

**Decision Support Systems using Object-Oriented Technology**

by

Xiaobo Li

Submitted to the Department of Electrical Engineering and  
Computer Science in Partial Fulfillment of the Requirement for the  
Degree of

Master of Engineering

at the

Massachusetts Institute of Technology

February 2000

© 2000 Xiaobo Li  
All rights reserved.

The author hereby grants to MIT permission to reproduce and to  
distribute publicly paper and electronic copies of this thesis  
document in whole or in part and to grant others the right to do so.

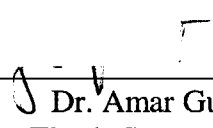
Signature of Author

---

Department of Electrical Engineering and Computer Science  
February 2, 2000

Certified by

---

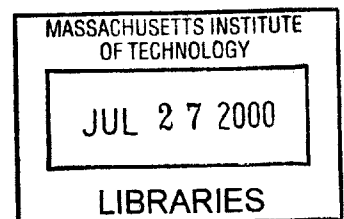
  
Dr. Amar Gupta  
Thesis Supervisor

Accepted by

---

  
Arthur C. Smith  
Chairman, Department Committee on Graduate Theses

ENG



Decision Support Systems Using Object-Oriented Technology  
By  
Xiaobo Li

Submitted to the  
Department of Electrical Engineering and Computer Science

February 2, 2000

In Partial Fulfillment of the Requirements for the Degree of  
Master of Engineering in Electrical Engineering and Computer Science

## **Abstract**

Object-oriented databases are a relatively new technology that has great potential for a variety of business applications. The primary advantage of the object-oriented database is that it preserves the natural hierarchical structure of the real world, and for certain cases allows more efficient communication between the database application software and the stored data. This thesis examines two decision support systems that currently employ relational databases and evaluates the structure of each of the application. A design change is then proposed for each of these applications such that an object-oriented database replaces the relational database and the potential benefits versus potential costs of the replacement are discussed.

Thesis Supervisor: Amar Gupta

Title: Co-Director, Productivity From Information Technology (PROFIT) Initiative

## **Acknowledgement**

I am very grateful for an inspiring year and the fruitful period spend working on this thesis at the MIT. I benefited extensively from the helpful and constructive advice of my thesis supervisor Dr. Amar Gupta. His friendliness, insight and guidance are appreciated greatly. I would also like to thank Doug Norman, the Distributed Computing Group and MITRE Corporation for providing me an exciting opportunity to carry out research and work. I am greatly indebted to Eric Mui whose valuable comments and recommendations eliminated many errors from this thesis. Finally, I would like to thank my parents for their constant support and encouragement throughout my years at MIT.

# Contents

<b>1. Background</b>	<b>7</b>
1.1 Introduction	7
1.2 Object-oriented Technology	8
1.3 Relational Database Management Systems (RDBMS)	10
1.4 Object-oriented Database Management Systems (OODBMS)	11
1.5 Decision Support Systems (DSS)	14
1.6 Outline of Thesis	18
<b>2. OODBMS Application vs. RDBMS Application</b>	<b>19</b>
2.1 Types of Stored Data	19
2.2 Data Manipulation	20
2.3 Connectivity with Existing Programming Languages	21
2.4 Database Design Techniques	22
2.5 Types of Applications	26
2.6 Summary	28
<b>3. OODB and DVI Knowledge Suite™</b>	<b>29</b>
3.1 MITRE	29
3.2 Background on DVI Knowledge Suite™	30
3.2.1 Target Users	30
3.2.2 DVI Knowledge Suite™ Structure	31
3.3 Evaluation	34
3.3.1 Criteria	34
3.3.2 Procedure	35

3.3.3 Previous Analysis	35
3.3.4 Analysis	38
3.4 Application of OODB to DVI Knowledge Suite™	42
3.4.1 Design Requirement	43
3.4.2 Data Components and Behavior	44
3.5 Effects of OODB on DVI Designer	47
3.6 Summary	48
<b>4. OODB and Contract Data Management System</b>	<b>50</b>
4.1 Components and Requirements for CDMS	50
4.2 CDMS Functional Requirements	53
4.3 Analysis of CDMS with Relational Database	54
4.4 Design of CDMS with OODB	56
4.4.1 Structure	56
4.4.2 Primitive Design	56
4.4.3 Improved Design	61
4.5 Effects of OODB on CDMS	63
4.6 Summary	64
<b>5. Conclusions</b>	<b>65</b>

## List of Figures and Tables

Figure 1.1 The Evolution of Object-oriented Databases	12
Figure 1.2 Data Orientation vs. Model Orientation in Decision Support Systems	16
Figure 1.3 Conceptual Model of DSS	17
Figure 3.1 Application Architecture of DVI Knowledge Suite™	32
Figure 3.2 Structure of DVI Designer	33
Figure 3.3 Interaction among DataView Objects	45
Figure 4.1 Normal Contract	51
Figure 4.2 Basic Ordering Agreement Contract	52
Figure 4.3 Version Control in CDMS	53
Table 4.1 Contract Lifecycle Status	54
Figure 4.4 Primitive OODB Design for CDMS	57
Table 4.2 Objects and their data members in improved OODB design for CDMS	62

# Chapter 1

## BACKGROUND

### 1.1 Introduction

Software applications written for business purposes interact with a large set of data. The data may come in many variations, such as integers, strings, or objects. New ways to store and efficiently query business data are continually being researched. Currently, the prevalent method to store data is the relational database. The relational database can store and query data of primitive types, such as integers, strings, and numerals. Relational database technology is mature and well developed, and has been deployed by the business community for a wide variety of software applications and systems.

As software engineering technology shifts towards the object-oriented paradigm, object-oriented databases have emerged as a new area of interest for both computer science researchers and software engineers. An object-oriented database stores objects, created while programming in object-oriented languages such as C++ or Java. The technology used to create the object-oriented database is young relative to that of relational databases. There are many technical issues to be resolved with object-oriented

databases and its related applications. However, this is an exciting field as object-oriented databases exhibit great potential for the creation of more powerful business applications.

There are advantages and disadvantages to both object-oriented and relational database technology. This thesis compares and contrasts the two types of database technology in its application to business software packages classified as decision support systems. Two specific decision support systems, implemented with relational database technology, are examined. A design for the modification of each of these two systems, switching the underlying database to object-oriented technology, is presented here to demonstrate the usefulness and advantages of object-oriented databases.

The remainder of this chapter describes elementary terminology, background, and fundamental concepts in relevant technology areas. These technology areas are object-oriented technology, relational database management systems, object-oriented database management systems, and decision support systems.

## **1.2 Object-Oriented Technology**

A program written in a traditional procedural computer language commands a series of procedures, which are invoked to act on a set of data. However, the evolution of the computation model and data structures has exposed serious deficiencies in structured programming languages. Keeping data and the tasks to be performed on the data separate has become increasingly difficult. Code reusability has become a serious weakness since the code that performs similar tasks must be rewritten when minor changes in the data structure are made. Object-oriented programming evolved to ameliorate these deficiencies.



Object-oriented programming languages share some defining characteristics: *encapsulation*, *inheritance*, and *polymorphism*.

*Encapsulation* provides users with a well-defined interface to a set of functions that hides their internal workings. In object-oriented languages, data that is internal to an object are not exposed, but instead are manipulated by class member functions. Data access through class member functions, rather than direct manipulation, is the preferred policy for object-oriented programmers [6].

*Inheritance* is the ability to derive new classes from existing classes. A derived class ("subclass") inherits the instance variables and methods of the base class ("superclass"), and may add new instance variables and methods. New methods may be defined in the subclass with the same names as those in the superclass, in which case they *override* the superclass. For example, we might have a class **integers**, which has the method **add()** defined. If we create a subclass called **bytes**, the new class will inherit the **add()** method from its superclass, and thus objects of the **bytes** class will have the **add()** method defined [6].

*Polymorphism* is the characteristic of being able to assign a different meaning to a particular symbol or "operator" in different contexts. In practice, this usually means the ability to overload and override existing functions. Overriding a function is explained in the previous paragraph. Overloading simply means the ability to assign the same name to multiple functions. As long as the input parameters are different, the compiler can distinguish between the different functions and the appropriate one will be invoked at run time [6].

In the use of object-oriented languages for business applications, objects are created to represent entities and functions of the business rather than the logical procedures of traditional data processing. The objects are models for real-world entities such as customers, products, invoices, employees, etc.

The effect of building business applications as object-oriented programs is the simplification of the development process. Object-oriented technology provides techniques for managing enormous complexity, achieving reuse of software components, and coupling data with tasks that manipulate that data.

### **1.3 Relational Database Management Systems (RDBMS)**

A database management system is a software package that manages large structured sets of persistent data, and offers ad-hoc query capability to many users simultaneously. The system organizes its data in such a fashion that it optimizes data querying. Data within a database are related in some manner, forming one or more *relationships*. A relationship describes the connection between data. This connection may also be referred to as references, associations, or links. Associated with a relationship are the ideas of direction and cardinality [10].

A relationship may be unidirectional or bi-directional. A unidirectional relational allows the reference from one datum to another but not in the other direction. A bi-directional relationship allows traversals in both directions.

A relationship also has cardinality. There are three common types of cardinality: one-to-one, one-to-many, or many-to-many. A one-to-one relationship allows a single datum to be related to one and only one other datum. A good example of the one-to-one

relationship is the spousal relationship. A one-to-many relationship relates one datum to many data. An example of this type of relationship is a house having many rooms. The inverse of the one-to-many relationship is the many-to-one relationship. The above example still applies, only in the opposite direction. The last type of relationship is the many-to-many relationship. A good example of this type of relationship is that between parents and children – a parent may have more than one child and each child may have more than one parent [10].

Relational databases are currently the most widely used database systems. Invented at IBM by E. F. Codd in 1970 [7], a relational database consists of a set of tables containing data fit into predefined categories. Each table, also known as a relation, organizes data into categories called columns. Each row of the table contains a unique instance of data for the categories defined by the columns. The main advantages of the relational database are ease of creation, accessibility, and extension. After the original database is created, a new data category can be added without requiring that all existing applications be modified.

#### **1.4 Object Oriented Database Management Systems (OODBMS)**

Research and practice in object-oriented databases dates back to the 1970's and had become a significant research area by the early 1980's. Initial commercial products appeared in the late 1980's [10]. Today, there are many companies marketing a variety of object-oriented databases that are considered second generation products.

At this time, object-oriented database management systems (OODBMS) are an emerging technology. The path of object-oriented database technology development is illustrated in Figure 1.1.

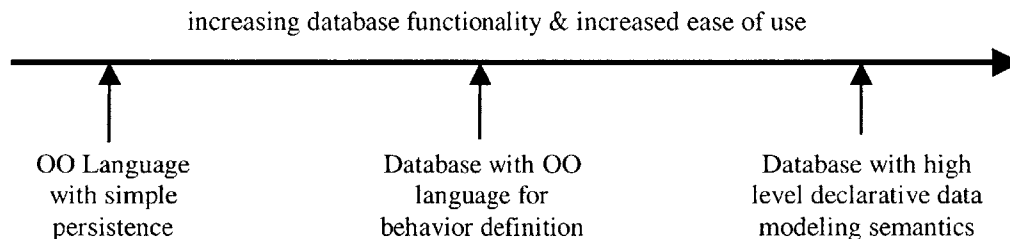


Figure 1.1: The evolution of object-oriented databases [9].

Initial object-oriented databases simply consisted of object-oriented languages that had been extended to provide simple persistence, allowing application objects to persist between user sessions. Minimal database functionality was provided in terms of concurrency control, transactions, recovery, etc. More database functionality was added as the systems matured. A more optimized system will maintain the object-oriented properties of the stored data while improving the data accessing and manipulation functions found in current database management systems [9].

Object-oriented databases are a marriage of object-oriented technology and database technology. An OODBMS must maintain the properties of both object-oriented technology and relational technology. This means characteristics such as *encapsulation*, *inheritance*, and *polymorphism* must be preserved for all objects stored in the database. In addition, database functionality must also be implemented. A number of specific issues arise as a result of the combination of technology, including persistence, relationships, composite objects, and referential integrity.

*Persistence* is the characteristic that makes data available across executions.

There are several models, depending on the specific OODBMS, for implementing persistence. One implementation model calls for persistence to be based on an object's class, which means that all objects of the same class automatically become persistent if the class is persistent. A second implementation model specifies the persistence of an object at the time of its creation [17]. Each object has its own persistent characteristic that may be different from the other objects of the same class. A third implementation model calls for all objects reachable from an already persistent object to be persistent as well. This type of system would require objects to have a method to explicitly set a particular object's persistence [10].

*Relationships* are an essential concept of the object-oriented paradigm. It allows objects to refer to each other and result in networks of inter-connected objects. These references between objects directly and efficiently models relationships, and is one of the major improvements of the object-oriented data model over the relational data model. Relationships in object-oriented databases can be represented as attributes of the class from which the relationships arise, or as independent objects, or as hidden data structures attached to the owning object.

A *composite object* is a feature of object-oriented databases that is not available in relational databases [10]. Composite objects refer to those objects that contain other objects. The relationship represented here can be interpreted as an is-part-of relationship. This is a natural extension of the property of objects. One object may "contain" another as one of its attributes. New behaviors and operations can be defined for this type of hierarchy. For example, operations applied to the 'root' object that can be propagated to

all objects associated with the root may be necessary, such as Identifier-Equality, Shallow-Equality, and Deep-Equality operations [10]. These forms of equality checking compare object identifiers, attribute values, and attribute values of component objects, respectively. All OODBMS must provide these types of operations to ensure maximum functionality of the system.

*Referential integrity* becomes necessary as a result of relationships between objects. Referential integrity ensures that objects do not contain references to deleted objects. With unidirectional relationships, referential integrity is not easily accomplished. Application level solutions such as maintaining a dependency list exist but result in poor performance. Given bi-directional relationships, referential integrity can be automatically provided because related objects can be efficiently found and deleted as well. One method of tracking deleted objects is called “tombstoning”. The system does not explicitly remove an object but marks the object when it has been deleted. When a reference to a deleted object is made, it can then be trapped as an error, or simply ignored, with appropriate update of the referencing object’s relationship [10].

### **1.5 Decision Support Systems (DSS)**

A Decision Support System (DSS) is any computer-based information system that helps decision makers utilize data and models and other tools to identify and solve problems as well as make decisions. To understand a DSS, one must understand the human decision-making process and the different kinds of problems business management tries to resolve.

In general, business problems can be classified into *structured*, *semi-structured*, and *unstructured* decisions [3]. Structured decisions are sometimes called programmable tasks. They have well-defined and standard operating procedures for the execution of these types of decisions. Examples of structured decisions are record keeping operations, payroll, and other types of inventory problems.

Semi-structured decisions are not quite as well defined as structured decisions. They do, however, have structured aspects that would benefit from information retrieval, analytical models, and other information-system technology. Examples of semi-structured decisions are sales forecasting, budget preparation, and capital acquisition analysis. These decisions are more “difficult” than those that are structured, though DSS can provide alternatives and options for the decision-makers.

Unstructured decisions are defined as having no standard operating procedures that pertains to their implementation. In this circumstance the DSS offers the least amount of support. The majority of the decision support responsibility is up to the end user. Completely unstructured decisions rarely occur in nature, though instances can be found in research and development phases of projects.

Depending on the type of operation performed, a DSS can range from extremely data oriented to extremely model-oriented systems, as illustrated in Figure 1.2. A DSS can be a management information system that just imports and displays information in an organized form, a mathematical programming package, or Executive Support Systems, etc. The variety of operations can include the following [1]:

- Retrieving a single item of information. This is the simple database system that allows immediate access to data items.

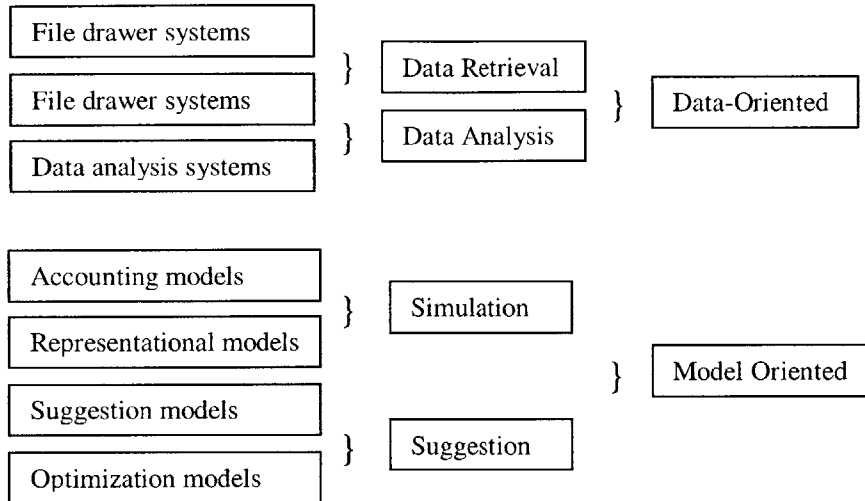


Figure 1.2: Data Orientation vs. Model Orientation in Decision Support Systems [1].

- Providing a mechanism for ad hoc data analysis. This allows the manipulation of data by means of operators tailored to the task and setting operators of a general nature.
- Providing pre-specified aggregations of data in the form of reports. This type of system provides access to a series of databases and small models.
- Estimating the consequences of proposed decisions. This is a very important category of decision support systems. In fact, recent and more restrictive definitions include systems in this category only. Different models can perform different tasks. For example, accounting models calculate the consequences of planned actions on the basis of accounting definitions. Representational models estimate the consequences of actions on the basis of models that are partially nondefinitional. Optimization models provide guidelines for action for generating the optimal solution consistent with a series of constraints.
- Proposing decisions. Suggestion models perform mechanical work leading to a specific suggested decision for a task.

Conceptually, a DSS can be thought of as composed of the one or more of the following components, as shown in Figure 1.3 [13]:

- Data Management: This includes the raw data usually stored in database management systems.



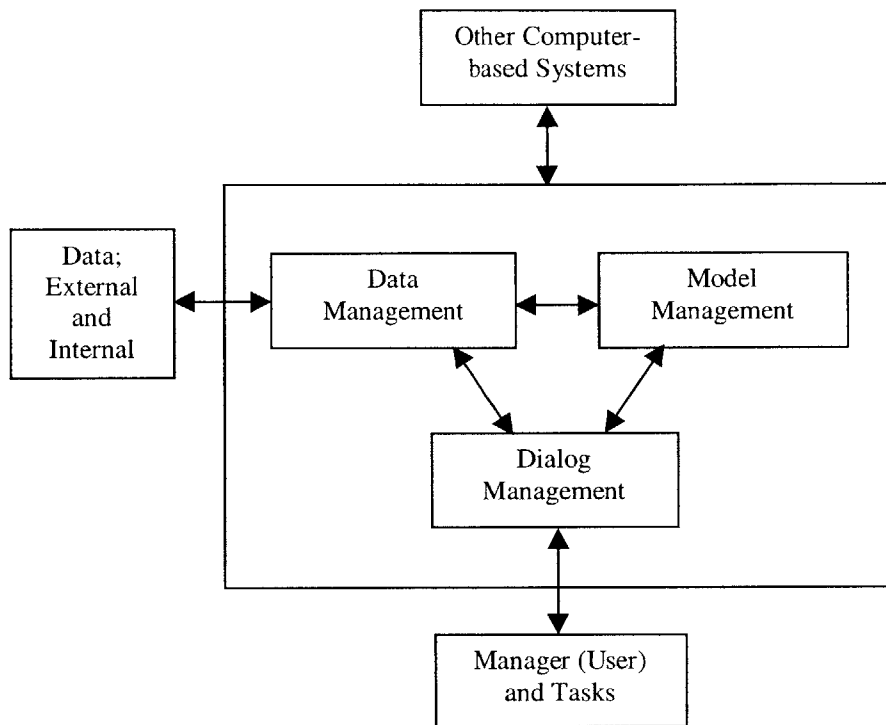


Figure 1.3: Conceptual Model of DSS [12].

- **Model Management:** This is the feature that organizes and analyzes the data stored in the DBMS. The results produced here will affect the decision making of the human user.
- **Communication Subsystem:** This is the subsystem through which the user can communicate and command the DSS.

A decision support system is very important tool and concept in information technology. They help business professionals make better decisions, and have become a valuable tool in the scientific and business communities.

## **1.6 Outline of Thesis**

In Chapter 2, the differences between various aspects of relational databases and object-oriented databases are explained and explored. In Chapters 3 and 4, two DSS software applications that currently use relational databases are investigated in detail. For each system, a design of a relevant object-oriented database to replace the relational database is proposed and described. Chapter 5 presents a summary of this thesis and outlines possible future research in related fields.

## Chapter 2

### OODBMS Applications vs. RDMS Applications

Relational databases and object-oriented databases are two types of software data storage and retrieval systems. There are a number of systems that can utilize either of these two database systems as the underlying method of data storage. Thus, it is essential that the differences between the two systems be examined. This chapter compares feature by feature these two systems from several technical and usability viewpoints, including the methods used to store data and manipulate data in each technology, as well as the impact of technology choice on database application development.

#### 2.1 Types of Stored Data

Relational databases use primitive types to store data. The data types used in a typical Oracle database, for example, include number, char, varchar2, date, raw, rowid, etc [9]. The meanings of these data types are generally self-explanatory. The *varchar2* type is a variable-length character string. The *raw* type represents fixed-length binary data that the system itself does not understand. *Raw* type data usually indicates binary audio or graphics files. The *rowid* type is used for a binary number representing the

unique database address of a row in a database. Other relational database systems maintain similar data type structures.

Object-oriented databases store objects, which loosely speaking are conceptual collections of other types. An object is an instance of a class defined by the application. Classes can be arbitrarily defined and have components of many types, whether primitive data types or other object types. Object-oriented databases cannot assume any patterns in the objects being stored. So, in a sense, it has to be more “encompassing”. Object-oriented databases should contain all the functionality of relational databases and more.

Theoretically, the functionality of an optimally developed OODBMS encapsulates the functionality of the RDBMS. This is because an object-oriented database not only embodies the object-oriented properties of current programming languages but also the database functionality and relationships of relational database systems. However, this phenomenon has not become reality yet as object-oriented databases are still a developing technology and many of its features have not become mature [10].

## **2.2 Data Manipulation**

In relational databases, data manipulation is performed through the use of Standard Query Language or SQL. SQL is used for querying and updating database tables. Tables form the input and output of SQL queries. The query evaluation is based on comparison of attribute (column) values placed in the search condition (query predicate). Prior to evaluation, each query can be optimized based on its syntax, existing tables and storage structures. Using features such as indexes further optimize relational database queries.

Since SQL has become a standard in the database world, the development of a Data Manipulation Language for object-oriented databases conformed to many of the syntactic forms used in standard SQL. The Object Query Language (OQL) is the query language used in object-oriented databases, and represents a superset of standard SQL. In addition to the standard syntax of SQL, OQL also includes object extensions for object identity, complex objects, path expressions, operation invocation, and inheritance [17].

### **2.3 Connectivity with Existing Programming Languages**

For database applications written in languages capable of supporting objects, there is a definite weakness in using a relational database to store data as opposed to using an object-oriented database.

In order for relational databases and software applications to communicate, an interface is required since the query language SQL used in relational technology is not suitable for software development. Relational databases have been integrated into application development. For example, ODBC (Open Database Connectivity) is an application programming interface (API) for accessing data in a number of different databases. A driver for each of the database is required in addition to the ODBC software. If the application is written in Java, then JDBC is the appropriate API.

Object-oriented databases mesh easily with the object-oriented languages that are prevalent in software development. In addition, the strengths of object-oriented databases reflect the weaknesses of relational databases. As application development increasingly shifts towards the use of object-oriented languages, objects created by the applications can no longer be stored efficiently within a relational database. In order to map an object

and its attributes into the table format required by relational databases, flattening of the object's structure is required [10]. This process becomes extremely difficult when the object is a composite object, which means that one or more of its attributes is another object. In this scenario, a tremendous amount of effort is needed to store that object, as well as querying for any specific attribute of that object. Using object-oriented databases, however, referencing an object through another object is easily accomplished.

## **2.4 Database Design Techniques**

When designing an application with data to be stored in a relational database, a lot of emphasis is placed on the design of the database. Data to be stored in relational databases are usually primitive types such as string, numeral, or date. The data typically are independent units that are unrelated to each other. Relational databases require that all data must be stored in structures called tables. The table structure enforces a relationship on the atomic data. Since the table is the only structure within the relational database that describes the relationship among the data, efficient table design is critical. Thus, database design theories are very important to RDBMS. A number of theories have been developed for relational database design, of which the normalization theories are most important.

The normalization theories in relational databases were developed to optimize the efficiency of using relational tables. Normalization is the process of organizing a relational database into tables such that the intended result from using the database is always achieved and is unambiguous. This process may have the effect of reducing the duplication of data items within the database, and often results in the creation of

additional tables [7]. Normalization is typically a refinement process after initially identifying the data objects that should be in the database, determining their relationships, and defining the tables required and columns within each table.

Normalization processes tables into normal forms. The *first normal form* corresponds to a database with tables consisting of rows and columns. The first normal form guarantees certain aspects of data integrity. First, there should not be any duplicate data, which means each row must be unique. Second, each column should reflect a property or characteristic of the data. Finally, all elements in the same column must be of the same type. The *second normal form* states that an attribute must depend upon its entity's entire unique ID. If the primary key of a table consists of one single column, then the table is automatically in the second normal form. If the primary key of a table consists of multiple columns, then all attributes must depend on the entire key to make the table satisfy this condition. The *third normal form* indicates that non-key attributes must be independent of any other non-key attributes. This means that all attributes must depend only the key of that table. If an attribute depends upon a non-key attribute, then a separate table is needed to house these data [7].

Normalization theories help database designers create databases that are efficient for data search and query. Redundant data is removed, and the data are split into the most efficiently divided tables. An application programmer developing an application based on a relational database does not necessarily have to understand the logic behind the structure of the relational database. His or her responsibility only involves storing and querying the appropriate data from the database. As database design is a specialized skill, a database administrator would need to work with the application programmer to design

the schema and table structures so that data can be transferred with ease and maximum efficiency.

Due to fundamental differences in the data being stored, object-oriented databases have a different set of requirements. When objects are constructed, the relationship each object has with other objects has been explicitly expressed through data members. Thus the application programmer has already established the relationships amongst the stored data. In that sense, the application programmer actually designs the structure of the data. This is the responsibility of the database administrator in the relational data model. When designing the data model for an object-oriented database, object-oriented design experience is very important. Though the techniques are somewhat different, the spirit of design is similar.

The key to object-oriented design is discovering how to represent objects in an easily understood manner while maintaining efficiency, utilizing the natural hierarchy inherent in object-oriented programming. There exist a variety of module diagrams designed to help object-oriented programmers construct their desired data structures better. These techniques can be selectively used for different types of applications.

*Interaction diagrams* are models that describe how a group of objects collaborate in some behavior – typically a single use case [4]. The diagrams show a number of objects and the messages that are passed between the objects within the use case. Interaction diagrams should be used when examining the behavior of several objects within a single use case. Such diagrams are suited for illustrating the collaboration between the objects; they are not as well suited for precise definition of the behavior, however.



The *class diagram* is a central modeling technique that runs through nearly all object-oriented methods. This diagram describes the types of objects in the system and the various static relationships that exist between them. There are three principal types of relationships that are important: associations, subtypes and aggregation. *Associations* represent relationships between instances of types [4]. For example, an employee works for a company. Association can be unidirectional or bi-directional. *Subtyping* refers to the inheritance property of the object-oriented paradigm. *Aggregation* is the property that describes a single entity within a group of items. Class diagrams outline the relationships with the help of constraint rules. Class diagrams are called the backbone of nearly all object-oriented methods.

*Patterns* are a way to describe key ideas in a system. Instead of expressing an object-oriented design, patterns display the outcome of the process, and explain the logic behind the design. An example would be describing the process and actions of the stock exchange. Scenarios are devised for the return and risks involved with each type of investment portfolio. New scenarios can arise as a result of the rise or fall of individual stock prices in the portfolio. The pattern diagram would illustrate the sequence of events if a particular stock price changes. In other words, the pattern diagram would display different scenarios and the relationships among individual scenarios. An important aspect of a pattern is that it is much more than a model, since it must also include the logic behind the model. A pattern is often said to be a solution to a problem – it must make the problem clear and explain why it solves the problem, and state under what circumstances the pattern valid or invalid [4].

The responsibility of designing the relationships amongst the data in the object-oriented data model rests with the application programmer. The importance of the design of data at the application programming level cannot be overstated. The application programmer, who is experienced in object-oriented design, constructs objects according to application need. Since the programmer understands the relationship between the objects, he can design the most efficient way to maintain the relationship structure and querying methods.

## **2.5 Types of Applications**

Since relational databases and object-oriented databases store fundamentally different types of data, the applications constructed for each of these database systems are very different. Applications based on relational technology will often process simple data such as numerals, strings, etc. This makes relational technology more useful for administrative applications that keep track of records of employees or students. These records usually only contain data having primitive data types. The volume of data is very large as these types of records usually come in large quantities. Relational databases are well suited to handling large quantities of simple data. Features such as indexes are designed to enable speedy query through large data sets. SQL also enables fast querying through the data.

The relational data model, however, has some significant shortcomings:

1. The relational data model is too simple for modeling complex entities that may be nested. For example, the relational data model does not provide mechanisms to handle engineering objects involved in CAD or CAM.
2. The relational database system usually only supports a limited set of data types. In Oracle 7, for example, the data types supported are: char, varchar (string), number, date, raw, long raw, long, rowid, mislabel. Of these, the raw types are intended for the stored binary data that cannot be interpreted by the database.
3. Performance of the relational database is unacceptable for various types of computationally intensive applications such as simulation programs.
4. The database language and application programming languages are fundamentally different. The goals of a database language are query speed and data integrity. Programming languages, however, are designed for the purpose of computation and the speed of computation. Using a database language to retrieve data and a general-purpose programming language to construct an application operating on the data creates a mismatch. The model of transactions supported in relational database systems is inappropriate for transactions in interactive, cooperative design environments [5].

Object-oriented databases are advantageous for applications in the following scenarios. First, object-oriented databases are beneficial when the application is written using object-oriented languages. The reason behind this has been repeatedly emphasized. Data manipulation is done more gracefully when the application, written in an object-

oriented language, stores the related data in an object-oriented database. The application may need to utilize these objects. The second scenario involves an application written to manipulate complex data structures. Typically those data are in the form of objects and mapping them into a relational database table is unacceptably complex and inefficient. An object-oriented database provides an efficient means of accessing referenced objects and attributes.

## **2.6 Summary**

Relational databases and object-oriented databases each have their own advantages and disadvantages. Relational databases can store large amounts of data in form of tables. Querying simple numerals or strings is fast due to the implementation of the database system. Relational databases, however, are not efficient when used as a storage mechanism for object-oriented applications. The amount of effort required to flatten out the natural object hierarchy and map the object and its attributes to a table makes relational databases unfit for applications that involve data with complex structures. Object-oriented databases are better suited to store objects that may have complex structure. Since object-oriented programming techniques are in widespread use, object-oriented databases are a natural fit for data storage and manipulation for today's software engineering needs. However, object-oriented databases are still an immature technology. There remain many technical issues to be resolved, such as adequately efficient storage and query. Also, there has not been one dominant object-oriented language that would establish a de-facto standard, which causes application programmers to hesitate using object-oriented databases in their design and implementation.

## **Chapter 3**

### **OODB and DVI Knowledge Suite™**

This chapter presents a case study of decision support software called DVI Knowledge Suite™ developed by Data View, Inc. Section 3.2 of this chapter contains an evaluation of the DVI Designer portion of DVI Knowledge Suite™. Software evaluation is done with a usability study that includes both user interface and the functionality of the system. The remainder of the chapter describes a proposal for the implementation of the Knowledge Suite™ software with object-oriented database as the underlying storage mechanism.

#### **3.1 MITRE**

Portions of this chapter describe a software evaluation study done for MITRE Corporation's Distributed Computing Group. MITRE Corporation is a non-for-profit organization that works closely with the federal government. MITRE uses technologies in system engineering and information technology to provide solutions in various technical areas of defense interest. The Distributed Computing Group at MITRE is responsible for the investigation of networked computing systems and its application for both projects

with federal government and internal to MITRE. Research areas of this group include XML, database systems and data warehousing, Jini, etc [18].

### **3.2 Background on DVI Knowledge Suite™**

The DVI Knowledge Suite™ is a knowledge visualization tool that allows the end user to create a graphical representation of desired data. The target data is deposited in a relational database and the final graphical representation is a Java applet that can be viewed through a web browser.

#### **3.2.1 Target Users**

The DVI Knowledge Suite™ is a knowledge visualization tool that presents data in a graphical form. The anticipated DVI Knowledge Suite™ user profile is broken down into three categories: Business Analyst / Expert, End User, and Systems Analyst. It is important to note that the skills defined for each of these categories could be contained in one person within any company.

The Business Analyst / Expert is responsible for developing the graphics, attributes and underlying values that the end user sees when using the DVI Knowledge Suite™. The business analyst is recognized within a company, department or work group as a person with thorough understanding of business rules and processes. This user creates and maintains the DVI Knowledge Suite™ graphics and hierarchy, converting business rules resulting from consulting engagements or audits, key performance indicators, regulatory and/or safety compliance issues into the graphical interface used by the end user.

The End User uses DVI Knowledge Suite™ to view graphics and attributes established by the Business Analyst who is supporting the End User's business area. The end user can examine data from within the graphic displays generated from the DVI Knowledge Suite™, which can highlight data fields that bear investigation. End users will primarily interact with the Navigator module of DVI Knowledge Suite™ by using a DVI certified Internet browser to view standard HTML generated by Navigator.

The Systems Analyst provides minimal but critical support to business analysts and end-users by allowing access to applicable data to support DVI Knowledge Suite™ business objects. This support could range from establishing security access via the DVI Administrator to supporting databases for DVI Knowledge Suite™ business rules.

### **3.2.2 DVI Knowledge Suite™ Structure**

The DVI Knowledge Suite™ allows the end user to directly manipulate data through its DVI Designer component. The output of Designer is a Java applet that displays desired data with specified properties. The Knowledge Suite™ application interacts with multiple software components, such as an HTTP server and the Database server. The end user utilizes Designer to customize the knowledge visualization application by combining data, business rules, and graphics rules. Please refer to the next section on the structure of the DVI Designer. Figure 3.1 shows the application architecture of the Knowledge Suite™.

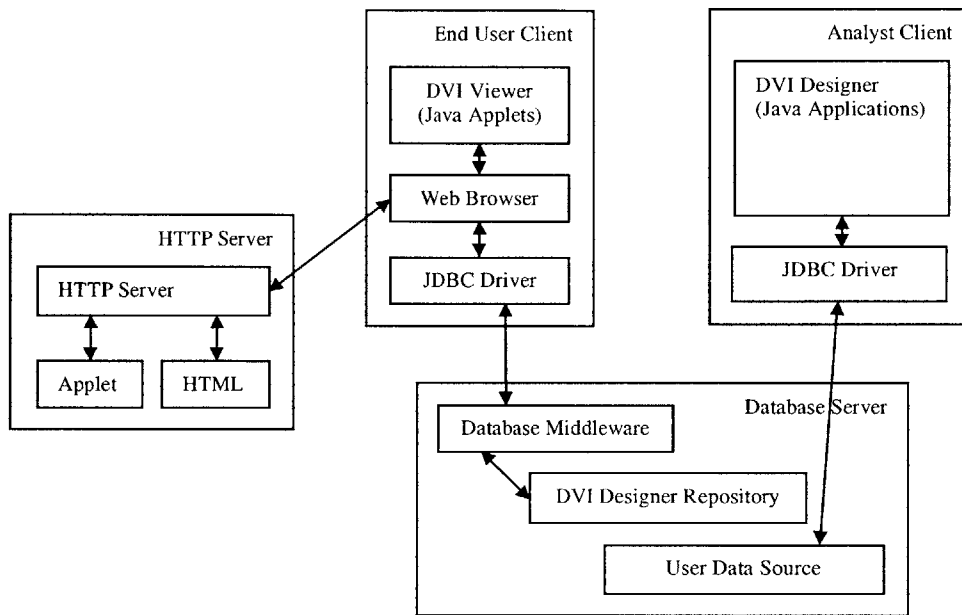


Figure 3.1: Application Architecture of DVI Knowledge Suite™ [14].

The Designer is the most important part of DVI Knowledge Suite™, consisting of five distinctive parts: the Database Organizer, Graphical Organizer, Rule Painter, Portal Painter, Page Organizer, and Navigator Painter [14]. Their interaction with each other and external data sources are shown in Figure 3.2.

- Database Organizer - The Database Organizer is used to establish the connection between the DVI Knowledge Suite™ and the external relational database where the actual data is stored. Host and driver properties for the database are defined by the Database Organizer. Since the Knowledge Suite™ is a pure Java application, JDBC is used to make the connection between the application and the database.
- Graphic Organizer – Every application designed must have at least one underlying graphical image. For example, this could be a structural diagram of a computer network whose computer activity is being tracked. The Graphic Organizer optimizes the use and reuse of graphics objects.



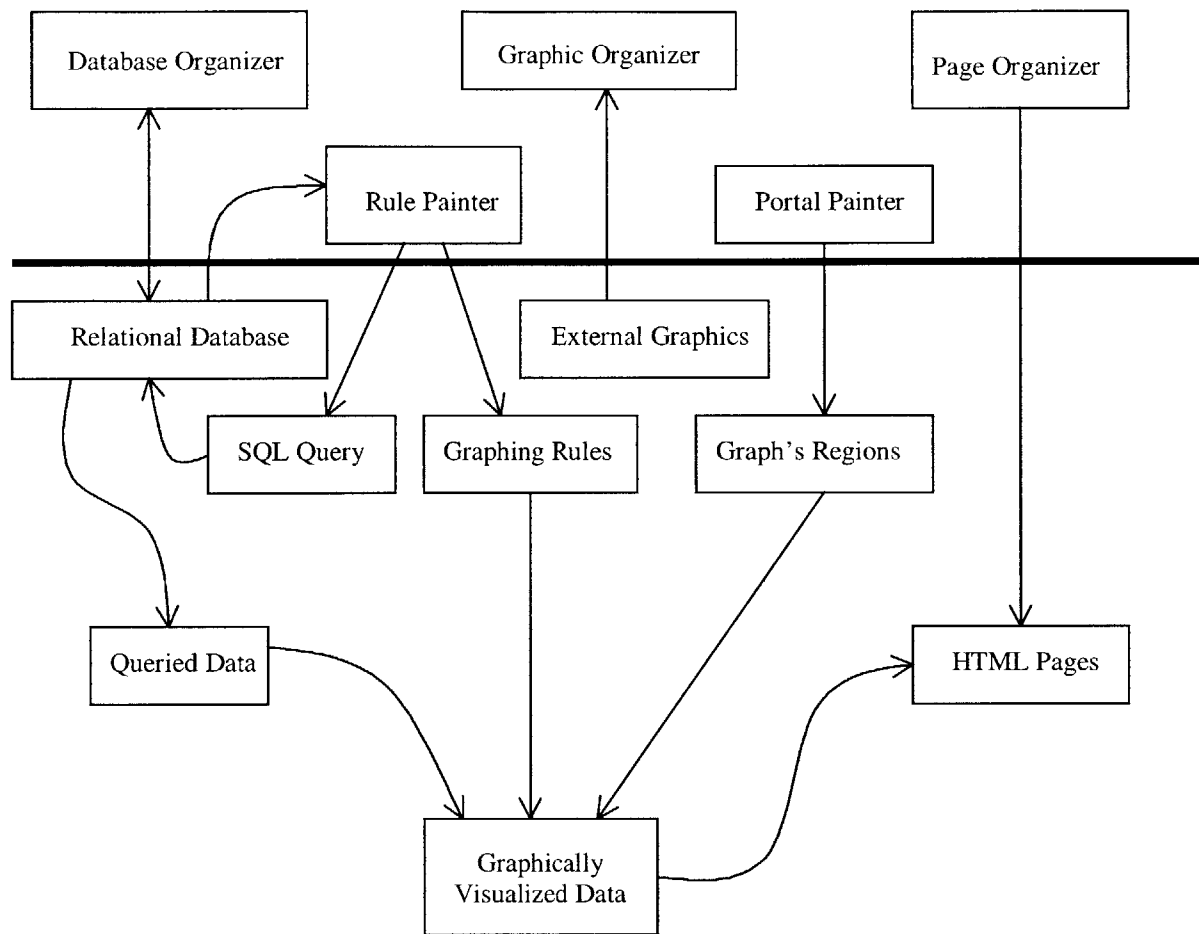


Figure 3.2. Structure of DVI Designer

- Rule Painter – The user defines the rules that will affect the look of the Java applet here. There are two different types of rules. The first type of rules are SQL query statements constructed inside the Rule Painter to extract the necessary data. The second type of rules are business rules that tie the graphical representation with the interpretation of the extracted data. Business rules are constructed graphically, and they have structure such as, “if tire1.age > 5, then region1.color = red”. The rules in Rule Painter are constructed graphically, and have no set syntax. These rules result in an application that enables rapid analysis and quick decisions. What happened, why and the corrective actions required are presented in a logical sequence to assure that the business will behave as management intended.
- Portal Painter - The Portal Painter outlines regions of interest in a user-defined graphic, which was defined by the Graphic Organizer, and supports association of rules with these regions. For example, the end user uses the Portal Painter to outline the specific computers of interest on the computer network diagram.

- Page Organizer – Page Organizer helps creating a web page that contains the knowledge presentation applet created. This is a totally automated service and the end user does not see the page creation process at all.
- Navigation Painter – This creates an application web page hierarchy that places the application in the appropriate directory.

Since this product aims for end users with very limited technical background, the application creation process is quite simple. All that the end user has to do is go through the above listed components one by one and follow simple instructions given in the instruction manual.

### **3.3 EVALUATION**

#### **3.3.1 CRITERIA**

The foremost criterion for evaluation of DVI Knowledge Suite™ is the applicability of this software for MITRE's stored data. In a technically specialized environment, custom software packages are often designed to accommodate specialized needs. For example, there is special graphical tool that displays the liquid flows.

The second criterion is the “look and feel” of the software. With Microsoft's dominance in the software market, customers have come to expect a “familiar” look for Windows software. For example, usually the menu bar is the top horizontal bar in a Windows based application. Giving the customer this familiar look shortens the learning curve and makes the software more user-friendly.

A third criterion is the functionality of Knowledge Suite™. The end user is likely to have a variety of specific requirements. These needs may or may not be addressed by the target software. Thus, it is a good measure of software quality. Ideally, a piece

software should be able to satisfy all the needs of the customer. But it is a difficult task considering the specific needs of the customers and the generics of the software [11].

### **3.3.2 PROCEDURE**

- Investigation of the necessity and the applicability of DVI Knowledge Suite™. Data from multiple sources must be acquired and analyzed for their usability in the Knowledge Suite™. A conclusion is reached at the end of this section of the investigation that DVI Knowledge Suite™ is a knowledge presentation tool appropriate for presenting data in the desired format.
- Once the above investigation returns a positive response, the actual investigation of the value proposition of the target software begins. The first stage of experiment is having an experienced programmer, using the current internet and database technology, construct an intelligent web application that mimics the behavior of the application constructed using the Knowledge Suite™. Technology used includes Java, JavaScript, and CGI scripts. Critical data such as the amount of time used to construct the application, and the ease of responding to change in requirements.
- The professional programmer will construct a very similar application using the Knowledge Suite™. The aim is to compare and contrast, from the point of view of a programmer, usability of Knowledge Suite™.
- After obtaining survey data by the professional programmer, direct involvement of end user is initiated. The end users get familiarized with the Knowledge Suite™ and they are told to construct the desired application themselves. Again, data such as the amount of time of usage and the ease of using Knowledge Suite™ are collected.
- Analysis is done afterwards.

### **3.3.3 Previous Analysis**

The DVI Knowledge Suite™ was a recently released product that has been on the market for less than one year. Thus, there are few reviews and studies of this product.

The few sources of reviews were either special interest computing magazines, or generic computing magazines briefly summarizing the functionality of Knowledge Suite™. A

number of positive and negative comments were given regarding various aspects of the system, as well as the overall functionality of the software package.

Positive comments regarding Knowledge Suite™ touched upon the simplicity of use and sophistication of the technology involved. All of the sources praised the use of Java in building the system. The generation of a Java-based applet that can be viewed through a standard web browser has also been lauded.

One advantage of DVI Knowledge Suite is that it gives the end-user complete control over the visual output of the application [19]. Knowledge Suite™ allows the user to change all aspects of the application's graphic display. The Graphics Painter component of Knowledge Suite™ is where graphics manipulation is performed [19].

Another advantage cited by multiple reviewers concerns the ability of Knowledge Suite™ to enable the creation of a Java application without much technical knowledge of programming in Java on the part of the user. The end-user can begin creating simple Java applications quickly after familiarizing oneself with the product.

The graphical tools found in Knowledge Suite™ found favor among product reviewers. The Rule Painter component allows users to construct SQL query statements without having much knowledge of SQL itself. While this can allow relative novices to quickly build applications, this could also be a disadvantage because the Rule Painter limits the query input form to graphical input only, and denies textual SQL query input. Thus, users with SQL experience may find Rule Painter more of a hindrance than a benefit [19].

The application utilizes a technique called "push" technology, which provides users with real-time data display. Knowledge Suite™ also includes caching capabilities

to reduce client memory and resource usage, and to prioritize activities. Other features such as the ability to schedule data caching during off-hours and to differentiate between cached and uncached data give the Knowledge Suite™ more power [20].

Finally, the Knowledge Suite™ package has been tested against all major database vendor products, including products from Oracle, Sybase, IBM, and Microsoft [21]. Thus many relational database products are supported.

One reviewer had particularly high praise of the potential usability of the system:

The DVI Knowledge Suite may be used by any organization in any industry to track, account for, administer, maintain service or report on attributes of any object. Users can access the system from any location at any time from any industry standard Web browser.

This suite is essentially a "fifth-generation" software tool. It has point-and-click interface and requires no scripting, which reduces the level of effort by users [22].

Negative comments mainly focused on the lack of overall functionality, insufficient data security, and high cost of the system. For example, the method of displaying the hierarchical structure of the objects being tracked was sufficient for a simple application with a few objects and web pages. However, the ability of Knowledge Suite™ to handle the navigation of applications with numerous objects and web pages elegantly was questioned [19].

Security of access to data created by Knowledge Suite™ was another point of concern among reviewers. There are multiple levels to the security mechanism of application created in by the system. One level of security is portal security, where the

DVI administrator is in charge. The administrator can decide which users or user groups have access to the application. A second level of security is the security measures of the underlying database. At the second level, the application itself does not protect the data in the database. Thus, only the measures that were already in place for data security in the relational database are available. The fact that the only data protection comes from the underlying database does not meet the approval of the reviewers [20].

Finally, the cost of DVI Knowledge Suite™ is a serious concern among the reviewers. The cost of a 25-user license is \$25,000. For such high cost, a more functionally complete package is desirable [19]. The high cost also would limit the number of users who would be able to afford the license and thus the potential impact of the system across the industry.

There is general consensus that Knowledge Suite™ provides a number of useful features, and its implementation in Java recognizes the advantages of the object-oriented paradigm. However, further refinements to the software are needed to address certain concerns about system functionality.

### **3.3.4 Analysis**

Due to the continuous release updates from DataView, Inc. on their software, some of the comments here may not remain valid.

The first issue that is being addressed is software applicability for the MITRE Corp. There are myriad sources of statistical data generated through the customers of MITRE. There are a variety of reasons that a limited number of data sources are available to investigate. Some data are too technical and cannot be stored in a generic relational

database. Some are sensitive information and inappropriate for access through the Internet. The data source that was determined appropriate in terms of data type was the administrative and employment information accumulated over many years. Such data is kept in relational databases and analyzed for purposes of business efficiency and identifying business trends. The final decision was made to use data from the Air Force Center Daily's metrics database. The Center for Air Force Command and Control Systems (CAFC2S) has a sizeable database containing data stored in relational databases, and which is queried to generate graphical representations on the CAFC2S metrics homepage. These graphs are used by the Air Force personnel to review performance. There are seven categories of data: Project Leadership, Business Performance, Staffing and Training, Customer Satisfaction, Technology and Infrastructure, Strategic Planning, and Integrated Command and Control System. Twenty-nine different items are graphed using the metrics database.

DVI Knowledge Suite™ was discovered not well suited to the desired uses of the data. Unfortunately, the DVI Knowledge Suite™ can only design an application that generates only one type of graph, namely an organizational program status using the “stop light” method, namely displaying different colors for different status. It is clear that MITRE's data sources do not fit the type of data Knowledge Suite™ expects. As MITRE looks for business trends and movements, the desired representation should be line and bar graphs. The need to indicate specific numerical values also diverges from Knowledge Suite™'s strong areas.

The second requirement is GUI compatibility and conformity. Knowledge Suite™ follows the basic concept of a Windows application. The initial layout of the features

seems reasonable although the look of the program is not extremely sophisticated. The icons and buttons, however, are somewhat confusing at times. They are custom created icons and without word labels on the button, it is difficult to understand its meaning. The buttons are not of the traditional Windows button with a 3 dimensional clicking area. Instead, it is either a character string or a picture icon that gets highlighted when mouse pointer goes over the image. This can create some confusion for the user if he is new to the application and does not recognize the icon as something click-able.

Another problem with the GUI interface is the speed at which features operate. Because the Knowledge Suite™ is implemented with Java for multi-platform compatibility, application performance is sacrificed. This is especially visible using the Viewer. It takes more than one minute for the initial viewer screen to get connection with the database with JDBC and set up the initial screen.

The third criterion is concerned with the overall performance of this software. This is perhaps the most important critique as it affects the success of the product the most. The concept that data + graphic + business rule = knowledge is a novel idea. It can potentially bring some very interesting ideas for software applications. DataView, Inc. has also displayed vision for the future in employing Java as the programming language to construct the Knowledge Suite™.

However, many criticisms can be made of the Knowledge Suite™. The biggest one is, perhaps, its lack of data manipulation ability. As of the present version, there is still no way to call any SQL functions. This is a glaring deficiency as it limits functionality to data lookup in existing tables. Another deficiency is the inability to look up Views in the underlying relational database. The view in a relational database is very



important as it allows manipulation of data in different tables. Using the software package requires the additional purchase of a JDBC driver. This adds to the cost of the end user and increases the complexity of both the installation and the usage.

There are some redundant features on the application. For example, every time a new application is added the data source host name and the data source driver, and the viewer hostname and driver must be specified. This is quite unnecessary considering it is most likely that a new application created would utilize the drivers and the hosts used in the previously constructed applications.

One interesting property of the software is its security mechanisms. Upon start up of the designer, a username and password must be entered to connect to the database. This username-password set, however, does not access the database where the target data is. It is a password that protects the application being used by a non-authorized user. A separate prompt is displayed when the user needs to access the “real” data. This feature is somewhat confusing. It has not been indicated anywhere that the first set of username/password is merely meant for start up of the application and not to the real data and that adds to the confusion. In fact, the first password is quite unnecessary. If the end user cannot obtain access to the desired data, then he cannot construct the application successfully. There is no need to guard the another user from even starting up the application.

The biggest complaint from the users regarding the Knowledge Suite™ is the limited functionality of this application. MITRE is interested in graphs that exhibit trends and dynamics. The best graphs that accommodate these requirements, unfortunately, are the conventional line graphs and the bar graphs that the Knowledge Suite™ avoids. The

Knowledge Suite™ does not generate any graphic features and relies solely on the static graphic the end user chooses

It takes very few lines to put a static graphical image on a Java applet and designate a certain area to change colors when a query into the database returns a certain value. It is perhaps more efficient to do so, given that the end user can just type up the query, type in the defined region quickly, compile, and view. Writing the Java applet gives more flexibility to the end user as well. When Java calls SQL statements, no limit is put on the kind of queries. The behavior of the graphics can also be manipulated as much as the Java language allows. In short, DataView must expand the capability and flexibility of its software. It has to make a “complex” procedure easy.

### **3.4 Application of OODB to DVI Knowledge Suite™**

Presently the DVI Knowledge Suite™ only supports data retrieval from relational databases. The application and the construction of a DSS using the DVI Knowledge Suite™ is also based on purely relational databases meaning that essential application related parameters are stored in the relational database. The capability of the system can be greatly extended if DVI Knowledge Suite™ can accommodate object-oriented databases.

There are several reasons why the inclusion of object-oriented database is a beneficial direction for the Knowledge Suite™. First, because more and more application programs are constructed using object-oriented languages, data in the format of objects are becoming more prevalent. Since the Knowledge Suite™ usually presents a graphical view of a physical object, the inclusion of object-oriented database with objects is quite

necessary. Another reason for the advised inclusion of object-oriented database in the Knowledge Suite™ is that the object-oriented database may provide more efficient means of storage and query for object data. The object structure and hierarchy must be flattened out and mapped to tables for it to be used in a relational database application. If only a relational model is provided, the application may have to do repeated query from database to either retrieve or store the relevant attributes of objects. This causes applications with complex objects to be slow and inefficient.

### **3.4.1 Design Requirement**

#### Incoming Data

The issue of database choice occurs in two areas. The first is the user data being passed in. This is the data that the user is most interested in tracking. The new design would have the application supporting objected oriented databases in addition to or replacing the relational database already supported. The modified Knowledge Suite™ should be able to retrieve data stored in a particular OODB through the specified querying method. For example, if information regarding a truck was stored in an instance of the class **truck**, then it is expected that the modified Knowledge Suite™ Designer should be able to extract the appropriate data member functions and utilize it however the end user wants.

#### Resulting Data

The second issue is the storage of the resulted project data. Currently, the resulting application, which is a simple Java applet that displays the processed data, is stored on the local hard drive and the Knowledge Suite™ simply keeps the applet name

in the database. There are many modules used by the DVI Designer that can be organized and stored into OODB. For examples, for every highlighted region added to the background graphic, an shape object can be created for that area detailing the shape, size, color, and the specific data concerning that marked region.

### Business Rules

Another feature of the DVI Designer that can be incorporated into object are the business rules that connect the data and the graphic display. These rules are in the form of “if data-predicate then graphic-property”. The data-predicate portion of a rule is a comparison statement that qualifies the data in a desired manner. The graphic-property describes how the specified region would behave if the data-predicate has been evaluated to true. For example, a graphic-property could be `circle0.changeColor()`, which changes the color of the circle.

Redesigning the DVI Designer to incorporate using an object-oriented database must consider the three requirements listed above. All three components can be grouped into objects and stored in object-oriented databases. Figure 3.2 demonstrates the interaction among the three main modules.

### **3.4.2 Data Object Components and Behavior**

As indicated above, there are three top-level objects to be considered for a successful implementation of a back end system to the Designer. These objects and classes must conform to a set of standards before all components can function seamlessly.

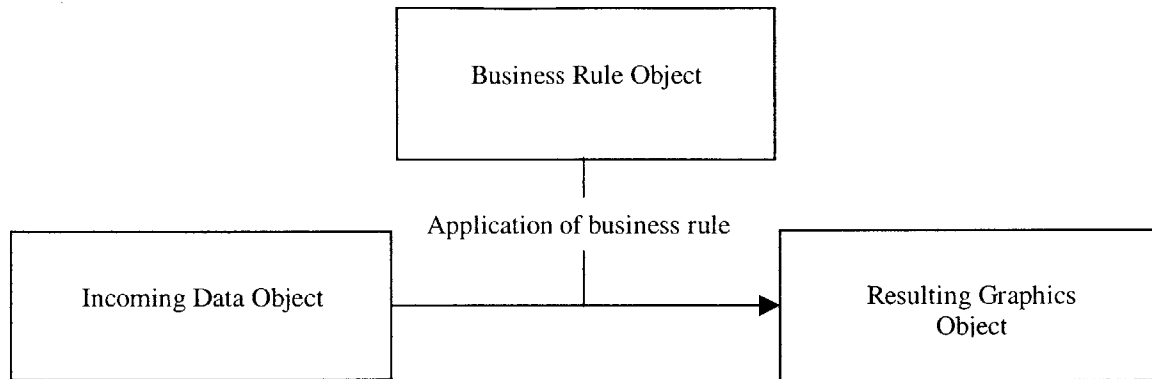


Figure 3.3:  
Interaction among DataView objects

### Incoming Data Object

The incoming data object contains the information that the end user is interested in and can come from any number of external sources. The Designer itself has no control over the structure of these objects. There are some basic rules that the incoming data object must adhere to for Designer to be able to parse the desired attribute of the desired data.

All of the attributes of the object must be declared as either public data members or private data members with the appropriate public access and manipulation member functions. Helper variables in these classes do not necessarily need to have an access function.

A second requirement is for each object class to contain comparison functions. Just as there are equal, less than, greater than functions for integers, each object class should implement comparison functions for its attributes. The equal operator seems to be a mandatory function. Other comparison functions can be implemented as well.

Because DVI designer does not have control over the content of these objects, a more complete design will avoid fatal errors.

### Rule Object

The rule object is actually two classes of objects. The first contains the Object Query Language used to query into the database. OQL syntax is extremely similar to SQL syntax. The following lines of OQL code is an example of a query:

```
select c.address
  from Persons p,
       p.children c
 where p.address.street="Main Street" and
        count(p.children) >= 2 and
        c.address.city != p.address.city
```

The "dot" notation used in the query traverses the data structure through all the relevant objects. The query inspects all children of all "Persons" to find people who live on Main Street with at least two children. It returns only those addresses of children who do not live in the same city as their parents. It navigates from the Person class using the child reference to another instance of the Person class and then to the Address and City classes.

The second object to be included here is the business rule. As stated before, business rules are those rules that dictate how regions of the graphic behave after comparison with results of the OQL. The end-user uses the business rule object to modify properties of the region object discussed in the next section. Thus, the business rule object must contain the target regions and relevant OQL query as data members. These data members should be declared as private. The reason for this is these regions and the query are very important functions and other classes should not access them directly. The

mandatory member functions in this object class include: comparison functions that compare query results and modification functions that change the graphic properties of the affected region.

### Resulting Graphics Object

The graphics object contains data that displays the graphical result of the relevant user data. It has many attributes that are manipulated by outside functions.

The properties to be included in the graphics include: shape, the coordinates with respect to the shape, color, location on the background graphic, and the specific data associated with that region.

The member functions of the graphics object class include functions that updates the shape, color, location of the desired regions.

## **3.5 Effects of OODB on DVI Designer**

With the successful implementation of the object-oriented databases in the DVI Designer, the DVI Designer functionality can be greatly enhanced. The ability to allow object data as the input data is clearly an advantage. The tolerance of more data types will allow the application to be more versatile. As more and more current data are of the form of objects, implementing a back end database system that allows object data to be stored and queried.

Implementing the internal structure of the Knowledge Suite™ as objects that can be stored in an object-oriented database allows the DVI Designer structure to be much more complex than it has ever been. The application can generate and delete objects while executing the application giving the application more efficiency.

### 3.6 Summary

DataView's Knowledge Suite™ is knowledge presentation software that benefits the end user by allowing him to directly design the final display application rather than requiring the services of a professional application programmer. This concept allows fast generation of the display application and saves tremendous amount of time and financial resources for the end user.

This software has the potential to be useful to the end user, although Knowledge Suite™ may not be recommendable for a corporation such as MITRE Corporation. Its weakness lies mainly within the fact that the DVI Knowledge Suite™ does not fit the business needs of MITRE. DVI Knowledge Suite™ expects simple input and provides simple output for a somewhat static data.

The potential utility for Knowledge Suite™ can increase dramatically if the Designer can be implemented with object-oriented databases. There are two factors affecting this. As the Knowledge Suite™ is still under constant development, increased complexity and functionality of the Designer will generate many application related objects. To manage these objects, an object-oriented database is preferable because it meshes with object-oriented programming more naturally. A second reason for incorporating object-oriented databases arises from consideration for future expansion when data to be stored will be increasingly in the form of objects instead of raw data stored in tables of a relational database.



The re-implementation of the underlying database object structure will require that the main modules of incoming data, rules, and graphics all become classes. These classes will interact with each other to manipulate both the data and the display.

## **Chapter 4**

### **OODB and Contract Data Management System**

The Contract Data Management System (CDMS) is a software package that helps MITRE's Contracts Office track contracts with sponsors and clients. The current implementation relies on a backend relational database that stores all the data. CDMS is a complex system with many components and complex behaviors. The remainder of this chapter explains the structure and life cycle that contracts go through, followed by an argument for replacing the underlying database used by CDMS with an object-oriented database. Finally, object-oriented database design for the contract system is presented.

#### **4.1 Components and Requirements for CDMS**

The two types of contracts and contract components tracked by CDMS are normal contracts, also known as task orders (TOs), and Basic Ordering Agreements (BOAs). In a Normal Contract, the contract document authorizes MITRE to spend an indicated amount of money as work on that contract is performed. One or more Task Orders (TO) may be issued under a Normal Contract, and describe in detail a specific task to be performed.

(See Figure 4.1) Common types of Normal Contracts can be Cost-Plus-fixed-Fee (CPFF), Firm Fixed Price (FFP), Time and Material (T&M), etc [15].

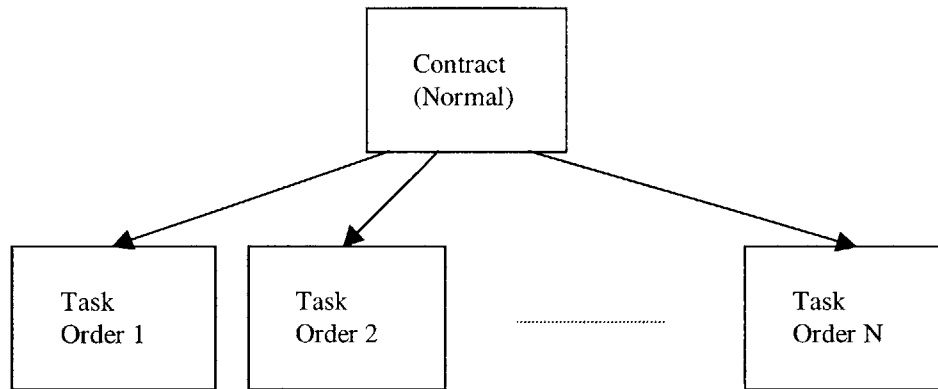


Figure 4.1: Normal Contract [15]

A Basic Ordering Agreement (BOA) has all the essential elements of a Normal Contract, but it establishes broad parameters for execution of a contract without obligating funding or authorizing MITRE to spend money in performance of the contract. As a result, individual orders must be established under the contract that actually obligate money and authorize its use by MITRE. Task orders are then sometimes established under the orders. One or more Delivery Orders (DO) that are established under the BOA Contract are required to obligate money and authorize work. One more task orders may be established under each delivery order. Sometimes a contract is initially classified as a normal contract, and at a later point in time evolves to become part of a BOA contract. The normal contract in essence becomes a delivery order within a BOA. Note that a task order can be issued either under the contract itself, or under a delivery order. (See Figure 4.2)

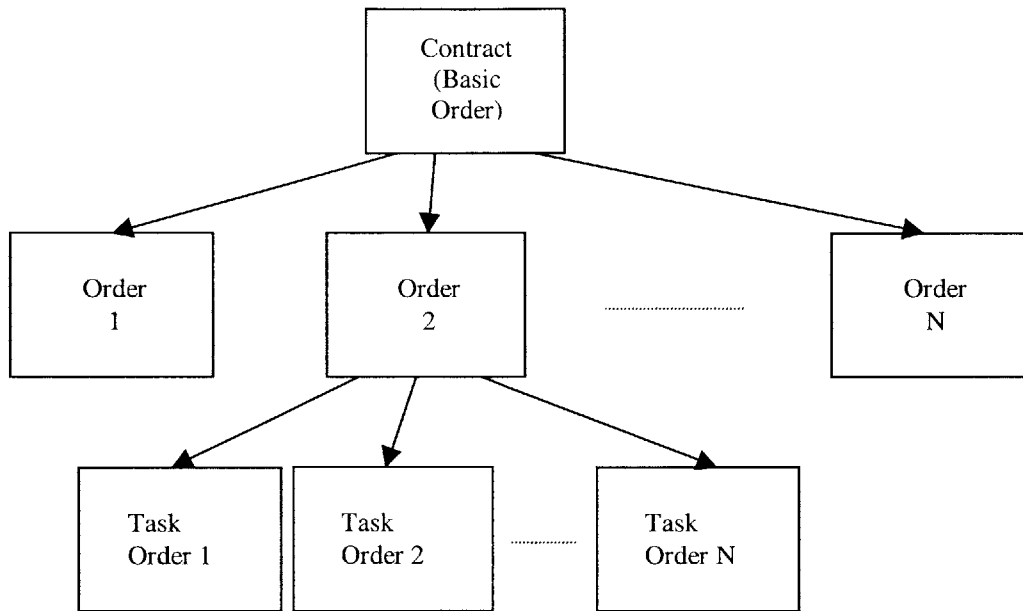


Figure 4.2: Basic Ordering Agreement Contract [15]

Sponsor modifications and other changes can affect the specifics of a contract. Under those circumstances, revisions must be amended to the contracts. The changes can occur to a contract, an order, or a task order. As revisions occur, modified data sheets are kept in the database. Each revision is assigned a version number. (See Figure 4.3)

In order to track a contract, all events that affect the contract must be tracked as well. Actions that must be included for tracking purposes are:

- Creating delivery orders and task orders
- Modifying contracts and their delivery and task orders to reflect
  1. Changing terms and conditions
  2. Changing funding
  3. Changing period of performance
  4. Changing project personnel

There are events that are performed to contracts. Some events are dependent while others are interdependent.

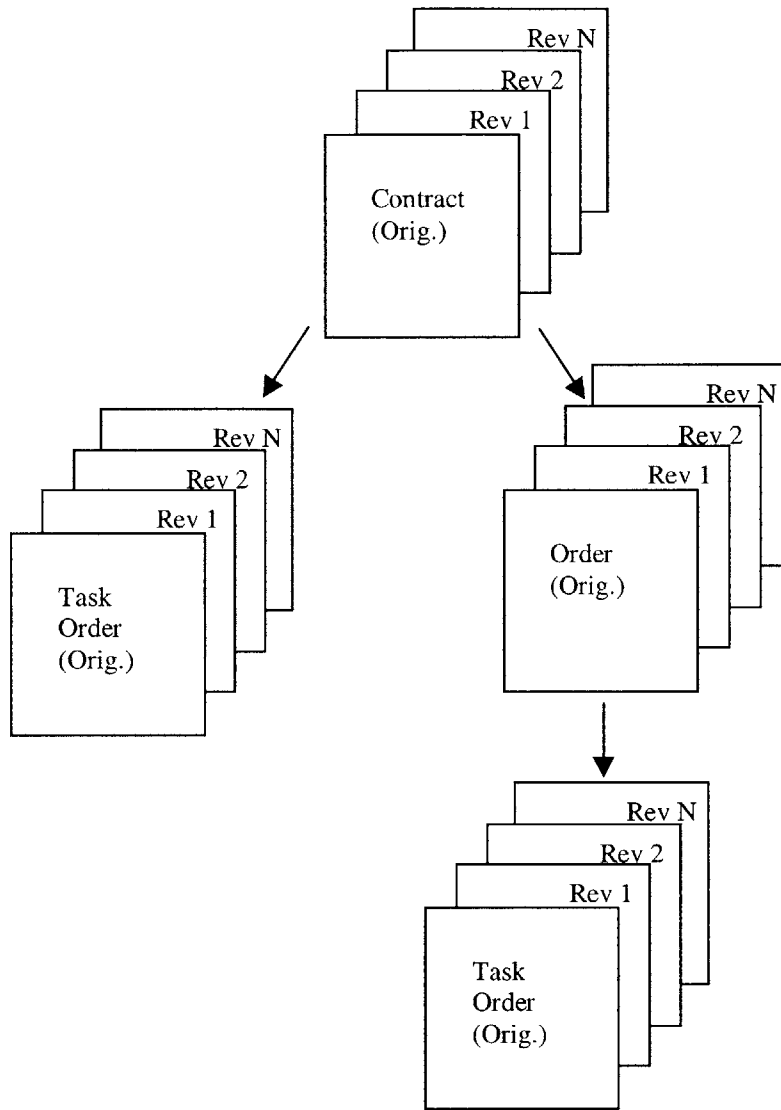


Figure 4.3: Version Control in CDMS

## 4.2 CDMS Functional Requirements

A specific version of a contract data sheet must be identifiable by a combination of CDS number, order number, and revision number. The CDS number is a five-digit number automatically assigned by CDMS when a Preliminary Data Sheet (PDS) is issued for the original version of a contract or when a sponsor paper is logged for the original

version of a contract. The order number is a three-digit number automatically assigned by the system (as the next available order number within the CDS number), when either of two conditions is met:

1. A PDS is issued or sponsor is logged, for the original version of a contract
2. A PDS or sponsor paper is logged for a new order.

Finally, the revision number is a four-digit code automatically assigned by the system when:

1. A PDS is issued or
2. When sponsor paper is logged

**Table 4.1 Contract Lifecycle Status.**

Status	Significant Events
Initiated	Contract Office logs sponsor paper (letter of Intent from sponsor) for new contract
Fully-Executed (FE)	Contracts Office logs sponsor award (i.e. funding) for new contract
Active	Contract becomes active when first FDS approved  Contract and contract components generated and revised.
In-closeout	Contracts Office closeout and performs closeout activities
Closed	Contracts Office closes DOs/contract when closeout activities completed

### **4.3 Analysis of CDMS with Relational Database**

The complexity of the CDMS system makes the system difficult to comprehend and unwieldy to analyze. Each contract being tracked will involve data from over 30 tables in the database. Thus, this system is very prone to error, since the front-end

application programmer can easily lose track of the relevant tables. If any single query of any single contract fails to return a valid result, then an error is logged. As there are a large number of contracts managed by this system, many errors are logged. This comes as a result of the error handling mechanism of relational databases, which is simplistic and direct. When encountering an error, the relational database suspends the current query process and logs the error. This error handling mechanism is somewhat primitive and lags behind the mechanisms used by object-oriented languages. For example, the ‘throw’ and ‘catch’ key words in Java are mechanisms where error handlers can be written to manage anticipated errors, or propagate errors to a higher-level object. This method is beneficial to the application programmer as it makes identification of the source of the error easier.

Part of the reason CDMS is very error prone is due to its complex structure. A system that lacks hierarchical structure can have behaviors that are difficult to debug. When multiple related errors occur, tracking them all down and correcting them one by one becomes difficult. If this system was implemented using an object-oriented database, the problem can be ameliorated. Keeping contracts in the database as composite objects would allow straightforward references to internal objects through the top-level contract object. Also, object-oriented databases have the property that a change affecting a root object can be propagated to all of its child objects, if desired. This makes modifying the data in the database much less error-prone.

The methods CDMS employs to handle its data present a strong argument for an object-oriented design for the system. Each instance of a contract, for example, has identical components. Different contracts are merely different “instances” of the same class. The only difference among the instances is the value of their class members. For

example, each contract must have a sponsor. The sponsor name and address may vary from contract to contract, but the fields must be present.

## **4.4 Design of CDMS with OODB**

### **4.4.1 Structure**

From the highest level, the most basic structure this system needs is a GUI and its supporting database. The GUI is the front end of the system that enables the end user or operator to enter essential data while the database supports the basic structure of data storage. The design of the GUI should echo that of the database. Having both GUI and supporting database using object-oriented technology removes a lot of the object conversion found when using relational databases. The important issue here is to design the objects and object structure in a manner that supports maximum performance while minimizing the confusion often experienced with relational databases. Clarity and logical hierarchy are the most important concerns.

### **4.4.2 Primitive Design**

One option for designing the structured object database is to construct one that is very similar to a relational database. That would mean keeping all objects at the same hierarchical level. Queries performed on such a database would require repeated requests to the database, as objects along the search path were requested. Such an approach would defeat the purpose of an object-oriented database design, ignoring the natural hierarchy in the relationship between contracts and orders. The advantage to this approach is that a minimum amount of redesign needs to be done. There can be a direct one-to-one



correlation between the table structure in the relational database and the objects in the object database structure. For each table in the relational model, a corresponding object would be constructed. The data members of each of these classes would consist of the column names of each table. Converting the tables into objects is a trivial task. A figure of the structure of the OODB system is as the follows:

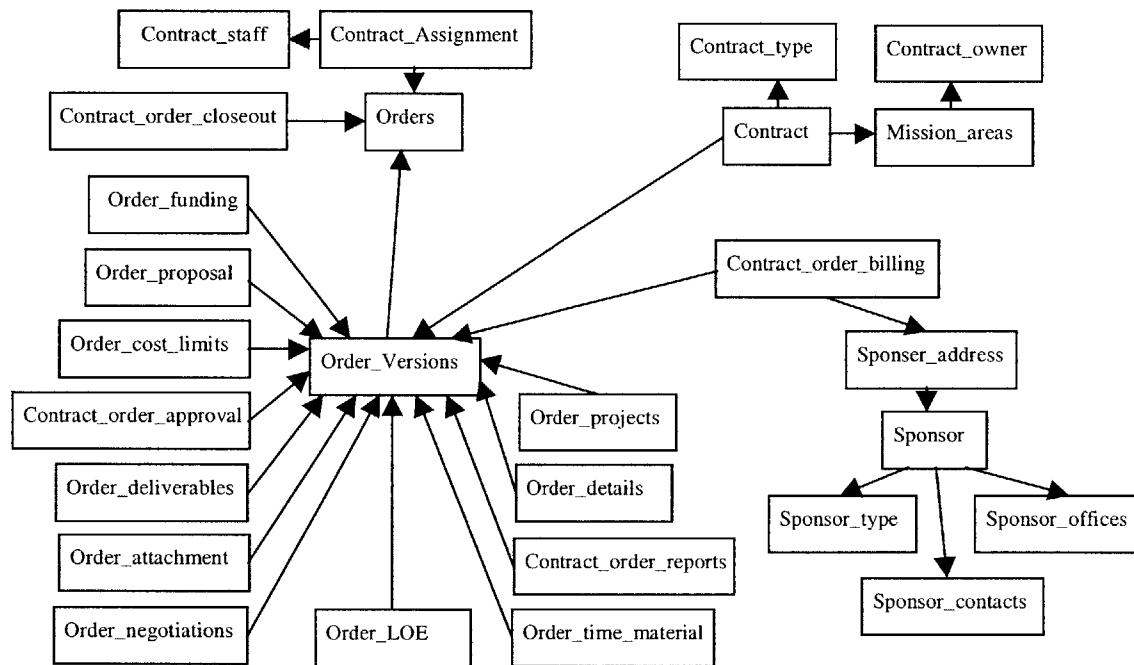


Figure 4.4: Primitive OODB Design for CDMS

Explanation of the modules:

*Contract\_Order\_Closeout:* This is the module to follow when a contract has come to an end. The valid data fields recorded include a series of dates concerning the notification of the contract closeout process, closeout notes, and closeout letter to the sponsor. This module has a many-to-one relationship with the Orders module, meaning that for each order, there can be many contract closeouts.

*Orders:* This is the module that defines an order. Important data recorded are: order type, order number, CDS number, TODS number, the status, dates that the order was initiated,

activated, and fully executed, and the closeout dates. Since orders can be inherited from a parent, each order contains a parent order id as well. Because of this inheritance property, this module actually has a many-to-one relationship with different instances of itself.

*Contract\_Assignment*: This module record the date of the contract assignment. It has a many-to-one relationship with the Order module and contract\_staff module.

*Sponsor\_types*: This module defines the sponsor type and the type status. This module has a one-to-many relationship with the Sponsor module. For every one type of sponsor, there are potentially many different sponsors.

*Sponsors*: This module defines the sponsor by recording the sponsor name, the sponsor agency name, and the sponsor status.

*Sponsor\_Offices*: This module records the sponsor office name and its status. This module has a many-to-one relationship with the Sponsor and a one-to-many relationship with the Order\_Details module

*Sponsor\_Addresses*: This records the sponsor address and has a many-to-one relationship with the Sponsor module. One sponsor can have multiple addresses.

*Sponsor\_contacts*: This module contains the specific information regarding contacts in of each sponsor. Information include name, phone number, mail stop, etc. This module has a many-to-one relationship with the Sponsors module.

*Contracts*: This module contains a variety for parameters concerning the contract. For example, the mission\_area code the fed\_gov\_subcon\_ind, and contract\_owner\_name are all tracked here.

*Contract\_types*: This module defines the different types of contract available. It has a one-to-many relationship with the contracts module meaning that each contract can belong to one and only one contract type.

*Contract\_codes*: This is the unique identifies type of contract.

*Contract\_owners*: This module indicates the name and the identity of the owner, who can be the Air Force or Department of Department.

*Mission\_areas*: This modules describes the area of concentration the contract belongs to. This module has a one-to-many relationship with contracts module. It also has a many-to-one relationship with contract\_owner module. This means that one owner can be in charge of multiple mission areas but each contract can only belong to one mission area.

*Order\_Proposals*: This module keeps track of the contract order proposal and

*Order\_versions*: This is a very important module as it contains the attributes that define an order. The attributes included are *version\_id*, *version\_type*, *version\_date*, and *parent\_version\_id*. This outlines a very important property of order, which is that it can generate offspring orders. The version number is a very important information regarding the contract as all versions of a contract must be kept in the database.

*Order\_contacts*: This contains the administrative information regarding the contact such as the address, phone number, etc. There can be multiple contacts per order. Therefore, this module has a many-to-one relationship with order-versions.

*Contract\_order\_billing*: This module lists out the billing information, such as the sponsor name and address.

*Contract\_staff*: The module defines the personnel information on the staff member assigned to the contract. Thus, this has a one-to-many relationship to *contract\_assignment* module.

*Order\_projects*: This module describes the administrative data on the project. The type of data include project manager, project department, division, etc. This has a many-to-one relationship with *order\_versions*.

*Order\_attachments*: This module indicates any additional attachments to an order version. An attachment is just a side comment on an order. An order can have multiple attachments. Thus this has a many-to-one relationship with *order\_versions* module.

*Order\_negotiations*: This module contains the data for the contract negotiations between the sponsor and MITRE. The negotiation amount, and the negotiation delta amount. This module has a many-to-one relationship with *order\_versions* module.

*Order\_time\_materials*: This module contains the information regarding the labor payments to a contract order. The time period, the labor measure, labor rate, time quantity are all recorded. This also has a many-to-one relationship with *order\_versions* module.

*Order\_LOE*: This module contains data about the LOE Labor, quantity, and line number. It has a many-to-one relationship to *order\_versions* module.

*Contract\_Order\_Reports*: This module doesn't contain the actual contract order report. Instead, it contains data such as report description and report category. This module has a many-to-one relationship with the *order\_versions* module.

*Order\_details*: This module contains the actual details of the contract. The risk flag and the reason for the risk flag are included. The sponsor id and related sponsor information are also included here. This module also includes information on cost, funding, etc.

*Errors*: This is the module that defines the database query errors that can occur during query. Typically, this can be handled by the programming exception handling service.

The disadvantage of the proposed design of the object-oriented database is that it does not take the advantage of the natural hierarchy found in the CDMS system. By maintaining the identical structure as the relational model, the system remains virtually the same. The efficiency of the system actually would decrease since object databases take longer to both store and query items. Object-oriented databases cannot perform queries with the same efficiency as relational databases because of the complexity of the data it contains. Many of the efficient search algorithms used in relational databases rely on the knowledge of the data type. Certain algorithms work better on integers while others work better on strings. However, given a generic data type such as an object, the object-oriented database system does not contain an optimized algorithm for searching and query without specific knowledge about that object. Another reason for the slower performance is that given an identical algorithm, object comparison and sorting is handled more slowly than with primitive data types. This is easily explained since an object is composed of primitive data. Thus any type of computation performed upon an object can be magnitudes slower than that of a similar operation on a primitive data type. The primitive object database design did not take advantage of the biggest asset of object oriented databases – the natural hierarchical approach to data representation. With the hierarchical representation, the programmer would have a much easier time to locate features and store/query from the appropriate location. This decreases the chance of human error and makes the system less prone to errors. Thus a better design would incorporate the natural structure of the contract system and follow the natural hierarchy found therein.

### 4.4.3 Improved Design

The initial design for the object-oriented database did not fully utilize the advantages of the paradigm. For example, the design did not utilize objects' ability to represent hierarchical structure naturally. An improved design takes the advantage of these properties of the object-oriented paradigm and builds the application on top of the data structure.

From the top level, the most important object is the contract. Thus the contract is the highest level object in this system. The most complicated contract object is the BOA, which contains both deliver orders and task orders. A task order must be classified as an individual class by itself. A delivery order is slightly different from a task order in that it can contain one or more task orders in itself. A delivery order is structurally identical as a Normal Contract. The difference between a contract and an order is the identification serial number they must keep for book keeping. Because a Normal Contract can also evolve into a delivery order, these two types of objects must be very similar. Since this object has the dual identity of being both an order and a contract, it must contain class members from both classes. For the purposes of structure, delivery orders and normal contracts should not be designed as a single class. The solution is for both delivery order and Normal Contract classes to have identical data members. The data members include data from being both a Normal Contract and an Order. For example, both a contract id and an order id must be provided within this class. But the member functions for these classes can be different in such a way that they limit the access to the member data. Similar to the previous example, in the Normal Contract class, there would be a function `get_contract_id()` that obtains the contract id number for that contract. But there

would not be a `get_order_id()` method to obtain its order id. But in the Task Order class, the opposite is true. The `get_order_id()` method would be available while `get_contract_id()` would not be.

The contract object is a composite object, which means that its data members contain other objects. There are several kinds of information in this case that can be included as individual objects. Information such as address and contact\_info occurs repeatedly in our database. Several modules require this type of information, i.e. sponsor\_address and sponsor\_contact. Therefore, defining an object class for these generic data types would simplify the attribute fields for more complex composite objects. Other modules such as the contract\_staff belong to this category.

Another type of information that can be constructed as an object is a significantly larger object like sponsors. The sponsor object occurs in every order and the information structure is identical. In fact, most of the fields inside order\_version would be individual objects unless they are extremely simple data types.

Table 4.2 outlines the defined objects in the new object database design.

**Table 4.2 Objects and their data members in improved OODB design for CDMS.**

<u>Significant Objects</u>	<u>Data Members</u>
Contract	Order_versions, Contract_type, Contract_owner, mission_areas
Order_versions	Order_funding, order_cost_limits, order_proposal, contract_order_approval, order_deliverables, order_attachment, order_negotiations, order_LOE, order_time_material, contract_order_reports, order_details, order_projects, contract_billing
Sponsor	Sponsor_type, sponsor_contacts, sponsor_offices

With the above object definitions, the structure of the underlying database for CDMS has been greatly simplified, consisting of only several prominent objects. The detailed data are hidden inside the larger objects. From a high level, the only significant object is the contract itself. Access to any information related to the contract necessarily must proceed through the top-level objects.

#### **4.5 Effects of OODB on CDMS**

With the reorganization of the system, the structure of the CDMS is much simpler than before. The hierarchical structure of the back-end system is quite clear. From the system structure, a programmer would have a good understanding of the structure of the data. This system would also make error tracking a lot simpler. Since errors would be propagated upwards through objects, the exact location of any error can be determined quickly.

With modules organized into objects, there are number of operations defined on the objects. In a sense, the object can 'behave' in a pre-defined way. This makes contract tracking less awkward conceptually than the relational database implementation.

One potential disadvantage associated with a hierarchical structure is described in following scenario. Suppose the application programmer wants to find specific data in a specific object. He cannot directly refer to this object from the top level. He must traverse the hierarchy of objects and references to finally arrive at the desired attribute. This can take quite a bit of effort. In a relational database, as long as the attribute name and a unique identifier is provided, a specific attribute can be queried immediately. The

difference in the speed and the ease of this particular query demonstrates the slight disadvantage of a hierarchical structure.

#### **4.6 Summary**

The CDMS is a complicated system with many components and complex behavior. Each contract has many components and components may exhibit dynamic behaviors. Implementing such system under a relational database environment makes this system highly error prone. The lack of hierarchy in relational databases exacerbates this problem.

The object-oriented design for CDMS utilizes the properties of hierarchical structure possible with objects. In the new database system, there are only several object types that need tracking. They include contract, order, and sponsors. The new design incorporates many previously independent modules into these objects, becoming data members of these top-level objects. Behaviors such as conversion from one type of contract to another, (i.e. from a Normal Contract to an Order under a BOA) can also be accomplished with explicit object type conversions. The operation is known as type casting, and is often used in object-oriented programming.

In short, an object-oriented database implementation of this system could provide many benefits by organizing data in a more manageable form. A possible disadvantage is the slower query performance for a specific data attribute, which arises from the overhead introduced by molding the data into objects and referencing through several objects. However, it is clear that the advantages in improving the functionality and manageability of CDMS outweigh the slight disadvantages.



## Chapter 5

### Conclusions

Relational databases and object-oriented databases are two different systems of storing and retrieving data. At first glance, they are similar in that they both store and query data for front-end software applications, especially in information technology related software.

However, these two systems are fundamentally different. Relational databases only store data of limited types. These data are usually of the type numeral, char, date, or other system-dependant variables. A relational database presents a relation as a table with columns as attributes. An object oriented database stores objects. Each object contains data members that form its attributes. In relational databases, queries are formed using SQL and keys to identify the unique elements to be selected. In object-oriented databases, each object has an object-identification and referring to a related object is easily performed by following a reference attribute to another, already identified object.

Database application design focuses on different levels of designs, depending on which database technology was chosen. For relational database applications, the responsibility of the database design rests with the database administrator because the application programmer does not need to understand the organizational structure of data in the

database. For object-oriented database applications, however, the programmer needs to understand the data structure of the data objects. He must understand the attributes of each object to correctly implement the functionality of the application. Finally, there is the issue of compatibility with popular programming languages. Relational technology is more mature and standards have been put in place to guide the interface between the database and the application programming languages. For example, the Open Database Connectivity (ODBC) standard will allow access to different relational databases simultaneously. This currently cannot be accomplished with object-oriented databases. Object-oriented databases offer different advantages. The fact that its stored data is in the form of objects already gives it so much more power than relational databases. The object-oriented languages can use object-oriented databases more readily than relational databases because storing and querying complex objects can be more easily done.

Having analyzed the advantages of each system, two individual cases are studied. The first case study covers DataView, Inc's DVI Knowledge Suite. The Knowledge Suite is a knowledge visualization tool that specifically targets the less technical end users. The software combines data, business rules, and graphics to present the data in an easily discernable manner. The Knowledge Suite is conceptually novel but functionally lacking. One of the insufficiencies arises from the fact that it only accepts data from relational databases as input. The functionality of the database has also been compromised by the fact that it only stores application specific data in relational database. If application related objects could be dynamically stored and retrieved, the functionality of the system would be improved substantially.

The design for DVI Knowledge Suite to incorporate object-oriented databases mainly focuses on redesigning the features in DVI Designer. There are three modules that would be designed as objects. The first one is the input data. When the DVI Designer accepts objects as input data, its versatility would be greatly increased. The second module is the graphics module. The graphics module would include many attributes such as the target region size, shape, color, etc. The third object module is the business rule module. This module is the data acquisition module. It contains the OQL language query that retrieves the data out of the database and it also contains the rules that the end-user defines to dictate the behavior of the region of the graphics with regard to the query results.

DVI Knowledge Suite would definitely become a more versatile application by allowing objects to be stored and queried more directly to the database and thus accommodating complex behavior. Another advantage is that by designing the graphics section as one single object, one can upgrade and modify the graphics object better without affecting the rest of the Suite. This fits well with object-oriented paradigm. Possible disadvantage comes from the fact that the creation of objects may create overhead. If the queried data were simple data, using objects would slow down the speed of the application.

The second case study involves a contract management system called the Contract Data Management System (CDMS). CDMS is a very complicated system that involves over 30 tables and modules. Data in this system exhibit interesting behaviors. For example, contracts will have child contracts. Sometimes an independent contract such as a Normal Contract would evolve into a task order of a different contract (BOA). Using a

relational database to keep track of the significant changes of properties in the contracts is a difficult and awkward task. CDMS has problems with poor error handling problems due to complex nature of the system and its behaviors. Object-oriented programming has error handling ability that can propagate the error from the attribute level all the way up to the contract level. This would make errors easier to identify and correct as well.

The design of CDMS to use an object-oriented database involves grouping the more than 30 original modules into several significant objects and defining behaviors for each of them. The main modules that are defined as objects include contract, orders, and sponsor. The reason for organizing objects in this manner is that this partitioning would create the least number of object types while avoiding redundant attribute data in each object. Since the classification of the object virtually follows the functional requirements of the system as given in the documentation, the objects' behaviors should be discerned intuitively.

Though there are numerous advantages to implementing the underlying system as object-oriented system, there could be potential disadvantages. One of the foreseeable disadvantages is the speed of query when querying a simply typed attribute. This can be especially true when the attribute is several references away from a 'root' object.

Querying a table would be much faster.

Relational databases and object-oriented databases each have their own advantages. Relational databases seem to be more appropriate for those applications that require the storing and querying of simple data types such as numerals and character strings only. For applications that do not require storing data with complex structure or behavior, relational databases are a great choice. For applications that involve complex

data types, the choice is obviously object-oriented database. Object databases allow for easy access through composite data.

Object-oriented databases are still a developing technology, with many issues to be resolved. For example, more efficient methods of querying the database need to be developed. Other specific issues, including database security, object persistence, and efficient storage, would also benefit from further investigation. These issues are instrumental in pushing the object-oriented technology forward.

## REFERENCES

- [1] Alter, Steven L., Decision Support Systems: Current Practice and Continuing Challenges. Addison-Wesley Publishing Company. . 1980.
- [2] Barry, Doug. ODMG 2.0: A Standard for Object Storage. <http://www.odmg.org/> 1998
- [3] Bidgoli, Hossein. Decision Support Systems: Principles and Practice. West Publishing Company, New York. 1996.
- [4] Fowler, Martin. "Techniques for Object Oriented Analysis and Design". <http://www2.awl.com/cseng/titles/0-201-89542-0/techniques/> 1997
- [5] Kim, Woo. Introduction to Object-Oriented Databases. The MIT Press. Cambridge, MA. 1990.
- [6] Liberty, Jesse. Teach Yourself C++ Programming in 21 Days. SAMS Publishing. Indianapolis, IN. 1994
- [7] Luers, Tom. Essential Oracle7. SAMS Publishing. Indianapolis, IN. 1995.
- [8] Maciaszek, Leszek A. "Relational versus Object Databases - Contention or Coexistence?" <http://www.comp.mq.edu.au/courses/comp866/oovsrel.html> . Sydney. 1997
- [9] McClanahan, David. Oracle Developer's Guide. Oracle Press. New York. 1996.
- [10] McFarland, Gregory. Rudmik, Andres. and Lange, David. "Object-Oriented Database Management Systems Revisited: An Updated DACS State-of-the-Art Report". Prepared for Air Force Research Laboratory. Contract Number SP0700-98-4000. 1999.
- [11] Nielsen, Jakob. "Ten Usability Heuristics". [http://www.useit.com/papers/heuristic/heuristic\\_list.html](http://www.useit.com/papers/heuristic/heuristic_list.html)
- [12] Silver, Mark S. Systems that Support Decision Makers: Description and Analysis. John Wiley & Sons. New York 1991.
- [13] Turban, Efraim. Decision Support and Expert Systems: Managerial Perspectives. Macmillan Publishing Company. New York. 1988.
- [14] DVI Knowledge Suite™ Release 1.5 User Manual. Data View, Inc. ©1999
- [15] "Contract Data Information System (CDIS) Documentation". MITRE Corporation. McLean, VA. 1996.

[16] “Contract Data Management System (CDMS) Documentation”. MITRE Corporation. Bedford, MA. 1998.

[17] “Object Data Management Group: The Standard for Storing Objects”. <http://www.odmg.org/> Object Data Management Group. 1999.

[18] MITRE Corporation, <http://www.mitre.org/>

[19] “DVI Knowledge Suite draws on expert analysis”  
<http://www.infoworld.com/cgi-bin/displayArchive.pl?/99/23/e04-23.60.htm> . InforWorld. June 7, 1999.

[20] “Data View DVI Knowledge Suite 1.5”.  
<http://www.oracle.com/oramag/oracle/99-May/39val.html>. Oracle Magazine. May 1999.

[21] “Real-Time Data Access Via Thin Clients”  
<http://www.geoplace.com/bg/1999/0699/699prod.asp> GeoPlace.com

[22] “Introducing: Java-based DVI Knowledge Suite™”  
<http://www.dataviewinc.com/newsitem11.htm> . Technology Times. February 1999.