allowed events which would change the active functions of the attributes in question to produce more stable values (e.g., accept, reject) for the newly created instances. That is, when a **DOMEXP** operation is invoked which adds an object with an 'undetermined' value to the active function of an attribute, the reasoning mechanism is also triggered, and checks the constraints on changes to the active function in question. To illustrate this, let:

n denote an object of class LOAN_APPLICATION,
a(n) denote the object of class LOAN_APPLICANT that is the value of **Applicant(n),**
s_1 denote the active function of **Status** (s_1(n)=undetermined),
s_2 denote an active function of **Status** such that s_2(n)=accept,
c denote the active function of **Capability** (c(x)=undetermined),
m denote the active function of **Commitment** (m(x)=undetermined),
r denote the active function of **Risk** (r(x)=undetermined).

Suppose that a constraint on events $<s_1,s_2>$ is as follows:

$$F_3\colon <s_1,s_2> := c(a(n))=\text{high}\vee((m(a(n))=\text{moderate}\vee m(a(n))=\text{low})\wedge r(a(n))=\text{low}).$$

Informally, this means that the status of an application may change from 'undetermined' to 'accept' if and only if the capability of the applicant is 'high' <u>or</u> if the commitment is 'moderate' or 'low' and the risk is 'low'. In order to determine if F_3 is satisfied (and the event $<s_1,s_2>$ possible), it is necessary to determine the values of **c(a(n)), m(a(n)),** and **r(a(n)).** Consequently, if the specification of constraints or "rules" is complete, there will also be F's which describe conditions under which each of the attributes **Capability, Commitment,** and **Risk** change active functions so that values for new applicant objects change from 'undetermined' to either 'high', 'moderate', or 'low'. For example, changes to **Capability** may depend on the existing attributes **Income** (of CUSTOMER) and **Balance** (of ACCOUNT). The specific conditions can be incorporated into behavioral constraints. For example, for the active function to change so that the value of **Capability** for an individual changes from 'undetermined' to 'high', it may be required that the salary of the application to be at least \$50,000 and the account balance at least \$10,000. This might be expressed as:

$$F_4: <c_1, c_2> := \text{Income}(a(n)) \geq 50000 \vee \Sigma_{t \in \text{Holds}(a(n))} \text{Balance}(t) \geq 10000,$$

where, for **a(n)**, $c_1(a(n))$=undetermined and $c_2(a(n))$=high.

Other rules could be similarly expressed using the **MAKEBEH** operation. For example, values assigned to **Commitment(x)** may depend on values such as **Number_dependents(x)** and **Housing_expenditures(x)**. The point here is not to describe an inference mechanism, but simply to illustrate that the MIMIC model supports the representation of knowledge for reasoning applications and the operations described earlier allow this knowledge to be represented for a specific example.

The third type of information systems activity supported by the model is *simulation.* A banking environment provides many areas in which simulations may be used to support decision making. This example considers the effect of the number of tellers who serve customers on statistics such as teller utilization and customer waiting time.

Some knowledge that is important in such a simulation might be of interest only for the simulation application and, therefore, not otherwise represented. This knowledge can nevertheless be expressed by defining new objects, attributes, and classes. Furthermore, a simulation can make use of knowledge about the current (actual) state of entities in the representation (e.g., accounts held by a customer).

A critical part of any discrete event simulation is the generation of events. The simulation events that are of interest for this particular problem are *potential* events (withdrawals, deposits, transfers). Records of these are distinguished from those of actual transactions by their membership in a newly defined class, S_TRANSACTION. In addition, changes in active functions (i.e., events) during a simulation run will affect only the attributes of *classes defined for the simulation* (e.g., QUEUE). Hence, the historical state of the representation of organization entities (e.g. CUSTOMERs) **will not** change during a simulation run. Note that this is not a consequence of the MIMIC model, but must be enforced by mechanisms in the software that drives the simulation.

To illustrate the use of the model in a simulation context, the example and its representation are described as follows. The primary classes of entities of interest for the simulation are the existing classes CUSTOMER, SAVING, and CHEQUING, along with three new entity classes: TELLER (a subclass of EMPLOYEE), QUEUE and SERVING. In addition, there are two classes of records of simulation events. The instances of TRANSACTION represent instances of service to a customer by a teller, while the instances of WAIT represent instances of customers waiting in queues. These classes are defined next.

First, class QUEUE is considered. QUEUE is defined to consist of a single attribute, **Contains** (indicating the customers in each queue), which is defined with the operation

MAKEREL(Contains,c).

Initially, Contains={} (each queue is empty). The class is defined using

MAKECLASS(QUEUE,Contains).

This class has a behavioral attribute constraining changes to the active function of **Contains**. Letting:

c_1 denote the active function of **Contains**,
c_2 denote another function of **Contains**,
x, y denote instances of QUEUE,

the restriction is:

$$F_5: \langle c_1,c_2\rangle := (c_1(x)-1\leq c_2(x)\leq c_1(x)+1)\wedge(c_2(y)=c_1(y)\ \forall y\in \text{QUEUE},\ y\neq x).$$

This constraint is established by invoking MAKEBEH(F_5), and requires that the contents of exactly one queue changes by exactly one element in any event on **Contains**.

The class SERVING has a number of instances equal to the number of tellers in the simulation. An instance of this class can really be viewed as a "queue" of 0 or 1 elements (depending on whether the teller is available or busy, respectively). Two attributes are defined:

MAKEREL(Teller,u),
MAKEREL(Is_serving,c).

Initially, Teller = {} and Is_serving = {}. The class definition is:

MAKECLASS(SERVING,Teller,Is_serving).

The instances of this class are in one-to-one correspondence with the instances of TELLER used in the simulation. For example, if the single teller case (with teller **t**) is being simulated, there is one instance of SERVING, **g**, for which Teller(g)=t.

Next, the transaction classes are defined. A single class, TRANSACTION, is defined to represent simulation transactions. Objects in this class are records of (i.e., represent knowledge about) individual simulation transactions. The primary attributes of interest are the customer and teller involved in the transaction, along with the time at which the transaction begins and the time at which it ends. These last two attributes, respectively, represent times at which (1) there is a change in the active function of **Contains** of QUEUE (reducing queue size) and at which (2) there is a change in the active function of **Is_serving** of SERVING. Consequently, there are two events relating to each instance of this class. First, the attributes are defined:

MAKEREL(For_customer,u)
MAKEREL(By_teller,u)
MAKEREL(Start_time,u)
MAKEREL(Stop_time,u).

Initially, all four attributes contain the empty set. The class is defined using:

MAKECLASS(TRANSACTION,For_customer,By_teller,Start_time,Stop_time).

The remaining transaction class of interest involves the waiting of customers in queues. Each instance of this class represents the wait of a customer in a queue. The attributes of interest involve the specific customer, the queue, the time of entry, and the time of departure, and are defined as above (each initially contains the empty set). The class definition is:

MAKECLASS(WAIT,Customer,Queue,Time_in,Time_out).

The values of the last two attributes for an instance of WAIT are, respectively, times at which there is a change in the active function of **Contains** (increasing queue size), and a change in the active function of **Is_serving**. Furthermore, for each instance of WAIT, there is an instance of TRANSACTION whose value for **Start_time** is the same as the value of **Time_out** of that instance of WAIT. Neither of these two event classes has behavioral attributes, as the instances are unchanging.

It is also necessary to have a way of determining the time required to serve a customer. This can be achieved by defining a class TELLER as a subclass of EMPLOYEE, with the added structural attributes **Mean_service_time**, which represents the mean time for a teller to perform an activity, and **Sd_service_time**, which represents the standard deviation (and possibly added relational and behavioral attributes). The example assumes that, for each teller, the actual time required to serve a customer is normally distributed around the mean. Each time a customer arrives at a teller for service, a value for service time can be obtained by sampling from the normal distribution using the parameters which describe the teller in question.

This extension of the example allows a simulation to be supported and statistics to be captured[154]. Statistical data are not considered as attributes of individual or composite entities (e.g., waiting times of entities in queues). Instead, statistics can be calculated at the completion of a simulation run by examining the attributes of events. The description above permits two important statistics to be supported. First, the waiting time of a customer in the queue is simply the difference between the time the customer leaves the queue and the time he/she enters the queue. For each entry in a queue, these two data are the values of the attributes **Time_out** and **Time_in** for an instance of IN_QUEUE. The average waiting time is simply the sum of these waiting times divided by the number of instances of IN_QUEUE (the latter is equal to one half of the number of changes in the active function of **Contains** of QUEUE). Second, the utilization of a teller, **t**, is simply the sum of the differences between the **Stop_time** and **Start_time** attributes for each instance, **s**, of S_TRANSACTION for which By_teller(s)=t, divided by the total duration of the simulation period.

The extensions of the example provided here allow the progress of customers through a simulation to be recorded. In order to run a simulation, arrivals of customers need to be generated. Just as the user interface was not considered in constructing a representation for transaction processing activities, the software to generate simulation events is beyond the scope of interest here. However, in general such arrivals would be generated by a piece of software which may simulate arrivals by sampling from some statistical distribution, and then triggering events in the system by invoking **DOMEXP**, **DOMCON**, and **MAKEVENT** operations as required by the values sampled.

To conclude, this appendix shows the way in which different kinds of applications may be supported within a single representation (or system) using the MIMIC model, highlighting the similarities and differences between such applications. In all three cases considered (transaction processing, reasoning, simulation), knowledge about the structure, relationships, and potential behavior of things must be represented. The same model constructs (object, attribute, class) can uniformly represent that knowledge. The differences between the applications are highlighted by what is outside the scope of MIMIC: namely, the mechanisms for generating events. In transaction processing applications, actual events are reported through an interface with the real world system that is modelled by the information system. In reasoning applications, the effects of possible events (as constrained by behavioral attributes) are explored by a mechanism which seeks a certain goal state. In simulation applications, hypothetical events are generated according to some pre-specified criteria. In each of these three cases, the mechanisms for "generating" events are embedded in code, but this code is not described using the constructs of MIMIC.

[154]Of course, statistics could also be kept for transaction processing activities.

APPENDIX 2

GUIDE TO NOTATION

This appendix summarizes the important notation of Chapter 4 in the order in which it is introduced. The terms are divided by level: that is, cognitive and object.

Cognitive Level

$\mathbf{T}$	a set of instances
$\mathbf{V}$	a set of values
$\mathbf{S}$	a structural property: $\{s_i \mid s_i{:}T \rightarrow V\}$
s_i	an element of the structural property $\mathbf{S}$
$\mathbf{S}_T$	a set of structural properties shared by the instances $\mathbf{T}$
$\wp(T)$	the power set of the set of instances $\mathbf{T}$
$\wp(\mathbf{T}^1 \otimes \ldots \otimes \mathbf{T}^K)$	the power set of the Cartesian product of $\mathbf{T}^1,\ldots,\mathbf{T}^K$
$\mathbf{Q}$	a subset of $\wp(\mathbf{T}^1 \otimes \ldots \otimes \mathbf{T}^K)$
$\mathbf{q}$	an element of $\mathbf{Q}$
$\mathbf{R}$	a relational property: $\{r_j \mid r_j{:}T \rightarrow Q\}$
$\mathbf{r}_j$	an element of the relational property $\mathbf{R}$
$\mathbf{R}_T$	a set of relational properties shared by the instances $\mathbf{T}$
$\mathbf{P}$	a structural <u>or</u> relational property
$\mathbf{p}^k_{ik}$	the $\mathbf{i}_k$ function of the property $\mathbf{P}^k$
$\sigma(T)$	the state of the set of instances $\mathbf{T}$: $<p^1_{i1},\ldots,p^K_{iK}>$
$\sigma(t)$	the state of an instance $t \in T$: $<p^1_{i1}(t),\ldots,p^K_{iK}(t)>$
$\Sigma(t)$	the potential state space of each $t \in T$
$\Sigma(T)$	the possible state space of a set of instances $\mathbf{T}$
$\mathbf{e}$	an event: $<\sigma(T),\sigma'(T)>$
$\mathbf{b}$	a behavioral property: $\mathbf{b}{:}P^1 \otimes \ldots \otimes P^K \rightarrow \wp(P^1 \otimes \ldots \otimes P^K)$
$\mathbf{C}$	a concept (potential or realized), denoted: $P = \{P^1,\ldots,P^K\}$ $= <S,R,B>$ $= <P,B>$
CS	a concept structure: $\{C^1,\ldots,C^K\}$

<u>**Object Level**</u>

$\mathbf{O}$	a set of objects
$\mathbf{U}$	a set of value surrogates
$\mathbf{S}^*$	a structural attribute: $\{s^*_i \mid s^*_i: O \rightarrow U\}$
$\mathbf{s}^*_i$	an element of the structural attribute $\mathbf{S}^*$
$\mathbf{S}^*_O$	a set of structural attributes shared by the objects $\mathbf{O}$
$\wp(O)$	the power set of the set of objects $\mathbf{O}$
$\wp(O^1 \otimes ... \otimes O^K)$	the power set of the Cartesian product of $\mathbf{O}^1,...,\mathbf{O}^K$
$\mathbf{Q}^*$	a subset of $\wp(O^1 \otimes ... \otimes O^K)$
$\mathbf{q}^*$	an element of $\mathbf{Q}^*$
$\mathbf{R}^*$	a relational attribute: $\{r^*_j \mid r^*_j: O \rightarrow Q^*\}$
$\mathbf{r}^*_j$	an element of the relational attribute $\mathbf{R}^*$
$\mathbf{R}^*_T$	a set of relational attributes shared by the objects $\mathbf{O}$
$\mathbf{P}^*$	a structural <u>or</u> relational attribute
$\mathbf{p}^{*k}_{ik}$	the $\mathbf{i}_k$ function of the attribute $\mathbf{P}^{*k}$
$\sigma(O)$	the state of the set of objects $\mathbf{O}$: $<p^{*1}_{i1},...,p^{*1}_{iK}>$
$\sigma(o)$	the state of an object $o \in O$: $<p^{*1}_{i1}(t),...,p^{*K}_{iK}(o)>$
$\Sigma(o)$	the potential state space of each $o \in O$
$\Sigma(O)$	the possible state space of a set of objects $\mathbf{O}$
$\mathbf{e}$	an event: $<\sigma(O),\sigma'(O)>$
$\mathbf{b}^*$	a behavioral attribute: $\mathbf{b}^*: P^{*1} \otimes ... \otimes P^{*K} \rightarrow \wp(P^{*1} \otimes ... \otimes P^{*K})$
$\mathbf{C}^*$	a class (potential or realized), denoted: $P^* = \{P^{*1},...,P^{*K}\}$ $= <S^*,R^*,B^*>$ $= <P^*,B^*>$
$\mathbf{CS}^*$	a class structure: $\{C^{*1},...,C^{*K}\}$