CHAPTER 1
INTRODUCTION

1.1 Data Modelling

A database is a collection of stored data together with their description and inter-relationships. It represents the semantics of an application sometimes called miniworld [Elmasri 1989] as completely and accurately as possible. Also changes to the miniworld should be reflected in the database. A Data Base Management System (DBMS) provides users with a conceptual representation of data that does not include details of how the data is stored. A data model is used to provide the conceptual representation and hence it hides storage details from the database users. Thus, it provides a framework of concepts used to express the semantics of the miniworld. It comprises of (a) defining types of data to be stored; (b) manipulation of a database which involves operations to update (insert, delete and modify) the database in order to reflect changes in the miniworld and querying the database; (c) implicit consistency constraints as well as mechanisms for defining explicit constraints that further reflect the miniworld semantics.

In classical data models (hierarchical, network or relational) [Tsichritzis 1982], [Date 1981] there is a considerable gap between the semantics of an application and its representation within the database. This is
because the semantics of an application may be modelled as a set of entities and relationships among them at various levels of abstraction. But, in classical systems the abstraction is not possible. This is primarily due to the fact that only atomic data can be stored in these systems. This is where object-oriented database systems come in, as they support the concept of data abstraction. Also, these systems are based on data models that allow one to represent a real-world entity, however complex its structure may be, by a single object in the database which may be composed of other objects. This is accomplished by binding the domain (called data type in programming languages such as Pascal and C) of an attribute of a class (or relation) to another class (or relation). Thus, object-oriented systems support modelling of complex entities and relationships directly.

The classical data models are tailored to account for the representation of simpler entities as those required in data processing applications such as inventory control, payroll, accounts, etc., which have fixed format data representation and which deal with few properties like name, age, salary, etc. In these applications, the number of instances of a type/relation is far greater than the number of types/relations in the database. But in advanced applications like Computer Graphics, CAD, Geographic Information Systems, VLSI-design, Image Processing and Office Information Systems, the entities
are complex and properties of entities can be other entities of arbitrary complexity. Also, the number of classes is substantial as compared to instances of these classes. Therefore, there is a need for representing strong relationships among classes of objects. In object-oriented systems the classes of objects are organised into a class hierarchy. Thus, traditional database technology has been found to be inadequate for these advanced applications. These are discussed in Chapter 1.

It is also desirable that some fundamental generic kinds of semantic relationships [Brodie 1984], [Hull 1987], [Peckham 1988] are expressible by object-oriented data models. These are: (i) 'has-subtype' relationship between two types, where one is a subtype of the other; (ii) 'has-instance' relationship between two objects, where one is an instance of the other; and (iii) 'has-component' relationship between two objects, where one is a component of the other.

As database systems move towards complex structured applications, emphasis shifts to describing fewer items with more complex relationships [Brodie 1984]. Since new applications involve a large number of types and few instances of each type, some techniques of knowledge representation [Brodie 1984], inheritance of procedures, 'is-a' hierarchies and 'part-of' hierarchies have migrated to object-oriented data modelling.
In the past few years research in the area of object-oriented systems have lead to the development of several prototypes. Each prototype supports different subset of characteristics required for object-oriented databases and programming. Some of the important models are ORION [Banerjee 1987], [Kim 1989], [Kim 1990a] developed at MCC; OZ+ [Weiser 1989], developed at University of Toronto; IRIS [Derrett 1986], [Fishman 1987], [Lyngbaek 1986] by Hewlett-Packard Laboratories; POSTGRES [Rowe 1987], [Stonebraker 1988], [Stonebraker 1990] developed at University of California; Probe [Manola 1986], [Dayal 1986], [Orenstein 1988] from Computer Corporation of America; Cactis [Hudson 1989] from University of Colorado; Starburst [Hass 1990], [Schwarz 1986] from IBM Almanden Research Centre; Exodus [Carey 1986] from University of Wisconsin and Genesis [Batory 1988] from University of Texas.

In Chapter 2, we review the research efforts in the area of object-oriented database modelling. The object-oriented paradigm and abstraction mechanisms are briefly discussed. Further, the conceptual schema and data operations in object-oriented data model are described. The advantages of object-oriented database systems over conventional record-based database systems are discussed. On-going efforts in creating object-oriented DBMSs are classified into three categories; (i) those that are directly based on the
object-oriented paradigm; (ii) extensions to relational systems; (iii) experimental toolkits (or storage systems). The salient features of some important sample prototype models under these categories have been reviewed. This is followed by a comparison of these prototype models.

1.2 Graphical Databases

In recent years computer graphics is emerging as one of the most interesting areas of application in database technology. It has been pointed out [Copeland 1984], [Kent 1979], [Kemper 1987] that the conventional relational database management systems do not allow the modelling of graphical objects.

Also, as more and more people want to use graphics, declarative approach [Tarlton 1989] is found to be more useful especially for the non-specialists. This is because in procedural approach, the system assumes that the user has knowledge of difficult mathematical functions. The graphics standards GKS [ISO 1985], GKS-3D [ISO 1987] and PHIGS [ISO 1986] adopt procedural approach. These are discussed in detail in Chapter 3. We adopt declarative approach.

In GKS and GKS-3D graphics contents of a segment cannot be modified. Also, a segment cannot include references to other segments, thus providing a single-level or flat data structures. PHIGS overcomes
these problems and allows structures to contain references to other structures which enables handling of hierarchical relationships among graphics data. It also provides some functions for editing the contents of a structure. It is however not possible to edit contents of individual elements.

Further, both in GKS and PHIGS, it is not possible to assign names to output primitives and to manipulate them as individual items. It is also not possible to assign attributes to single primitive or collection of primitives by its name or identifier.

Graphics database systems should have the capability of creating and manipulating (i) individual objects, their geometry and attributes, and (ii) the complex semantics of application objects [Jiang 1989], [Wisskirchen 1990], [Zhu 1988]. Object-oriented paradigm [Banerjee 1987], [Cox 1984], [Nierstras 1989], [Pascoe 1986], [Stroustrup 1988] as a basic concept provides these features. In fact Spooner [Spooner 1986] was the first to apply these concepts for modelling CAD data. For further related work for CAD applications we refer to [Batory 1985], [Mitschang 1989], [Harder 1987]. Yu [Yu 1989] has applied object-oriented paradigm to CAD database and for graphics database we refer to [Jiang 1989] and [Zhu 1988].
The graphics systems mentioned above address different needs and applications and therefore no single system meets all the requirements. They cannot be called complete systems and cannot compete with the graphics standards. Thus it is necessary to synthesize between existing object-oriented graphics systems and functionalities provided by graphics standards.

In Chapter 3, we present a new scheme [Bhalla 1991a] to integrate the declarative approach to graphics and object-oriented data modelling techniques to form a fruitful symbiosis for constraint-based graphics database systems. It has rich modelling constructs to describe graphics data, allows sharing of representation and supports part hierarchies.

1.3 Constraints

In any miniworld situation, there are always many rules, called integrity constraints that govern it. In a database an integrity constraint [Date 1981] [Elmasri 1989] (or simply a constraint) is a restriction or condition which the data in the database must satisfy to be consistent with some model of the real world from which it comes.

There are several ways in which constraints can arise in the database. They may express application dependent rules for consistency among objects and relationships.
For example, one constraint in a personnel database could be that the age of an employee is expected to lie between 16 and 65.

Going beyond simple algebraic relationships, constraints have been used to support graphical movement of objects. One of the early environments for direct manipulation of graphical objects using constraints was the work of Alan Borning on the ThingLab system [Borning 1979], [Borning 1981], [Borning 1986], [Maloney 1989]. However, Sutherland's Sketchpad system [Sutherland 1963] was the first to realize the importance of constraints on graphical data. In Nelson's JUNO graphics system [Nelson 1985], constraints are allowed to be specified by selecting icons and points.

Of the numerous constraints involved in CAD/CAM application and engineering databases, only some have been made explicit while majority of constraints are still remain embedded in the application software and in designer's head.

These constraints are derived from the meaning or semantics of the data and the miniworld it represents. It is the responsibility of the database designers to specify integrity constraints during database design. Some constraints can be specified to the DBMS and automatically enforced. Other constraints may be checked by update programs or at the time of data entry.
However, no data model is capable of representing all types of constraints that may occur in a miniworld. Hence, it is usually necessary to specify additional explicit constraints on each particular database schema that represents a particular miniworld application.

Since it is difficult for a programmer to take into account all relevant constraints and their interaction in every program, it is essential that we take those constraints out of the programming domain and move them to the automatic processing domain. This not only minimizes the amount of work involved, but it also ensures that relevant constraints will be taken into account.

In the constraint-based systems mentioned earlier constraints are used to control the layout of objects on the screen. These systems allow constraints to be imposed on the geometry of objects and on the relationship between two or more objects. But, these systems do not allow multiple output devices. Hence they do not consider constraints that are important in this framework. As these systems are not database systems they do not impose constraints when new objects are inserted in or deleted from the database, or are modified.

In Chapter 4, we have identified important classes of constraints in the context of object-oriented graphics database systems [Bhalla 1991a]. In our scheme, we allow constraints to be imposed on individual objects, between
two or more objects, on a class of objects, on the relationship between window, viewport and device, and on part hierarchies. Our scheme also provides useful mechanisms for management of integrity constraints. Examples are given for maintenance of constraints at the time of insertion, deletion and modification.

1.4 Formal model and operations

Unlike relational databases [Maier 1986] which have theoretical foundation, a universally agreed upon model to support object-oriented databases has not yet emerged. The major reason is that unlike the independent relations in relational databases, an object-oriented schema is a hierarchy of classes. A class is simultaneously a node on 'is-a' hierarchy and 'part-of' hierarchy.

In most object-oriented databases [Kim 1989], [Hudson 1989] in which a class not only defines the type of objects belonging to it but also represents a collection of objects. That is, in these systems, the concept of types and collection of objects is overloaded in the definition of a class. This approach creates confusion and complicates the model. Also, it is not possible to have classes which share the same type. Further, it is not easy to define operations on classes that create new classes.
In Chapter 5, we present a formal model [Bhalla 1991b] in which we make a clear distinction between the two concepts and define association between them. This approach allows us to define classes which share the same type and separate the two meanings of "is-a" relationships into subtype/supertype and subclass/superclass relationships.

Using this approach we attempt to extend the query model of ORION [Banerjee 1987], [Banerjee 1988]. The extension is based on adding two new features. First, is to extend relational algebra operations and set-theoretic operations to the world of classes. Second, we allow new classes to be created as a result of applying these operations on classes along the lines of relational databases. Also, since new types and classes cannot exist in isolation we have identified their position in the type and class hierarchy respectively. These operations allow queries against multiple target classes and enable us to extract attributes similar to relational project.

1.5 Operations on Complex Objects

In Chapter 6, following the approach of separating types and classes we have extended the relational algebra operations (select, project) and set-theoretic operations (union, intersection, difference, cartesian product) on component-class hierarchies [Bhalla 1991c]. We allow new component-class hierarchies to be defined by applying
these operations. The extended query model allows a query against multiple target component-class hierarchies. It also enables us to retrieve complex objects and put them as members of new classes. We can also extract subobjects from complex objects.

In Chapter 7, we briefly describe our important results as conclusions.