Design and Prototype of an Object-Relational Database for Medical Images

by

Ngon D. Dao

B.S. in Mechanical Engineering
University of Texas at Austin, 1996

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of

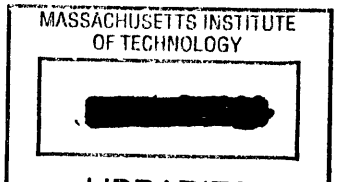
Master of Science in Mechanical Engineering
at the
Massachusetts Institute of Technology

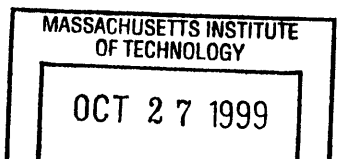
June 1998

Signature of Author: _____
Department of Mechanical Engineering
May 08, 1998

Certified by: _____
C. Forbes Dewey, Jr.
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: _____
Anthony Patera
Acting Graduate Officer





Design and Prototype of an Object-Relational Database for Medical Images

by

Ngon D. Dao

Submitted to the Department of Mechanical Engineering
on May 08, 1998 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Mechanical Engineering

ABSTRACT

Current electronic patient medical records are based on relational databases that have been very successful in managing text-based information. Recently, however, complex data types, such as digitized images, are becoming increasingly important in healthcare. Unfortunately, existing relational databases are not able to manage these complex data types in a satisfactory manner. The goal of this project is to study the feasibility of developing a high-performance image archive that will handle both text and complex data types. In order to achieve this goal, this project designed and prototyped a medical imaging archive. The prototype consists of an Informix Universal Server database engine, an object-oriented database schema, and an application-programming interface (API). The schema and API are implemented in a software module called a DataBlade. This DataBlade extends a generic Informix database with the ability to manage medical information objects as defined in the DICOM (Digital Imaging and Communications in Medicine) standard. The schema incorporates an extensible inheritance hierarchy, while the API includes encapsulation of routines and data structures built specifically for DICOM information. The prototype demonstrates how developers can be shielded from the complexities of the DICOM standard, yet still be able to store and query the complete DICOM information model.

Thesis Supervisor: C. Forbes Dewey, Jr.
Title: Professor of Mechanical Engineering

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I love this place! It is 1998, and I can sense the convergence happening around me, right now, right here at MIT. Engineering, computer science, medicine, and biology are coming together, and this place feels like it is in the middle of it all. Professor C. Forbes Dewey Jr. has given me the fantastic opportunity to not only witness this convergence first-hand but to also be a part of it. For this, I am greatly indebted to him.

I dedicate this thesis to my Mother and Father and to my three brothers. My parents taught me perseverance and tenacity through their life-long examples, while my brothers were the ones to show me the meaning of competition.

This thesis is partly supported by a National Science Foundation Graduate Fellowship.

Chapter 1 Background

Within the last ten years, healthcare providers have increasingly adopted electronic patient records as the method of choice in managing patient information. This industry-wide movement brought about the development of computer-based patient records (CPR) based on the relational data model. Although these CPRs have proven quite capable of managing text-based patient information, they have had limited success in incorporating digital images into medical records [1]. Specifically, existing relational CPRs cannot satisfactorily store and query digital medical images which are widely created and communicated in accordance to the DICOM 3.0* standard (Digital Imaging and Communications in Medicine) [2].

The remaining sections of this chapter present background information on the DICOM standard. This Chapter will then conclude with a brief discussion of prior work in managing DICOM information.

1.1. The DICOM Medical Imaging Standard

In 1983, the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) formed a joint committee to develop a vendor neutral standard for communicating digital medical images. This committee developed the first two versions of the DICOM standard (ACR-NEMA 300-1985 and ACR-NEMA 300-1988). These versions failed to garner widespread acceptance because they lacked network support and an explicit information model for radiological information [3, 4]. Thereafter, it was decided that version ACR-NEMA 300-1988 would be completely redesigned to accommodate network support and a clearly-defined object-oriented information model [5]. This new design was ratified in whole in 1993 and is known as DICOM 3.0. DICOM 3.0 is currently the most widely used protocol for communicating diagnostic medical images in digital format.

The remaining parts of this section is not intended to be a tutorial on DICOM, but rather, they serve to explain characteristics of the standard that are major drivers in

* Hereafter referred to simply as DICOM.

designing the image archive. These characteristics are that the standard is object oriented, it defines a highly nested information model, and it incorporates large data types.

1.1.1. DICOM: Object-Oriented

The DICOM standard is object-oriented because it defines an information model based on encapsulation and inheritance [6]. Specifically, the model defines objects, such as patients and images, that have attributes and services arranged in both an entity-relationship hierarchy and an inheritance hierarchy. Both hierarchies are discussed in more detail later (Sections 1.1.2 and 2.1.4); this section describes DICOM objects themselves. There are two categories of DICOM information object definitions (IOD): normalized object classes (NIOD) and composite object classes (CIOD)*. NIODs contain only those attributes that inherently describe their corresponding real-world entity. CIODs, however, contain inherent attributes as well as other non-inherent but related attributes. For example, DICOM defines a patient NIOD that only contains attributes that inherently describe a patient, such as name and date of birth. On the other hand, the standard also defines a computed radiography CIOD that contains attributes describing characteristics inherent to an X-ray, such as pixel data, as well as related attributes for interpretation, such as the patient-owner of the X-ray. Figure 1-1 illustrates the general structure of normalized and composite information objects. In general, a CIOD consists of information entities (IE), and two of these entities, the Patient and Study entities, are subsets of their NIOD counterparts. Table 1-1 on page 18 lists the most commonly used NIODs and CIODs defined in DICOM.

* Normalized and composite object classes are more commonly referred to as normalized information object definitions (NIODs) and composite information object definitions (CIODs), respectively.

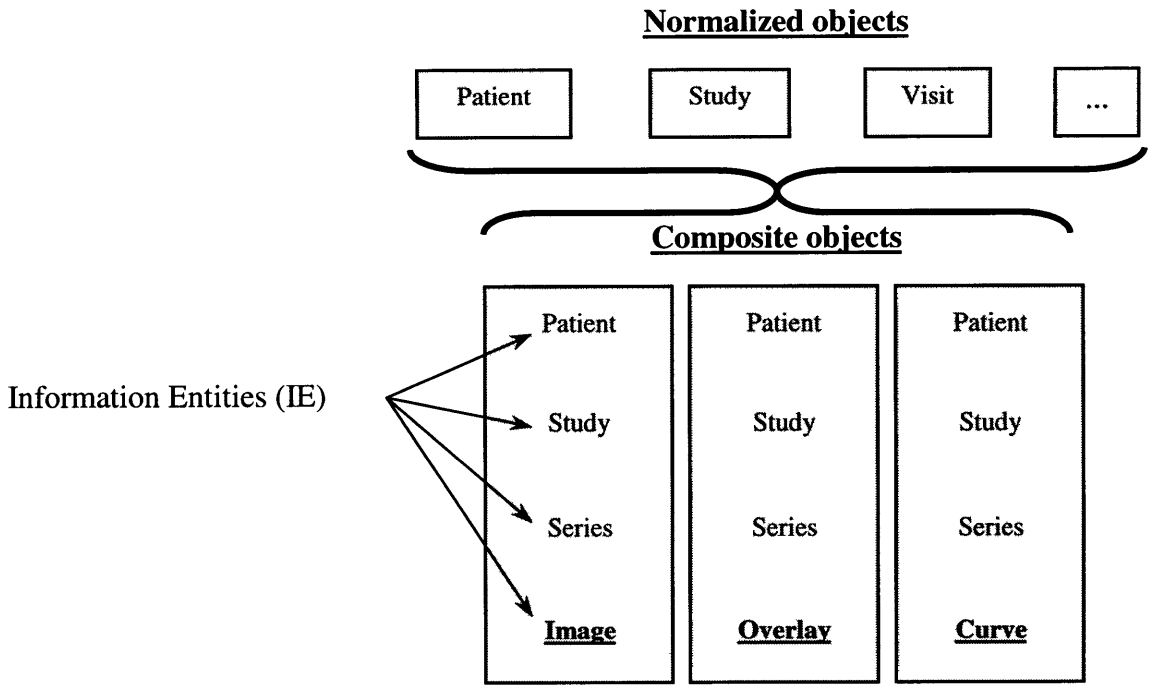


Figure 1-1 Relationship between normalized objects (NIOD), composite objects (CIOD), and information entities (IE) within the DICOM standard. CIODs are made of IEs. Some IEs, such as the Patient IE and the Study IE, are subsets of their NIOD counterparts.

There are several important aspects concerning the object-orientedness of the DICOM information model. First, the model is based on semantically natural information objects (NIODs and CIODs) and information entities. That is, it is intuitive for composite objects to be comprised of a Patient IE who may own one or more Study IEs, for which there may be recursively one or more Series IEs and Image IEs. Second, although composite object definitions are semantically natural, they are not normalized. A storage and management system based on non-normalized objects will encounter data redundancies that eventually will result in integrity problems. So while DICOM, on one hand, defines a highly natural information model that should be used for the sake of ease-of-use, it is intimately based on a very non-normalized object model and should be avoided when implemented in an archiving system. A major effort of this project involves developing an information model that is fully normalized but is as natural as the DICOM model.

Table 1-1. Normalized and composite object classes within DICOM

Object Class	Class Type
Patient	Normalized
Visit	Normalized
Study	Normalized
Interpretation	Normalized
Results	Normalized
Computed Radiography Image	Composite
Computed Tomography Image	Composite
Magnetic Resonance Image	Composite
Nuclear Medicine Image	Composite
Ultrasound/Ultrasound Multi-frame Image	Composite
Secondary Capture	Composite
Standalone Overlay	Composite
Standalone Curve	Composite
Waveform	Composite
Visible Light Image	Composite

1.1.2. DICOM: Nested Information Model

Recall from the previous section how a Patient IE can have multiple Study IEs where each Study IE can have multiple Series IEs, etc. This is an example of complex data where information objects are related to one another by many one-to-many relationships. Figure 1-2 is a graph illustrating how complex data defined in DICOM can result in a highly nested information model. This nested structure poses a tremendous information management problem because queries based on multiple objects can be computationally costly as they require the reconstruction of nested relationships.

1.1.3. DICOM: Large Data Types

The final important aspect of DICOM is that it defines objects with very large attributes. For example, a typical diagnostic image object can have pixel data larger than 20 Mbytes. Incorporating very large data streams within object definitions poses several potential problems. First, query processing requires more complex routines than those developed for shorter data types, and second, transactions on large data types tend to impose blocking calls that slow or interrupt other processes. As will be presented in the next section, many developers have circumvented these requirements by developing

DICOM management systems that do not incorporate the entire DICOM information model.

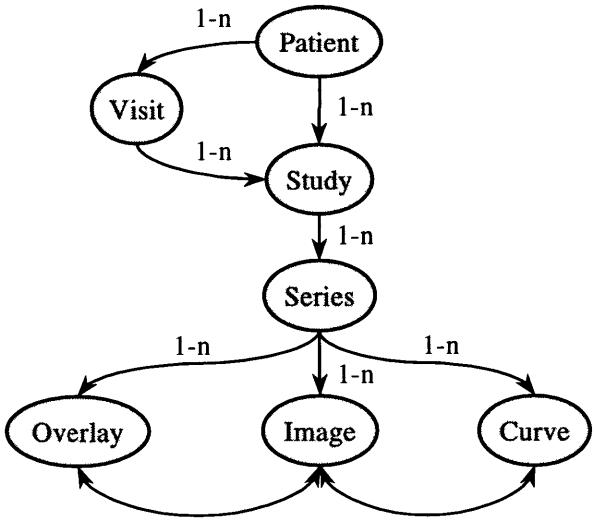


Figure 1-2 A directed-acyclic graph representing a subset of the DICOM information model for composite objects.

1.2. Prior Work

Since the late 1980's, manufacturers of diagnostic imaging equipment have been moving toward digital format in lieu of film-based outputs. This industry wide movement has prompted much effort to develop information management systems to organize and manage digital medical images. H. K. Huang was instrumental in spearheading this effort between 1982 to 1992 by developing several picture archiving and communication systems (PACS) based on either proprietary communication standards or the earliest versions of DICOM [7, 8, 9]. In 1992, Ratib et al at the University Hospital of Geneva developed a fully functional medical image archive. This system pioneered a highly distributed architecture, but its proprietary Papyrus image format has limited its acceptance [10]. Concurrent to but independent of this development, Humphrey and DoVan developed one of the first image management system designed specifically for DICOM information [11]. They used a relational data model, and as a result, were only able to represent a subset of the DICOM information

model within their database*. To compensate, they used pointers within the database to reference DICOM files residing outside of the database's management domain for image data as well as other large unstructured data. To this day, their work serves as the model for many commercial PACS, including those from AGFA, Siemens Medical Systems, and GE Medical Systems [12, 13, 14]. In 1992, H.K. Huang and S. Wong [15] extended their previous work to include support for DICOM and an object-relational mapper. This mapper serves as the object-oriented interface for their underlying relational database.

The goal of this thesis is to design and prototype a database to support the complete DICOM information model. The next chapter presents the design goals for this project.

* For a discussion of the limitations of the relational data model in supporting the complete DICOM information model, see Section 3.2.

Chapter 2 Design Goals and Specifications

This project studies the feasibility of a commercial grade image archiving system for managing the DICOM information model. In this study, a design is chosen and a prototype is developed that can manage the entire DICOM information model while shielding its users and administrators from the complexities of the standard. The design process consists of selecting a data model as well as a method for how information is organized within the model. To this end, design specifications were developed to assist in evaluating design alternatives. The following sections explain design specifications for both the data model and its internal representation (schema). In addition to these, however, another component is required to isolate users and developers from the complexities of the schema and standard. This project uses an application-programming interface for this purpose, and its design specifications are presented after those of the data model and schema.

2.1. Data Model Design Specifications

A data model defines how information is presented to the users of a database. Since the invention of the modern computerized database, many data models have been developed, ranging from the older hierarchical and networked data models, to the currently prevalent relational data model, to the more recent object-oriented and object-relational data models. In order to evaluate these existing data models for the image archive, design specifications were developed. They are explained below.

2.1.1. Use Relational Technologies

The selected data model must leverage existing relational database technologies, such as transaction management, security, SQL, and concurrency control. These technologies are well developed for the relational database domain and currently serve as industry-wide benchmarks for new database technologies. Since the prototype developed for this project is intended to serve as a demonstration of a commercial grade system, this requirement is essential to the functionality of the archive.

2.1.2. Support User-defined Types and Methods

The selected data model must natively support user-defined data types and access methods. DICOM defines many information objects with large unstructured attributes that can be indexed, queried, stored, and retrieve in many ways. For example, the pixel data of a diagnostic image can be sequentially read or read by random access according to a variety of optimization algorithms. To accommodate the many ways a DICOM information object can be represented and analyzed, the selected data model must support user-defined data types and access methods. This facility allows developers to specify the internal representation of data as well as support routines for accessing it in a manner optimized for DICOM objects. This requirement is essential to the functionality of the archive.

2.1.3. Support Complex Data

The selected data model must support complex data in a natural manner. The DICOM information model specifies a directed acyclic-graph with many one-to-many references (see Figure 1-2 on page 19). Since a goal of this project is representing the DICOM information model within the image archive, the selected data model must have facilities for representing relationships between highly nested objects. Although nested objects can be represented, to one degree or another, in all existing data models, their representations can be awkward or natural depending on the data model used. A detail discussion of this is presented in Chapter 3.

2.1.4. Support Class Inheritance

The selected data model should support the notion of inheritance. The term "inheritance" is used here in the object-oriented context: a class can inherit attributes and methods from another class. Although not mentioned in Section 1.1, DICOM IODs are highly modular, and related objects extensively share attributes with one another. For example, the computed radiography CIOD shares over 40 attributes with the computed tomography CIOD, the ultrasound CIOD, and the magnetic resonance CIOD. Such sharing of attributes makes it natural for these CIOD objects to be related to one another in an inheritance hierarchy. The use of an inheritance-based schema promotes the re-use

of class definitions for a more modular and compound schema. These two mechanisms help shield developers from the complexity of the DIOCM standard. This requirement is not essential for the functionality of the image archive, but it is a preferred attribute.

This section has outlined the major requirements for selecting a data model for the DICOM image archive. Chapter 3 presents a thorough evaluation of existing data models and selects one for this project.

2.2. Database Schema Design Specifications

Database design typically involves 1) identifying data objects to be managed, 2) designing a schema, and 3) optimizing the schema by resolving and normalizing it. This project does not involve the identification of objects because they have been specified by the DICOM standard. This section explains the design specification for the database schema. Chapter 4 explains the schema design process after evaluating alternative designs based on the specifications presented here.

A database schema defines the logical relationship between data items and their integrity constraints. Modern database design uses the entity-relationship (E-R) approach developed by E. R. Bachman. In this approach, an E-R diagram is developed to model both the information entities of interest and their relationships. This project uses the DICOM information model (which is an E-R diagram itself) as the starting point but makes major modifications to it. These modifications are necessary because the entities defined in the DICOM model are not normalized, the DICOM model does not explicitly use object inheritance, and the DICOM model is not optimized for querying information entities.

Table 2-1 lists design requirements that serve as guidelines for the image archive schema. Most requirements are self-explanatory, but those that need further elaboration are noted with an asterisk and are discussed below.

Table 2-1 Schema design specifications

General Requirements - Must be independent of the application level interface - Support all DICOM information model entities and data types - Entity attributes must be consistent with those used in the DICOM standard **Modeling Requirements** - *Be in Fifth Normal Form (5NF) - *Be self-contained (lossless decomposition and nomenclature conservation) - *Avoid non-homogeneous class definitions - *Use type inheritance - Minimize the number of entities - *Support expanded query scopes

2.2.1. Minimize Data Redundancy

One goal of modern database design is the minimization of data redundancy within the schema. Data redundancy is the storage of the same information more than once or when information that can be derived is stored. Redundancy should be avoided because it promotes inefficient storage space utilization, complicates updates and data modifications, and leads to data inconsistency [16]. Data redundancy can occur as a result of many situations, ranging from simply having class definitions with semantically identical attributes to more subtle cases involving multi-values dependencies. To eliminate these redundancies, conditions have been defined that are the basis of the five normal forms used in relational database design: first normal form (1NF), second normal form (2NF), Boyce-Codd normal form (BCNF), fourth normal form (4NF), and fifth normal form (5NF). The explicit conditions defined by these normal forms will not be presented here because they can be readily found in the literature [16]. Although these normal forms are most commonly presented in the relational context, they can be generalized for use with object-oriented schema design. All class definitions used for this project should be in 5NF to avoid running into data integrity problems.

2.2.2. Be Self-contained

The image archive must provide a reasonable interface to communicate with other DICOM compliant devices and/or access applications. This interface is dictated by the fact that when information is communicated between DICOM compliant devices, the data

are typically sent and received as complete NIODs and CIODs. Recall from Section 1.1.1, however, that the image archive cannot store CIODs as they are defined within the standard because CIODs are non-normalized objects. Instead, the image archive can only store normalized decompositions of CIODs, but these decompositions must be lossless decompositions so that the archive can reconstruct the original objects at a later time. Hawryszkiewycz [17] gives the necessary and sufficient condition for lossless decomposition as follows

> *If a class definition, Class_1(X,Y,Z), is defined to contain attributes X, Y, and Z, its decomposition into Class_1A(X,Y) and Class_1B(X,Z) is lossless if Y is functionally dependent on X or Z is functionally dependent on X.*

In addition to this requirement, the image archive must provide facilities to map between DICOM specific nomenclature and their internal database representation. For example, most commercial database environments limit the string length of attribute names to eighteen characters. DICOM, however, defines many attribute names, such as the *Patient's Mother's Birth Name* attribute, that exceed this limit. For the image archive to be able to handle such attributes, it must provide a mechanism to convert attributes between a shorter internal representation and the longer DICOM representation. Another reason why mapping of attribute names is important is because all DICOM attributes are assigned unique tags, and it is these tags that are more often used to identify attributes. Mapping between an internal representation, tags, and attribute names is called nomenclature conservation. Satisfying both lossless decomposition and nomenclature conservation makes the image archive self-contained because it can reconstruct fully compliant DICOM CIODs from self-contained internal representations.

2.2.3. Avoid Non-homogeneous Class Definitions

Class nonhomogeneity occurs when a class contains one or more attributes for which its object instances never or rarely possess values [18]. Having nonhomogeneous classes can result in inefficient memory usage and can force external applications to make provisions for handling null values. The latter implication violates one of the fundamental goals in modern schema design, application level independence. DICOM CIODs, unfortunately, are highly nonhomogeneous. Two example of DICOM class

nonhomogeneities are in the definitions of the nuclear medicine and ultrasound CIODs containing mutually exclusive information entities (see section A.5.3.1 in Part 3 of the standard) and in the definitions of mutually exclusive cine and non-cine modules for multi-frame and non-multi-frame objects.

2.2.4. Use Type Inheritance (Class Inheritance)

Type inheritance is the ability of classes to inherit attributes and methods from other classes. When applied to schema design, type inheritance allows for a very modular schema that promotes re-use of both class definitions and methods and minimizes errors when defining new classes. There are two aspects to the DICOM information model that makes it very amenable to using type inheritance. First, recall from Section 1.1.1 that the DICOM information model is object-oriented, and as a result, associates methods with objects. Second, the model contains approximately 600 unique attributes, and a typical image information entity contains approximately 70 or so of these attributes. To provide developers with an easy mechanism for extending the schema without having to recreate existing methods or to know about all DICOM attributes, type inheritance must be an integral component of the schema.

2.2.5. Support Expanded Query Scope

If type inheritance is supported within a schema, the notion of expanded query scope can be realized. Expanded query scope refers to the ability of querying all sub-classes of a class inheritance hierarchy using only one query. That is, a query against a parent class has an expanded query scope if its scope is the parent class and all subclasses of the parent class. Expanded query scope can be extremely powerful if the underlying schema is highly modular.

2.3. Application Programming Interface Design Specifications

One major requirement of this project is to develop an image archive that allows end-users to manage DICOM information without having to know about the details of the standard. This isolation can be achieved through an application-programming interface (API). For this project, the API must satisfy the following functional requirements:

provide DICOM views, support DICOM input/output services, and facilitate schema extensions. Each of these requirements is elaborated below.

2.3.1. Provide DICOM Views (presenting an object interface)

Recall from Sections 2.2.1 and 2.2.2 that the schema only stores normalized decompositions of CIODs instead of CIODs themselves. Although breaking CIODs into normal components is a requirement of good database design, there are several disadvantages to doing so. First, users who are familiar with the DICOM object model may want to construct queries based on the CIOD model instead of its normalized version. This is because applications written in the C programming language can use linked-lists and data structures to construct and process queries more easily if they were directed against the CIOD model. Second, normal decompositions of CIODs introduce an abstraction layer that is not defined within the standard, and therefore, can be perceived by many as a proprietary approach that adds more complexity to an already impenetrable standard. Third and probably most important, all queries, no matter how simple, for aggregate information regarding a CIOD requires the reconstruction of the CIOD from its normalized components. Reconstructions can be simple or they can be very computationally costly; in which case, they can render the image archive useless in a large-scale environment.

A solution to all three of the above problems is to use DICOM views. DICOM views are analogous to views in a relational database or to cached data in fast I/O applications. For this project, there will be functions defined in the API that construct non-normalized objects having the same class-composition hierarchies as those defined in the standard for CIODs*. These DICOM views are linked to their underlying normal representations so that commands issued against a view are translated into commands on its underlying data. Figure 2-1 illustrates the DICOM view concept. By using DICOM views, users who prefer the DICOM object model are presented with that representation, while those who prefer a normalized version are presented with the actual schema. These views also eliminate the need for "on-the-fly" reconstruction of CIODs to answer aggregate queries.

---

* DICOM class-composition hierarchies for CIODs are defined in Annex A of Part 3 of the standard.

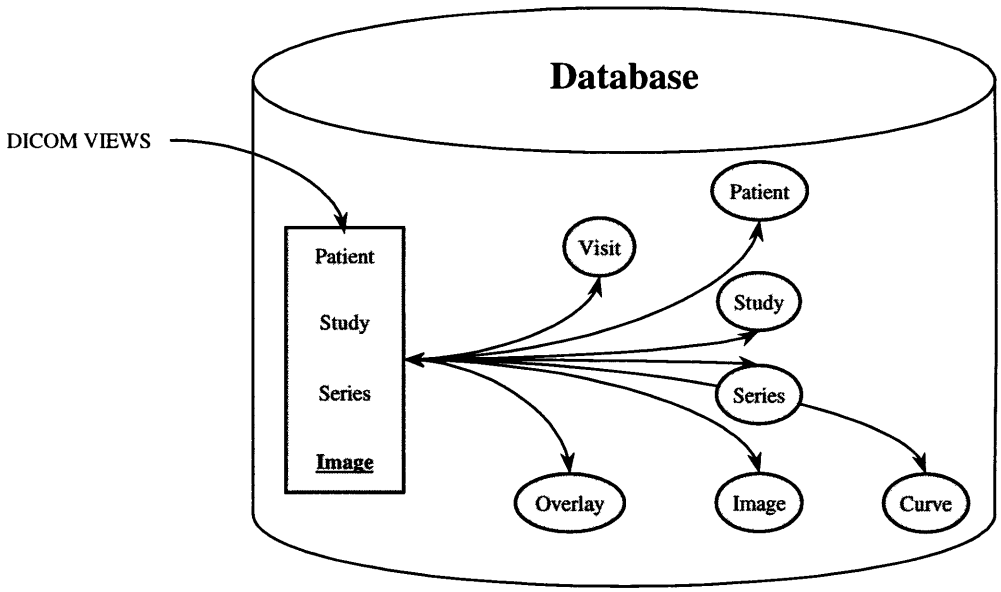


Figure 2-1 DICOM views are constructed from normalized object representations. Commands issued views operate on their underlying normalized structures.

2.3.2. Support DICOM Input/Output Services

The image archive must support queries requesting individual attribute values and those requesting entire NIODs or CIODs. Most external applications querying the image archive will use SQL, but there are two possible ways for them to receive query results. The first way is where an application requests attributes of objects within the archive and receives returned values pre-parsed. For example, if the result set of a query involves more than one value, the SQL interface of the image archive presents the result set to the application one result at a time so that the application can place each result into separate memory buffers. In the alternative way, the external application does not ask for an attribute of an object but for an entire NIOD or CIOD whose attributes satisfy some predetermined condition. An application that issues such a query still receives data through the archive's SQL interface, but the returned data will be presented as one large data stream. That is, all the attributes and values of a returned NIOD or CIOD is treated as one unit bundled into one data stream, and the access application is required to place the entire stream into one memory buffer. The format of the data stream will comply with the DICOM format for data sets as defined in Section 7 of Part 5 of the standard.

Figure 2-2 illustrates both mechanisms by which an external application can receive data from the image archive.

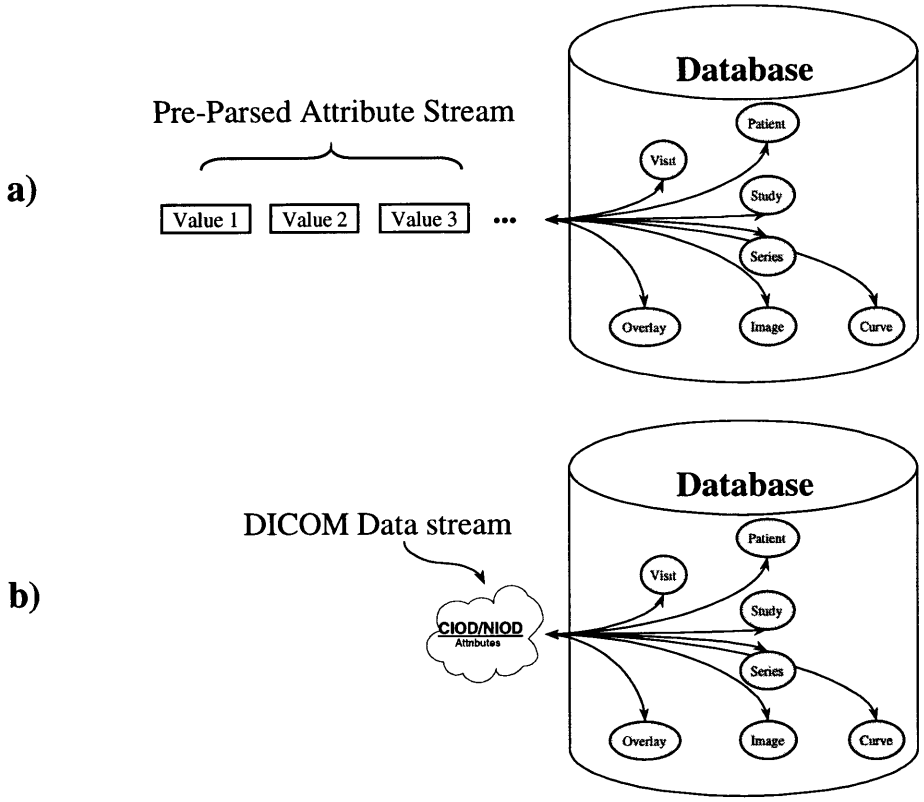


Figure 2-2 Two methods for DICOM input/output services: a) First method where individual attribute values are returned to access applications b) Second method where complete NIODs or CIODs are sent to access applications as one data stream.

2.3.3. Facilitate Schema Extensions

A schema will usually change throughout the life of its database to accommodate the changing needs of its users. In addition to changes associated with natural schema growth, the schema developed for this project must additionally accommodate two types of changes owing to the fact that it stores DICOM data. First, the DICOM standard allows for the specialization of DICOM defined IODs by the addition of private attributes. These extensions are quite common in practice as the standard commonly defines entities with a sparse set of attributes. Second, the DICOM standard is considered a large work-in-progress because ACR/NEMA is continuously augmenting it with new IODs, such as those for waveforms and visible light images. As new IODs are

ratified, users may want to incorporate them into their existing schema. Merely allowing users to modify the schema is not enough because, in reality, expecting users to understand the DICOM information model or how it is represented in the schema is unreasonable. Therefore, the image archive must provide facilities where users can modify the schema at a level separated from the low-level details present in the standard and in the schema. For example, consider the case where a user wants to add the attributes *image_attribute_1* and *image_attribute_2* to the computed radiography image CIOD at the level of the image IE (see Figure 1-1 on page 17). Without a mechanism facilitating this extension, the user would need to define a completely new entity within the database that includes the fifty-three attributes DICOM defines for the CR image entity as well as the two new attributes. This exercise will undoubtedly require the user to familiarize himself/herself with the DICOM information model as well as how CIODs are normalized in the schema.

This project circumvents the above problem by providing an API service that removes users from the low-level schema interface. The API service creates new objects by extending existing classes with subclasses (see Section 5.4).

2.4. Summary of Design Specifications

This chapter discussed the major design specifications for selecting a data model, designing a DICOM specific database schema, and for developing an API that provides services for schema information objects. We will summarize these specifications before moving on.

The data model must support highly mature relational technologies such as security, SQL, concurrency control, and transaction management. The data model must also allow user-defined data types and routines within the schema to accommodate the many ways DICOM information entities can be represented and stored. Two other requirements of the data model are that it supports class inheritance and that it can store complex data without requiring too many attribute references between objects.

In designing the schema for this project, the DICOM information model is used to identify all pertinent information objects to be represented in the schema. The DICOM model is a good starting point for the schema, but it can't be used directly because it is

based on non-normalized objects and it is not optimized for query processing. Table 2-1 lists the major requirements set forth for this project in designing the database schema. Of these requirements, the most notable ones are that the schema has to be application-level independent, be self-contained, be based on normalized objects, and incorporate type inheritance.

The schema alone is able to store all DICOM information objects, but it lacks facilities to provide DICOM specific services. An API is used to provide these services and to isolate the details of the schema from developers and database administrators who may not be familiar with the DICOM standard. Services that this API will provide are DICOM view functions, DICOM input and output functions, and schema extension functions.

Chapter 3 Conceptual Design of the Data Model

In the previous chapter, design specifications for the data model, database schema, and API were explained. This chapter explains the evaluation of several data models for this project.

The research involved in the development of a database data model is a massive undertaking that typically requires years and the cooperation of many individuals. Acknowledging this fact, this research project does not aim to develop a new database data model. Instead, this project uses work others have done in this regard and applies it to the diagnostic medical imaging domain. Specifically, this project evaluates existing database data models and selects the most appropriate one for managing DICOM information. This chapter gives an overview of existing data models and then present a discussion of the strengths and weaknesses of each with respect to managing DICOM information.

3.1. Overview of Existing Data Models

Since 1970 when E. F. Codd proposed the relational data model, much research has been done to refine relational transaction management, concurrency control, scalability, security, and distributed processing [19]. Today, these database technologies are so mature and reliable that they are the administrative hearts of most commercial and government institutions. Although the relational data model has served its purpose well for the last 15-20 years, more and more shortcomings have been discovered in recent years as demand for the management of complex data has risen [20]. Most notable of these shortcomings is the lack of a natural mechanism to represent nested information entities and a limited set of supported data types. Taken together, these limitations preclude the storage of unstructured data, such as images and audio, and render the representation of complex data difficult at the very least. These limitations have, however, provided impetus for the database community to develop novel alternatives. Three recent developments have come to the fore as promising alternatives: object-oriented databases, object-relational databases, and object-relational mappers [21]. These alternatives, along with the relational data model, comprise nearly all database models

currently in use or in development. Therefore, this project only evaluates these models and does not provide any treatment for older models like the hierarchical and network models.

3.2. Relational Data Model

The relational data model is based on relational theories in mathematics. Specifically, a relation is a set of ordered tuples defined over a set of not necessarily distinct domains. Each domain is itself a set. For example, given the sets (domains) D_1, D_2, and D_3, there can be a relation R defined over these domains such that each tuple in R contains one element from each of the three domains. The logical representation of this relation is a table containing three columns, one for each domain, to store attribute values. Each table row is a tuple representing an instantiation of the information entity represented by the table.

3.2.1. Advantages

The relational model is the most mature data model to date. There are several advantages of using this model for storing DICOM information. First, relational databases are so robust that many hospital information systems (HIS) for patient admissions, discharge, and transfer data use relational databases. Additionally, recall from Section 1.2 that many healthcare institutions have implemented commercial picture archiving and communications systems (PACS) based on this data model. This large number of HISs and PACSs currently in use is a good reason for using the relational model for the image archive because it makes interfacing the archive with legacy HISs and PACs much easier. A second reason for using the relational model is that it is just as good as any other model in handling certain types of data [22]. For example, non-complex data involving short string, integer, or real data types are probably most efficiently indexed using B-tree indexing schemes developed for the relational model [23]. Another advantage of this model is that many standards have been developed for it to promote interoperatability between commercially developed relational databases and their access applications. These standards include the Structure Query Language (SQL),

Open Database Connectivity (ODBC), and Java Database Connectivity (JDBC) protocols.

3.2.2. Disadvantages

As mentioned previously in Section 3.1, the recent move within the healthcare industry toward managing digital images has exposed weaknesses of the relational data model. The most obvious weaknesses stem from the fact that the relational model is not object-oriented. Relational databases lack built-in mechanisms to support object inheritance and to map between objects and tables. Without inheritance, an object-oriented information model stored in a relational database loses all of the semantic content represented in its inheritance hierarchy. Without this semantic content, the database has no mechanism to support such facilities as expanded query scope (see Section 2.2.5). The relational database also cannot present object-oriented interfaces (see Section 2.3.1) defined by object-oriented information models. Take for example a relational database used to store DICOM information. Within the database, all DICOM CIOD and NIOD instances will be disassembled and represented as rows in tables. Since the database cannot map between objects and their tabular representations, it exposes to its users essentially all attributes of all objects. Clearly, this is a violation of DICOM's use of encapsulation where objects expose their attributes only through strictly defined interfaces.

Other weaknesses of the relational model are related to its limited set of supported data types. One direct consequence of this limitation is the poor support for nested or hierarchical data. Consider, for example, trying to represent the relationships of the directed-acyclic graph illustrated in Figure 1-2 (page 19) using tables that can only store dates, strings, integers, etc. In Figure 3-1a, the one-to-many relationship between *Patient* and *Visit* is completely represented within the *Patient* table. This representation is quite natural, but it contains redundancies because Tom's *DOB* is stored more than one once. Figure 3-1b illustrates an alternative normalized representation that removes this redundancy from the *Patient* table, but this method introduces an extra table for the relationship between *Patient* and *Visit*. This extra table is an artifact of the normalization process because the columns of the *Patient* relation can only store simple data types (see

Sections 4.1.1 and 4.1.2 for a discussion of alternative ways of representing one-to-many relationship without artifact tables). Extra tables like these can be quite numerous in complex information models, and they are one of the many reasons behind the high cost of processing queries involving multiple objects. Another problem of having limited data types is the lack of native support for large unstructured data. In current relational databases, large data objects are stored as Binary Large Objects (BLOBs) where they reside outside of these databases' logical boundaries for concurrency control, logging, etc. [23]. These BLOBs are only represented within the database by pointers that reference their operating system files. As a result, large objects cannot be searched, indexed or read by random access, and if an external agent changes them, additional code is required to propagate these changes to the database.

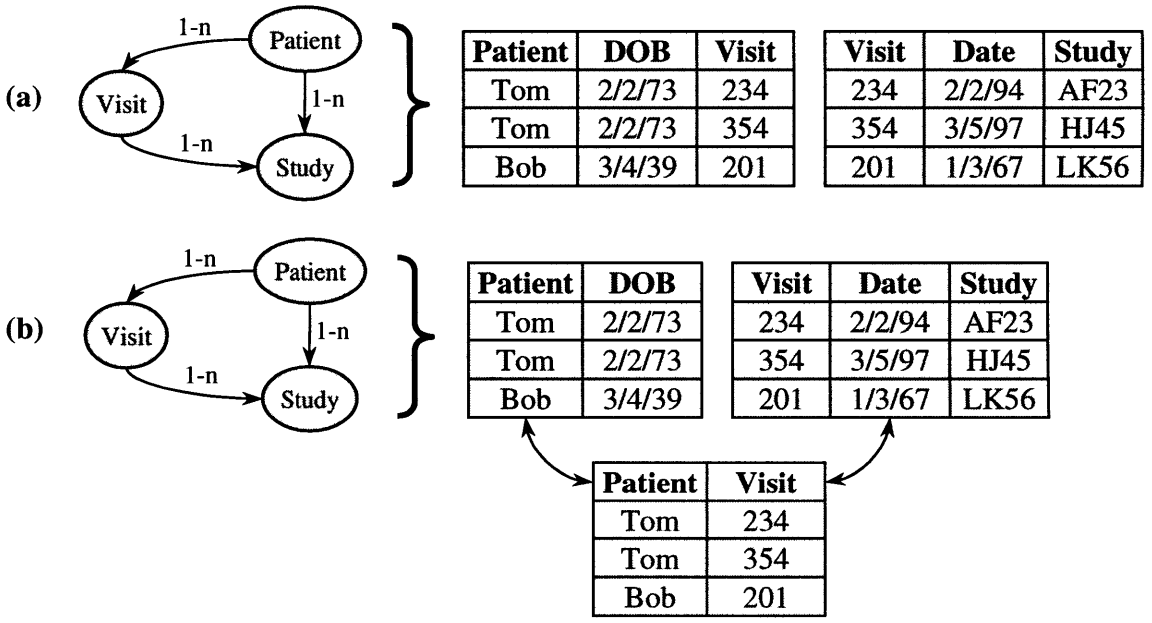


Figure 3-1 Two methods for representing one-to-many relationships between the patient, visit, and study entities. (a) This is a non-normalized representation because DOBs are stored more than once for patients who have more than one visits. (b) This is a normalized representation using an extra relationship table.

3.3. Object-Oriented Data Model

Object oriented databases (OODB) evolved in the mid-to-late 1980's following the development of object-oriented programming languages such as Smalltalk and C++. An

object-oriented database can be defined as a system that extends an existing object-oriented programming language with persistent data, concurrency control, query facilities, and other database capabilities [24]. These databases use a data model that incorporates unique object identifiers, data encapsulation, and inheritance. Unlike the relational data model, there is still no internationally accepted standard for defining an object-oriented data model (OODM).

3.3.1. Advantages

A clear advantage of the OODM is the ability to directly map between the DICOM information model and its persistent store. DICOM objects stored in an OODB are stored in data structures native to an object-oriented programming language. If these data structures are closely modeled after DICOM object definitions, there is very little mismatch between how DICOM defines objects and how they are represented in the database. Additionally, there will be very little difference between what the database exposes for its interface and what DICOM defines for object interfaces. Several other advantages of the OODM come from its support of object inheritance. When an OODB stores an object of a class belonging to an inheritance hierarchy, the database retains all of the object's relationships within this hierarchy. These relationships can then serve as the basis for defining rules for expanded query scopes. These relationships also facilitate the creation of classes by allowing new classes to inherit existing class attributes and/or methods.

Another benefit of inheritance is the re-use of objects and methods. Within an inheritance hierarchy with many classes that share many attributes, method inheritance can be used to dramatically reduce the number of methods that have to be created from scratch. Beside inheritance, the OODM allows object attributes to have any data structure as their domains. If these domains are defined as other classes, an OODB can easily represent nested and hierarchical data without creating artifacts, as was created with the relational data model (see Section 3.2.2). Eliminating these artifacts can potentially boost query-processing performance significantly.

A final advantage of the OODM is the notion of enforcing semantic integrity constraints. In relational databases, the primary integrity constraint on attribute values is

the data type designated for the attribute. For instance, if the integer data type is designated as the domain of an attribute, then the database will only allow integers to be stored for this attribute. This kind of integrity constraint does not necessarily imply any semantic constraint since many different meanings can be attached to generic data types like integers and strings. In an OODB, however, the domain of an attribute can be a class or any one of the simple data types available to the relational model. If the domain of an attribute is a class, the integrity constraint for this attribute can imply a PART-OF or CONSISTS-OF relationship between an object and the other objects that it references. These implied constraints can be made explicit to form the notion of composite objects, objects that are made of many component objects. This ability to strong type composite objects is very useful for the DICOM information model because the model is riddled with PART-OF relationships; images are PARTS-OF series and series are PARTS-OF studies.

3.3.2. Disadvantages

Most weaknesses of the object-oriented data model do not come from inherent limitations with the model. Instead, these weaknesses arise from the fact that the OODM, and hence commercial OODBs, are not as mature as relational databases. For instance, there has yet to be developed a national or international standard for creating an object-oriented data model. Consequently, no standards have been developed that is analogous to the SQL standard for relational databases. This lack of standardization within the OODB community has resulted in the development of many commercial systems each implementing a separate and proprietary model and interface. This means that if the image archive is developed based on an OODM, it will most likely be incompatible with other existing and/or future models. Within recent years, however, the Object Data Management Group* (ODMG), an independent consortium of OODB vendors, has proposed a vendor neutral object model and query language, but this work is far from widespread acceptance [24]. Other examples where the OODM trails the relational data model are in transaction management, replication support, and the sheer availability of access application development environments.

---

* Formerly known as the Object Database Management Group

Another disadvantage of using an OODB for DICOM information is that many HISs and PACs currently use relational databases. An image archive using an OODB will unduly complicate the migration process from existing legacy HISs and PACs to the newer model.

3.4. Object-Relational Mapper

Recall from Section 3.2.2 that the relational data model does not present an object-oriented interface. Many people have devised work-arounds to this problem in order to use the relational data model for storing object-oriented data [21, 22, 25, 26, 27]. One approach is to use an object-relational mapper (OR mapper) that serves as the interface between a relational database and its object-oriented access applications. An OR mapper presents an object interface to all access applications by mapping between object representations and their underlying tabular forms. Figure 3-2 illustrates the generic relationship between access applications, an OR mapper, and a relational database. Two approaches have been developed to perform object-to-relational mapping. In the first approach, the OR mapper translates each relation tuple to an object instance of the relation. This approach is called the table-equals-type (table=type) approach because it makes a one-to-one correspondence between tables and types (classes). The alternative approach is called the object-modeling approach. This approach involves translating a relational schema into an object model so that semantic concepts like inheritance and composite objects can be added the object model because they are missing from the relational model. An OR mapper using this approach performs object-to-relational mappings as well as enforces semantic constraints implied by inheritance.

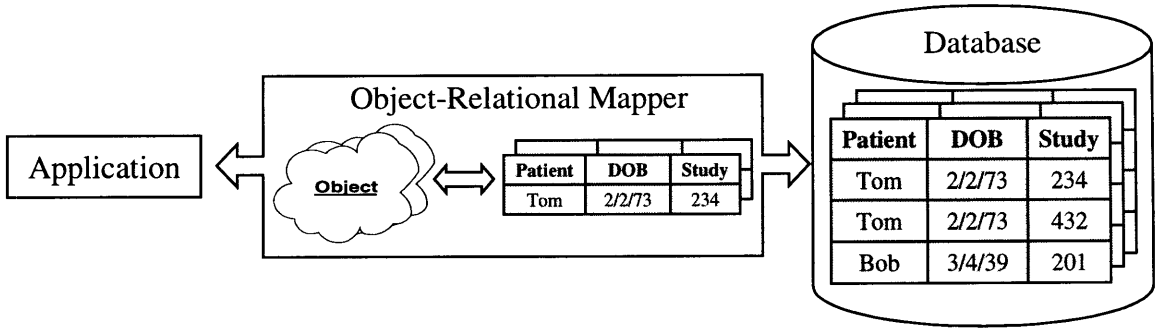


Figure 3-2 An object-relational mapper translates between an application's object-oriented data model and a relational database's relational model.

3.4.1. Advantages

The major benefit of using an OR mapper is the ability to present an object model for relational data. This ability relieves client applications of the burden of performing table to object conversion, and as a result, makes the overall system more scalable. Another benefit is that an OR mapper can present more than one object view for any given relational schema. This is quite convenient because requiring all access applications to share one common object model is not usually feasible.

3.4.2. Disadvantages

The underlying data in a system using an OR mapper is relational. Therefore, this approach shares many of the disadvantages of the relational model discussed in Section 3.2.2. Additionally, mappers based on the object modeling approach require high maintenance because they are intimately tied to the underlying schema. Changes in the schema will require corresponding changes in the object model. For mappers based on the table=type approach, objects lack the semantic content usually available in pure object-oriented models; a tuple mapped into an object by a table=type mapper will lack metadata such as its relationships within an inheritance hierarchy. The result is that client applications must now perform some of the semantic conversions usually performed by object-oriented databases.

3.5. Object-Relational Data Model

Stonebraker [28, 29] and, independently, Kim [23, 30] developed the object-relational data model (ORDM) in the early 1990's. This model extends the relational data model proposed by E. F. Codd to support object-oriented notions, such as encapsulation, inheritance, and class definitions having attributes and methods. In addition to these extensions, modern object-relational databases (ORDB) support data types for complex and unstructured data.

An ORDM can be thought of as a relational model where tables are considered as classes, tuples are object instantiations of classes, and columns of tables are attributes of their corresponding classes. Just as in the relational data model, tables can have constraints, storage options, triggers, indexes, and methods. Class inheritance is

supported by allowing tables (subtables) to inherit all or some of the columns, constraints, storage options, etc. of other tables (supertables). Within an ORDB, attribute domains can be abstract data types (ADT), user-defined data types (UDT), or any of the types supported in conventional relational databases. Abstract data types are data types constructed from other built-in types (integers, strings, char, etc.). These types allow attributes to possess structure or to store sets of values. User-defined data types are data types whose internal structures are completely opaque to the database; the database has no intrinsic mechanism to access such data. All UDTs, therefore, must be accompanied by user-defined access methods for reading, writing, indexing, and querying information contained within them. Allowing UDTs to have these access methods is one way in which ORDBs support encapsulation. Encapsulation is also supported by ORDBs when methods are assigned to tables and serve as the only mechanisms through which users access and manipulate table attributes [31].

3.5.1. Advantages

The ORDM combines the best both of the relational model and the object-oriented model. The ORDM can exploit the large relational user-base and its mature technologies and standards because the model is based on relations. Algorithms developed for concurrency control, transaction management, and query optimization as well as constructs from ODBC and SQL are still valid within the ORDM framework because the notion of relations and tuples are conserved. Unlike pure relational models, relations in the ORDM support two mechanisms for conserving the semantic content of object-oriented information models. First, tables support inheritance. All table-subtable relationships are maintained within the ORDB and they directly map to class-subclass relationships. The direct correspondences between tables and classes in inheritance hierarchies of the database and the underlying information model make explicit the notion of objects specializing other objects. Access applications can then use these explicit relationships without having to reconstruct them.

The second way some* ORDBs conserve semantic content is by allowing attribute domains to be tuples (objects). This facility allows strong enforcement of semantic integrity constraints as discussed in Section 3.3.1. Another advantage of the ORDM is that interface mismatch problems present in relational systems storing object-oriented data can be avoided. For example, in pure relational systems, the act of normalizing objects into tables violates object encapsulation because all object attributes become exposed via SQL. In ORDBs, however, objects can be represented by tables with opaque data types and user-defined access methods such that tables strictly adhere to the interface specifications of the underlying object-oriented model. Finally, the ORDM supports complex data types. These data types allow an attribute to possess structure or to contain a set of atomic values instead of just one. For example, a *person* abstract data type can be defined such that it contains fields for first name and last name. Although the type consists of two string data types, it is addressed and stored as one unit. Another example of a complex data type is the collection data type. Collections allow attributes to store multiple values. This type is quite useful in representing one-to-many relationships because no artifacts of normalization are created (see Section 3.2.2).

3.5.2. Disadvantage

There is currently no nationally accepted standard for defining object-relational data models. This problem was alluded to in Section 3.5.1 where it was noted that some commercial ORDBs support semantic integrity constraints while others do not. Other areas where vendors differ in their implementation of ORDMs include mechanisms to support collections and user-defined data types [23, 32, 33, 34, 35, 36]. To compound this problem, data type extensions, such as collections and user-defined data types, do not have SQL analogs or SQL means of access. Therefore, vendors have developed highly proprietary mechanisms to handle these data types. Besides differences in implementations among vendors, there are also problems inherent to having relations support collections and user-defined data types. For instance, the collection type can store more than one atomic value, so there is no meaning to indexing it via the B-tree

* Some implementations, such as IBM's Universal Database version 5.0, do not support system-wide object identifiers (OIDs), and thus they do not support semantic integrity constraints.

indexing scheme. One also cannot perform relational algebraic operations, such as the join operation, on this data type.

User-defined data types pose even a greater problem than collections since they are intrinsically not "understood" by the database environment. Not only are they not indexable* and not compatible with many relational algebraic operations, they can promote data redundancy. To clarify this point, consider an example where all objects are represented as opaque data within an object-relational database. This database schema would consist of one table with two columns, one for object IDs and another for object instances stored as opaque data. Figure 3-3 illustrates the example schema where the database stores opaque data for the DICOM ultrasound (US) CIOD. If more than one CIOD comes from a patient, a series, or a study, the patient's information, the series information, or the study information will be redundantly stored.

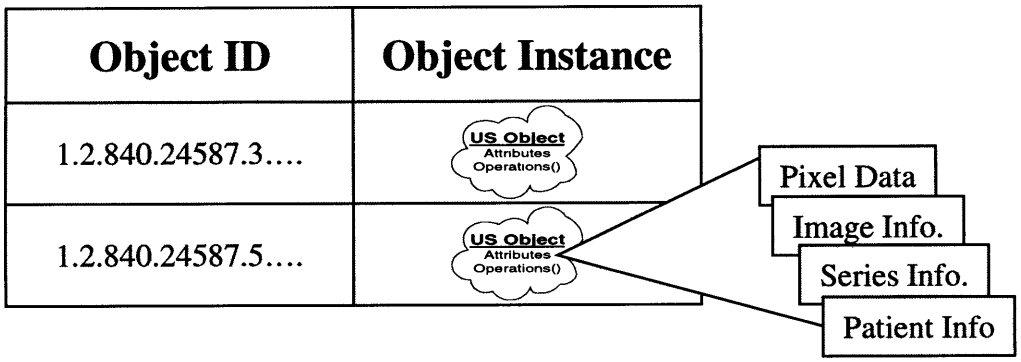


Figure 3-3 An example schema showing how opaque data types can be used to represent the DICOM ultrasound CIOD.

Although incompatibilities currently exist between different ORDMs, it should be noted that SQL3 is being developed by ISO and is expected to be ratified in 1999. SQL3 will provide a complete language and model for the management of persistent complex objects. This standard should provide vendors with a neutral model for implementing portable database environments.

* User-defined data type are not indexable by the traditional B-tree indexing scheme, but this does not preclude the development or use of other indexing schemes designed specifically for the type of information stored.

3.6. Data Model Selection

This section compares the data models presented in the previous sections with respect to design specifications of Section 2.1. Table 3-1 summarizes how each data model compares in term of supporting the design specifications.

Table 3-1 Comparison of different data models with respect to design requirements of Section 2.1

Data Model	Relational Technologies*	User-defined Data Types	Complex Data	Inheritance
Relational	√		-/√	
Object-Oriented		√	√	√
Object-to-Relational Mapper	√		√	
Object-Relational	√	√	√	√

*Relational technologies are SQL, ODBC, security, transaction management, concurrently control
√ = supported
-/√ = minimally supported

This project uses the object-relational data model because it supports all four of the design specifications shown in the table above. The relational data model and the object-to-relational mapping paradigm were not chosen because they do not support user-defined data types or type inheritance. The pure object-oriented data model was eliminated primarily because it lacks standardization and widespread acceptance.

Chapter 4 Design of the Schema

This section presents the design of how information will be organized within the image archive. This organization is referred to as the database schema, and it defines how DICOM NIODs, CIODs, and information entities (IE) are represented and related to one another in tabular form.

The schema design for this project uses a top-down approach involving two stages. The first stage develops a general framework for treating non-normalized CIODs; the framework determines whether or not CIODs are to be decomposed into normalized forms, and if so, the method for doing so. In the second stage, the framework is used to design a detailed entity-relationship diagram for the entire schema. The following sections present the framework design followed by the entity-relationship design.

4.1. Schema Framework Design

This project involves both relational and object-oriented design theories. As such, consideration must be given to issues involving the best use of user-defined data types, type inheritance, encapsulation, and relationship tables to maximize ease-of-use and to optimize query processing of CIODs. This project evaluated three frameworks for doing this and ultimately selected one. Each of the three alternatives is discussed below followed by an explanation of the selection process.

4.1.1. Framework Alternative 1: Opaque Data Types for all IODs and IEs

This schema alternative uses opaque data types to represent all DICOM information object definitions (IODs) and information entities. Recall from Section 3.5 that object-relational databases can support user-defined types (UDT) called opaque data types. These types are termed opaque because their data structures are not accessible to the database without user-defined methods. The idea behind this alternative is to represent all NIOD and CIOD objects as opaque data so that access to information stored within them are restricted to interfaces defined by their access methods. Figure 4-1 is an example schema based on this approach. It illustrates how the DICOM patient NIOD, study NIOD, CT image CIOD, and series IE are represented in tables. The patient, study,

and series tables are normalized, so their attribute values are self-contained. For the image table, however, a CT instance will inevitably have redundancies because it contains attributes pertaining to other objects, such as patients and studies (see Figure 4-1). This problem can be remedied by using user-defined access methods such that opaque data types for CIODs do not store attributes pertaining to other objects. Instead, these types retrieve attributes "on-the-fly" when needed.

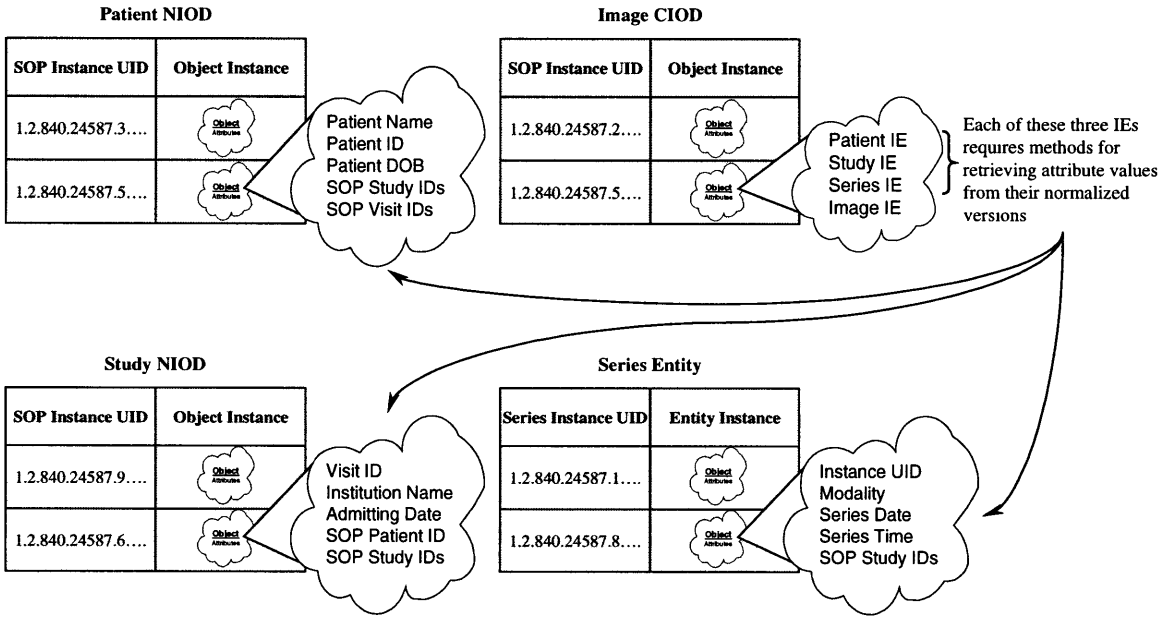


Figure 4-1 An example schema based on the opaque data type framework. The patient, study, and series tables contain normalized objects, whereas the image table contains non-normalized objects.

There are several reasons for desiring this approach. First, this method does not require CIODs to be decomposed into normalized components. Instead, the opaque data types can be made to preserve the class definitions of the DICOM information model; this eliminates the need for a second database-specific information model. The nested structure of the DICOM model is intrinsically preserved because opaque types can have attributes that are link-list structures like that of the ANSI C. Another benefit of this framework is that it enforces encapsulation to prevent applications from becoming too dependent on the underlying data structure. That means the schema can be changed without affecting how access applications perceive the underlying data store.

A major disadvantage of this approach is the sheer amount of effort required to develop access methods for each opaque data type. These access methods must not only provide basic input/output services but also provide query-optimizing services such as indexing and query cost analysis. If this project uses a schema similar to the one in Figure 4-1 a significant amount of time will be spent duplicating traditional database services for opaque data types. This problem is further compounded by the fact that there is currently no standardized way among database vendors for implementing opaque data types. That means access methods developed for one vendor's database environment will most likely be incompatible with another vendor's environment. Another problem with this framework is that there is no database intrinsic mechanism to extend opaque types using inheritance. This is because the data structures of opaque data types are not accessible to the database environment, so the database cannot implement and track attribute inheritance between opaque types. Users who want to use inheritance will have to provide their own user-defined methods to implement it. This work-around is still only a partial solution because facilities like expanded query scope require intrinsic database support and therefore will be not available.

4.1.2. Framework Alternative 2: Nested Tables using ADTs

This framework uses abstract data types (ADT) to represent the nested DICOM information model. The best way to understand this approach is by considering the example schema shown in Figure 4-2. This schema in uses nested tables to represent the directed-acyclic graph (DAG) of Figure 1-2 (page 19). Each tuple of *Patient_Table* represents a unique patient, and it contains columns for patient characteristics as well as a column named study of type *Study_ADT*. The type *Study_ADT* is a collection type where each field in the collection is an ADT for study specific information. That is, each field within the collection is an embedded table having one row and three columns to store three study attributes (Study, Date, Series). The Series column of this ADT is itself another collection type similar to the *Study_ADT* type. This pattern is recursively implemented such that the nested hierarchy of the schema starts at the *Patient_Table* and ends with ADTs for the image, curve, and overlay objects.

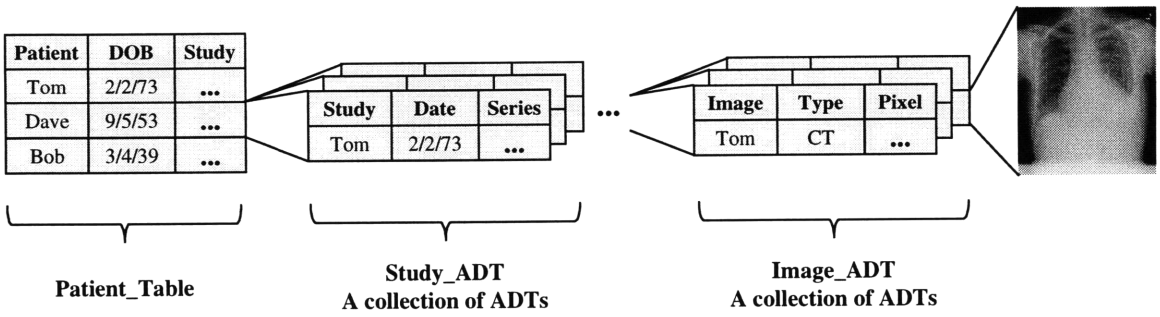


Figure 4-2 An example schema based on nested relations using abstract data types.

This particular approach provides a very intuitive representation of the DICOM information model. It preserves the nested structure of the DICOM model so users can traverse the schema from patient to image in a very transparent manner. For example, suppose one wants to select from the above schema all patients who owns at least one CT image. The SQL statements for this query would be as follows:

```
Select Patient from Patient_Table
Where Study.Series.Image.Type = "CT";
```

The DOT notation allows users to traverse the entire nested hierarchy without requiring multiple-table joins. Contrast this with a pure relational system where at least 7 tables are required (4 tables for the patient, study, series, and image objects and 3 relationship tables to relate patients to studies, studies to series, and series to images). The same query previously posed would require a minimum of six table joins.

Although abstract data types are natural for nested relations, there is currently no industry-wide standard for using and implementing them. Several vendors-specific implementations are available on the market, but they are for the most part incompatible with each other. As a result, most databases, including those from Informix, Oracle, and IBM, have very limited querying and updating facilities for ADTs. Another problem with this framework is that the schema is not very extensible. The *Patient_Table*, which embodies most of the schema, does not provide for a mechanism to separate and make independent each of the embedded information entities. As a result, changing the *Patient_Table* requires knowledge of the entire nested hierarchy to determine the correct

insertion point for new attributes. It also requires that one determine the effect of new attributes on the normalized entities above and below the insertion point.

4.1.3. Framework Alternative 3: Non-nested Tables

The idea behind this framework is to have a one-to-one correspondence between the tables of the schema and IODs and IEs of the DICOM information model. Tables for NIODs will have the same attributes as those defined for them within the standard. Tables for CIODs, however, will only contain attributes of the image IEs, curve IEs, overlay IEs, VOI LUT IEs, or modality LUT IEs that they represent. Other attributes that make a CIOD non-normalized will be accessed via their respective normalized tables. For example, a table for the CT Image CIOD will not contain attributes about patients, studies, or series to which images belong. Instead, the CT Image table will use primary and foreign key references to other tables for this information. Other IEs defined within CIODs (see Figure 1-1 on page 17) may or may not be represented as separate independent tables, depending on whether or not they are subsets of NIODs (see Section 1.1.1). IEs with NIOD counterparts will not be represented by separate tables, whereas IEs without NIOD counterparts will be represented by independent tables.

A major benefit of this framework over the first two alternatives is that it is based on a highly transparent schema. That means the usage of opaque and abstract data types is kept to a minimum so that all querying and data manipulation facilities of the relational model are applicable to the schema. This transparent schema is also less vendor-dependent than the previous two as it does not rely heavily on unstandardized data types. The schema is also very extensible because each table can have its own user-defined inheritance hierarchy below it.

This framework has several disadvantages compared to the first two alternatives. First, this schema does not preserve the nested structure of the DICOM information model as nested tables. Rather, it uses the more awkward approach based on relationship tables and table joins (see Section 3.2.2 for a discussion of artifacts of normalization). Another weakness of this framework is that it violates encapsulation by exposing the entire DICOM information model to SQL data manipulation commands. This problem

can be overcome, however, by using user-defined access methods assigned to tables (see Section 3.5)

4.1.4. Framework Selection

The main difference between the three alternatives presented in the previous section is what aspects of the DICOM information model each alternative preserves. The first alternative uses opaque data types to preserve both the class definitions and nested hierarchy of the DICOM model. This is accomplished at the expense of intrinsic database support for queries and data manipulation. The second framework uses a more transparent schema to preserve only the nested structure of the DICOM model using abstract data types. It allows for the direct navigation from Patient to Images using DOT notations, but it too sacrifices query support. The third framework limits the use of opaque and abstract data types to preserve only NIOD and normalized IE definitions. It does not preserve CIOD definitions nor does it support the nested structure of the DICOM model as naturally as the first two alternatives. This framework, however, does not sacrifice data manipulation or query support.

This project chooses framework 3 over the others for two main reasons. First, this framework is able to support the nested structure of the DICOM model to an acceptable level without foregoing support for query processing and data manipulation. Second, the framework provides developers and end-users with enormous flexibility and scalability through its treatment of inheritance and user-defined routines.

4.2. Entity-Relationship Design

This section presents the detailed schema design based on the framework chosen in the previous section. The goal of the detailed design is to construct an entity-relationship (E-R) diagram that specifies all tables of the schema and their interrelationships. The E-R design consisted of several steps, and each step is presented below.

4.2.1. E-R Design Step 1: Entity and Relationship Identification

The DICOM information model serves as the starting point for identifying information entities for the schema. Figure 4-3 on page 53 lists entities that are defined in the DICOM information model and how they are used within the schema. The table

also lists entities that are created for the schema but are not defined in the DICOM model. Some entries in the left column have indentations after them to indicate inheritance. For example, the entry for Image IOD has several objects listed under it to represent image objects that inherit attributes from the dcm_image table. Several entries have "Not represented" in their right columns because they pertain to print services, and therefore, are not used within schema. Three schema tables do not correspond to any DICOM information model entities. These are the dcm_ciod_prt, dcm_hospital, and dcm_physician tables. The dcm_ciod_prt table is created to allow expanded query scope between all images, curves, overlays, and modality and volume-of-interest (VOI) look-up-tables (LUT) (see Section 4.2.4 for a detailed discussion of tables for expanded query scopes). The dcm_hospital table is required to normalize the DICOM Visit NIOD (dcm_visit table) and the Equipment IE (dcm_equipment table) because these tables contain hospital information within their definitions. If these hospital-related attributes are kept within the dcm_visit or dcm_equipment tables, they would render both tables non-normalized. Similarly, the dcm_physician table normalizes the dcm_visit and dcm_study tables.

Relationship tables allow normalized tables to participate in multiple one-to-many relationships. As in the case with entities, most of the relationship tables in the schema are derived from relationships in the DICOM information model. Other relationship tables, however, are not represented in the DICOM model but are required for expanded query scopes (see Section 4.2.4).

Figure 4-3 on page 54 shows the E-R diagram for all entities and relationships of the schema. Primary key and foreign key relationships are represented as solid lines. Inheritance relationships are indicated with heavier dashed lines.

There is one major difference between the DICOM information model and the E-R diagram. The DICOM information model explicitly specifies a one-to-many PART-OF relationship between the Study Component IOD and its component images, curves, Modality LUTs, overlays and VOI LUTs. This relationship is not represented within the schema because the standard does not define Study Component IODs with all the necessary attributes for referencing images, curves, Modality LUTs, overlays and VOI LUTs. Instead, the standard defines the Study Component IOD with attributes for

referencing Series IEs, and provides Series IEs with attributes for referencing images, curves, etc. The schema uses a PART-OF relationship between Study Components and Series IEs and between Series IEs and it components. No connection is explicitly represented between Study Components and images, curves, etc.

Table 4-1 Correspondence between DICOM information objects and schema entities

DICOM Information Model Entities	Schema Tables
Annotation IOD	Not represented
Basic Study Descriptor IOD	Dynamically created into view
Equipment IE	Dcm_equipment
Nuclear Medicine Equipment IE	Dcm_nm_equipment
Secondary Capture Equipment IE	Dcm_sc_equipment
Film Box IOD	Not represented
Film Session IOD	Not represented
Frame of Reference IE	Dcm_frame_ref
Ultrasound Frame of Reference IE	Dcm_us_frame_ref
General Series IE	Dcm_series
Computed Radiography Series IE	Dcm_cr_series
Nuclear Medicine Series IE	Dcm_nm_series
Image Box IOD	Not represented
Image Overlay Box IOD	Not represented
Interpretation IOD	Dcm_interpret
Not represented	Dcm_ciod_prt
Image IOD	Dcm_image
Computed radiography image IOD	Dcm_cr_image
Computed tomography image IOD	Dcm_ct_image
Magnetic resonance image IOD	Dcm_mr_image
Nuclear medicine image IOD	Dcm_nm_image
Secondary capture image IOD	Dcm_sc_image
Ultrasound image IOD	Dcm_us_image
Ultrasound multi-frame image IOD	Dcm_us_mf_image
Stand-alone Curve IOD	Dcm_curve
Stand-alone Modality LUT IOD	Dcm_modality_lut
Stand-alone Overlay IOD	Dcm_overlay
stand-alone multi-frame overlay IOD	Dcm_modality_mf_lut
Stand-alone VOI LUT IOD	Dcm_voi_lut
Not represented	Dcm_hospital
Not represented	Dcm_physician
Patient IOD	Dcm_patient
Print Job IOD	Not represented
Printer IOD	Not represented
Results IOD	Dcm_results
Study Component IOD	Dcm_stdy_compnt
Study IOD	Dcm_study
Visit IOD	Dcm_visit
VOI LUT Box IOD	not represented

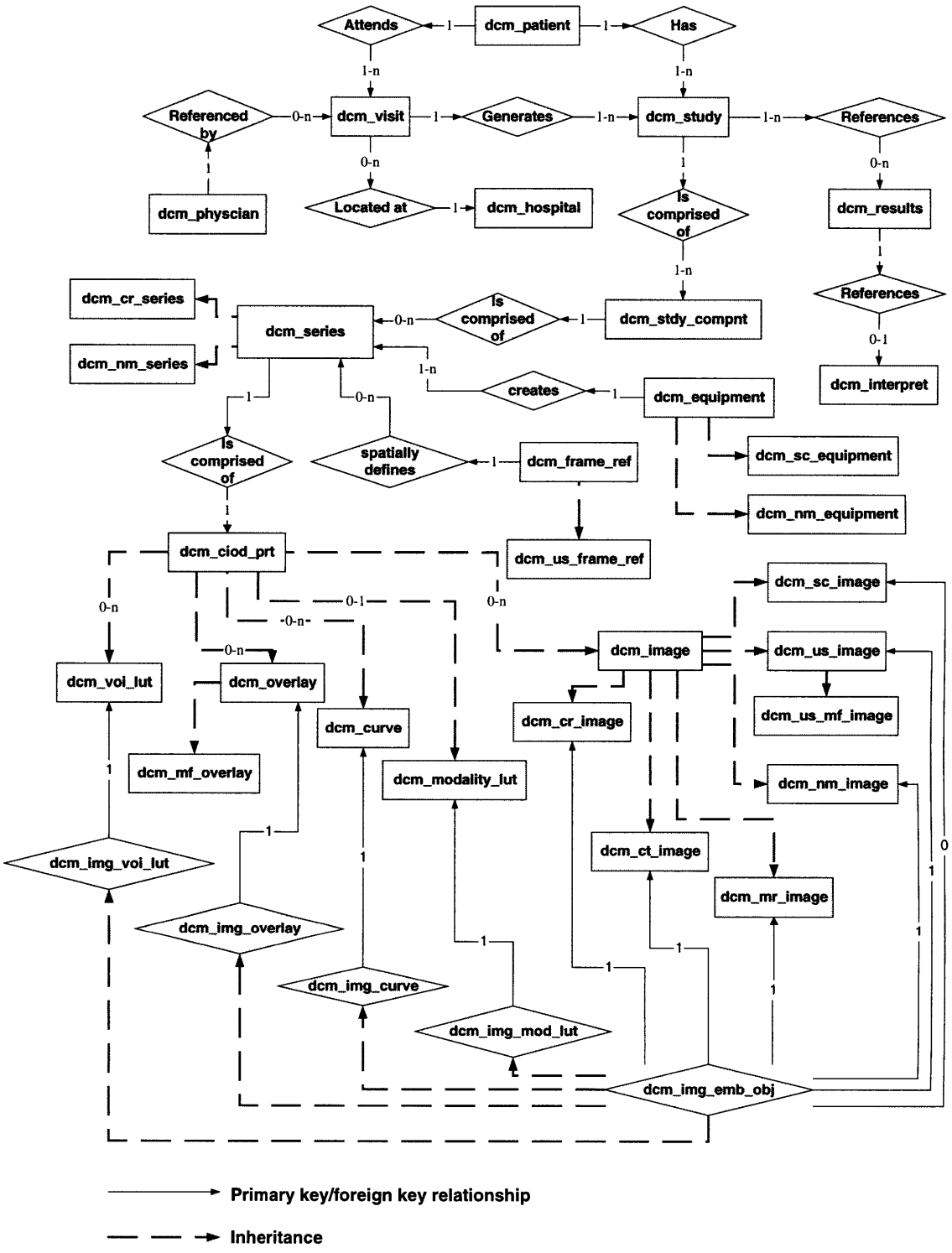


Figure 4-3 Entity-Relationship diagram for the image archive schema. Primary key and foreign key relationships are indicated with solid lines; heavier dashed lines indicate inheritance between objects.

4.2.2. E-R Design Step 2: Attributes Definition

The schema cannot directly use DICOM attribute names for table column names because many attribute names are longer than the eighteen-character limit of conventional databases. To avoid using an alternative naming convention, table columns are named after the unique tags* that are assigned to each DICOM attribute. Each column name within the schema is composed of the *'tag_'* string followed by an attribute group number (hexadecimal form), followed by an underscore character, followed by an attribute element number (hexadecimal form). For example, the column name for the *Patient's Birth Name* (0010,1005) attribute is 'tag_0010_1005'. Table A- 1 and Table A- 2 in Appendix A list all entity and relationship tables and their attributes. The DICOM value representation for each attribute is also listed. The correspondence between value representations and data types is discussed in Section 4.2.3. To convert between the English name of an attribute and its tag designation, the API dcm_deref() function is used. This API function is discussed in detail in the next chapter.

Some entity and relationship tables within the schema have attributes that are not DICOM defined attributes. The dcm_physican table includes a primary key column named dcm_mitx_phyx for storing unique identifiers for physicians. The DICOM standard does not make provisions for storing physician information separate from the visit and study NIODs, so a unique identifier has to be created if physician information is to be kept separate and not stored redundantly (see Section 4.2.1). The dcm_curve, dcm_overlay, dcm_modality_lut, dcm_voi_lut, and dcm_emb_obj tables contain a column for dcm_mitx_objx. This attribute is also not a DICOM defined attribute. It is required because DICOM image CIODs can have embedded curves, overlays, modality LUTs, and VOI LUTs, and these embedded objects are different from their standalone counterparts in that they do not have unique identifiers assigned to them. The dcm_mitx_objx attribute allows these embedded objects to have unique identifiers when they are separated from their CIODs (recall from Sections 2.2.1 and 2.2.2 that CIODs are separated into normalized components when they are stored within the database).

*DICOM attribute tags are composed of two ordered pairs of hexadecimal numbers representing a group number and an element number, respectively.

Schema entities are very similar to their DICOM counterparts in terms of attribute definitions, but several differences should be noted. These differences are explained below.

- The dcm_visit table does not include the following attributes of the Visit Identification Module (Table C.3.2-1 of Part 3 of the standard): Institution Name, Institution Address, Institution Code Sequence. These attributes are replaced by the Institution Code Sequence>>Code Value (0008,0100) and Institution Code Sequence>>Code Scheme Designator (0008,0100) attributes. The dcm_visit table does not include the following attributes of the Visit Admissions Module (Table C.3.4-1 of Part 3 of the standard): Referring Physician's Name, Referring Physician's Address, Referring Physician's Phone Numbers. These attributes are replaced by the dcm_mitx_phyx attribute.
- The dcm_study table does not contain the Series in Study and Acquisitions in Study attributes of the Study Acquisition Module (Table C.4.5-1 of Part 3 of the standard). These attributes are derivable attributes and should not be stored*.
- The dcm_stdy_compnt table does not include the following attributes of the Study Component Module (Table C.4.7-1 of Part 3 of the standard): Study ID, Study Instance UID, and Referenced Series Sequence and all of its elements. Study ID and Study Instance UID are stored within the dcm_study table and/or the dcm_stdy_stdycmp table. Elements of the Referenced Series Sequence are stored within the dcm_series table or the dcm_stdycmp_ser table.
- The dcm_image table does not have the Smallest Image Pixel Value and Largest Image Pixel Value attributes because they are derivable (Table C.7.6-3 of Part 3 of the standard).
- The dcm_cr_image table does not store the Exposure attribute because it is derivable (Table C.8.1-2 of Part 3 of the standard).
- The dcm_ct_image table does have the High Bit and Exposure attributes because they are derivable (Table C.8.2-1 of Part 3 of the standard).

* Derivable attributes require user-defined methods to derive their values. These methods are not developed for this project but are required in the case of a commercial implementation.

- The dcm_curve table does not contain the Minimum Coord. Value, Maximum Coord. Value, and Curve Range attributes because they are derivable (Table C10.2 of Part 3 of the standard).
- The dcm_overlay table does not have the ROI Mean and ROI Standard Deviation attributes because these are derivable (Table C.9.2 of Part 3 of the standard).

4.2.3. E-R Design Step 3: Data Type Definition

Table 4-2 below lists the correspondence between DICOM value representations and schema data types. All schema data types listed are SQL-3 standardized types, including the CLOB (Character Large Object) and BLOB (Binary Large Object) data types [37]. An attribute with the SQ value representation consists of a sequence of zero or more items, where each item contains a set of attributes. For example, the attribute *Patient's Insurance Plan Code Sequence* (0010,0050) has the SQ value representation because it contains fields for Code Value (0008,0100), Coding Scheme Designator (0008,0102) and Code Meaning (0008,0104). That is, this attribute specifies a nested structure where three other attributes are contained within it. Attributes that have the SQ value representation require abstract data type definitions because the multiset data type alone cannot store the nested structure these attributes posses.

Table 4-2 Mapping between DICOM value representations and schema data types

Value Representation	Meaning	Schema (SQL) data type
AE	Application Entity	Character varying(16)
AS	Age String	Char(4)
AT	Attribute Tag	Char(4)
CS	Code String	Character varying(16)
DA	Date	Date
DS	Decimal String	Decimal(16)
DT	Date Time	Datetime year to fraction
FD	Floating Point Double	Float
FL	Floating Point	Smallfloat
IS	Integer String	Integer
LO	Long String	Character varying(64)
LT	Long Text	CLOB
OB	Other Byte	BLOB
OW	Other Word	BLOB
PN	Personal Name	Character varying(64)
SH	Short String	Character varying(16)
SL	Signed Long	Integer
SQ	Sequence of Items	Multiset of abstract data types
SS	Signed Short	Smallint
ST	Short Text	BLOB
TM	Time	Datetime hour to fraction
UI	Unique Identifier	Character varying(64)
UL	Unsigned Long	Integer
US	Unsigned Short	Smallint

4.2.4. Designing for Expanded Query Scope

Recall from Section 2.2.5 that expanded query scope is the ability to include an entire inheritance hierarchy within the scope of a query. This ability allows users to quickly search and select all components of a composite object if the components share a large set of common attributes. For example, consider trying to select from the information model shown in Figure 1-2 (page 19) all images, curves, and overlays belonging to given a series. Without expanded query scope, this query requires a separate select statement for the overlay, image and curve entities. Within the actual DICOM information model

where there are actually seven types of images and four types of stand-alone objects each represented by different tables, this query requires a total of eleven compound statements. The schema used for this project overcomes this problem by creating two 'link' tables between objects that share common attributes. The first link table is the dcm_ciod_prt (CIOD PART) table (see Figure 4-3 on page 54). This table serves as the root for the inheritance hierarchy that includes all components of the series IE; all image modality tables and stand-alone object tables are sub-tables of the dcm_ciod_prt table. This inheritance hierarchy allows users to search all image modalities and stand-alone objects with just one select statement directed against the dcm_ciod_prt table. The select statement automatically searches all sub-tables under dcm_ciod_prt because the inheritance hierarchy is present. The second link table is the dcm_img_emb_obj relationship table. It allows users to search for all embedded objects of a given image CIOD. Recall that when a CIOD is stored within the archive, it is disassembled into normalized components. This disassembly process causes the image component of a CIOD to be stored in one of the dcm_image tables (dcm_cr_image, dcm_ct_image, etc.), while its embedded curve, overlay, and look-up tables are stored in the stand-alone tables (dcm_curve, dcm_overlay, etc.). Once these components are separated, they are completely independent of each other because there is no information within each of them that specifies the type of objects that they were embedded with before separation. For example, if a CIOD is disassembled into an image component and an embedded curve component, the image component does not contain information to indicate that the original CIOD also contained an embedded curve component. This is also true vice-versa such that no information regarding the image component is indicated in the curve component. That means reconstructing a complete CIOD requires exhaustively searching all relationship tables that link images to stand-alone objects until every component is identified. The image archive reduces the number of select statements required to perform this search by creating a relationship inheritance hierarchy rooted at the dcm_img_emb_obj table. This table allows one query, directed at this root table, to search all relationship tables between images and stand-alone objects.

4.3. Summary

This chapter has presented the schema design consisting of two stages. The first stage involved developing a framework for treating DICOM CIODs using both relational and object-oriented design methodologies. Three different frameworks were developed and evaluated for this project, but only one was selected. The selected framework decomposes CIODs into highly transparent and normalized tabular representations with minimal use of abstract and opaque data types.

In the second stage of the design process, a detailed entity-relationship diagram was developed based on the framework chosen in the first stage. This E-R diagram specifies all information entities to be represented within the schema, their interrelationships, and their attribute definitions and data types.

Chapter 5 Application Programming Interface

This Chapter presents the application programming interface (API). The API is a set of C language functions that implements the design requirements presented in Section 2.3. The API developed for this project is the result of a collaborative effort with significant contributions from Stefan Claesen under the guidance of Professor Richard Kitney at Imperial College in England and from Patrick McCormick and myself under the guidance of Professor C. Forbes Dewey here at MIT.

The API currently consists of the following functions:

```
dcm_bld_us_mf_ciod()
dcm_callback_handled()
dcm_ciod_to_tga()
dcm_create_class()
dcm_create_dict()
dcm_create_schema()
dcm_deref()
dcm_expl_ciod_f()
dcm_remove_schema()
dcm_view()
```

All functions are compiled into the shared library file DICOM.bld. Except for the dcm_callback_handled() function, all functions are SQL accessible functions. That means they are invoked within SQL statements* (see Section 6.1). The syntax presented in the following sections refer to SQL syntax.

* SQL accessible functions need to be declared within a database environment as external functions or procedures before they can be invoked within SQL statements. SQL functions are calls that have return values. SQL procedures are calls that do not have return values.

5.1. dcm_bld_us_mf_ciod()

This function is named for DICOM Build Ultrasound Multiframe CIOD. It is used to retrieve an entire DICOM Ultrasound Multi-frame Image CIOD with attribute values that match the supplied identifier list (see below for definition of an identifier list). This is the only IOD build function that this project developed. Other similar functions for building other modality images still need to be developed.

Syntax:

```
dcm_bld_us_mf_ciod("'attribute_name_1', 'value_1',
      'attribute_name_2', 'value_2', 'attribute_name_3',
      'value_3', …")

Note: 1) The identifier list is the sequence of attribute name and
      value pairs shown in the syntax above.

      2) The beginning and ending double quotation marks are
      required as well as interior single quotation marks around
      each attribute name and value.
```

Return Values:

On success, this function returns a data stream for composite IODs as a dcm_ciod_stream distinct type. For applications using this function through ODBC 3.0, the dcm_ciod_stream distinct type is analogous to the SQL_C_BINARY type.

Shared Library function in DICOM.bld

```
blob * dcm_bld_us_mf_ciod(mi_lvarchar *identifier, MI_FPARAM
      *Gen_fparam)
```

5.2. dcm_callback_handled()

This function is registered by every DICOM API function as the default callback to handle all events. Informix defines an event to be "something that occurs in the database server or in a user-defined routine that might be of interest to an application". This function is not registered within the database as an SQL-accessible function. Therefore, it cannot be accessed through SQL.

All DICOM functions that require the service of dcm_callback_handled() must register dcm_callback_handled() as its default callback. See Chapter 9 of the Informix DataBlade API documentation for details.

Shared Library function in DICOM.bld

```
int dcm_callback_handled (MI_EVENT_TYPE event_type, MI_CONNECTION
      *conn, void *cb_data, void *user_data)
```

5.3. dcm_ciod_to_tga()

This function is named for DICOM conversion from CIOD to Targa. It takes in a complete CIOD stream and outputs TGA images for each frame within the CIOD. The TGA files created are 24-bit images.

Syntax:

```
dcm_ciod_to_tga('temp_blob')

Note: temp_blob is a temporary SQL data instance created from the
      result of a select statement using the dcm_bld_us_mf_ciod()
      function.
```

Return Values:

On success, this function returns a data stream for TGA images as a dcm_tga_stream distinct type. If more than multiple TGA images are returned, as in the case with multiframe CIODs, individual frames are returned in the same manner that result sets with multiple values are returned. For applications using this function through ODBC 3.0, the dcm_tga_stream distinct type is analogous to the SQL_C_BINARY type.

Shared Library function in DICOM.bld

```
int dcm_ciod_to_tga(mi_lvarchar *identifier, MI_FPARAM
      *Gen_fparam)
```

5.4. dcm_create_class()

This function is used to extend the DICOM schema with subclasses to existing classes. For example, if Class_A is the parent class of Class_B, then extending the inheritance hierarchy with a new class, Class_C, such that it is a subclass of Class_B is extending the schema with subclasses.

Syntax:

```
dcm_creat_class("Name","'Attribute_name1','Attribute_name2','..'",
      "Parent_Object");

Note: 1) Name refers to name of the new class to be created.
      2) The beginning and ending double quotation marks are
      required as well as interior single quotation marks around
      each attribute name list.
      3) Parent_object refers to the name of the parent classes to
      be extended.
```

Return Values:

No return values. If an error occurs while creating the new subclass, the database server will issue an error message call to the user via standard output.

Shared Library function in DICOM.bld

```
int dcm_ciod_to_tga(mi_lvarchar *name, mi_lvarchar
      *attribute_list, mi_lvarchar *parent_object, MI_FPARAM
      *Gen_fparam)
```

5.5. dcm_create_dict()

This function is named for DICOM Create Dictionary. It reads from the file `dicitonary.dat` to retrieve all entries of the DICOM data dictionary and then populates the dcm_data_dict table (created by the dcm_create_schema() function) with these entries (entries are defined in Part 6 of the standard). The DICOM dictionary is the official registry of all DICOM data element tags, names (attribute), value representations (VR), and value multiplicity (VM). The logical representation of the dcm_data_dict table is given below:

Tag	Attribute	VR	VM
00080000	Group Length	UL	1
00080001	Length to End	RET	
00080005	Specific Character Set	CS	1
00080008	Image Type	CS	1-n
...	...	...	...

This table cannot be accessed through direct SQL Data Manipulation Language calls (select, insert, update, etc.). To access elements contained within the dcm_data_dict table, use the function dcm_deref().

The dcm_data_dict table allows the schema to be self-contained because it maps between attribute tag and attribute names (see Sections 2.2.2 and 4.2.2).

Syntax:

```
dcm_create_dict()
```

Return Values:

No return values. If an error occurs while populating the dcm_data_dict table, the database server will issue an error message call to the user via standard output.

Shared Library function in DICOM.bld

```
void dcm_create_dict (MI_FPARAM *Gen_fparam)
```

5.6. dcm_create_schema()

This function creates the entire DICOM schema as specified by the E-R diagram of Section 4.2.1.

Syntax:

```
dcm_create_schema()
```

Return Values:

No return values. If an error occurs while creating the schema, the database server will issue an error message call to the user via standard output.

Shared Library function in DICOM.bld

```
void dcm_create_schema(MI_FPARAM *Gen_fparam)
```

5.7. dcm_deref()

This function is overloaded to provide both tag-to-attribute and attribute-to-tag translation under the same function name. This function is used to retrieve a DICOM data element tag given an attribute name or to retrieve a DICOM attribute name given a data element tag.

Syntax:

```
dcm_deref("dcm_tag"::dcm_tag)
character varying (8) dcm_tag;

dcm_deref("attribute_name")
character varying (255) attribute_name;

NOTE: The quotation marks above are required as part of the SQL
      statement.
```

Return Values:

1) `dcm_deref("dcm_tag"::dcm_tag)` returns the name of the attribute corresponding to the dcm_tag parameter as defined by the DICOM data dictionary. The returned value has the character varying(255) SQL data type.
2) `dcm_deref("attribute_name")` returns the DICOM tag corresponding to the attribute_name parameter as defined by the DICOM data dictionary. The returned value has the dcm_tag distinct type.

NOTE: The dcm_tag data type is a distinct type created by dcm_create_schema() with the SQL data type char(8) as its source type. The type is an 8-byte string formatted as XXXXYYYY where XXXX and YYYY denote the DICOM group number and element number in hexadecimal, respectively.

Shared Library function in DICOM.bld

```
1) mi_lvarchar *dcm_tag_to_attribute(mi_lvarchar *tag, MI_FPARAM
      *Gen_fparam )

2) mi_lvarchar *dcm_attribute_to_tag(mi_lvarchar *attribute,
      MI_FPARAM *Gen_fparam)
```

5.8. dcm_ expl_ciod_f()

This function is name for DICOM Explode CIOD from File. It is called by a user when loading a DICOM CIOD file into the database schema.

Syntax:

```
Dcm_expl_ciod_f("file_name")

Note: file_name must specify the entire operation system file path
      to the file to be loaded.  The current implementation uses
      the UNIX path expression format.
```

Return Values:

No return values. If an error occurs while exploding a CIOD file into the schema, the database will issue an SQL error message call to the user.

Shared Library function in DICOM.bld

```
Void dcm_expl_ciod_f(mi_lvarchar *blobfile, MI_FPARAM *
      Gen_fparam)
```

5.9. dcm_remove_schema()

This function removes the entire DICOM schema, populated or not.

Syntax:

```
dcm_remove_schema()
```

Shared Library function in DICOM.bld

```
Void dcm_remove_schema(MI_FPARAM *Gen_fparam)
```

5.10. dcm_view()

This function creates a DICOM view for an NIOD or a CIOD (see Section 2.3.1).

Syntax:

```
Dcm_view("IOD_table);
```

Return Values:

No return values. If an error occurs while creating a DICOM view, the database will issue an SQL error message call to the user.

Shared Library function in DICOM.bld

```
Void dcm_view(mi_lvarchar *IOD_name, MI_FPARAM * Gen_fparam)
```

Chapter 6 Image Archive Prototype

This chapter discusses the prototype database that was developed based on the designs presented in Chapters 3, 4 and 5. The purpose of the prototype is to demonstrate how DICOM information stored within the schema can be extracted for viewing. The prototype consists of an Informix Dynamic Server database with the Universal Data Option* (IDS-UDO) and a software module that extends the generic database with capabilities to store, retrieve, and query DICOM information. This software module is called the DICOM DataBlade. Section 6.1 provides background information on the Informix database environment and how the image archive is implemented using the DICOM DataBlade. Section 6.2 discusses the configuration of the prototype.

6.1. Informix Database Environment

The IDS-UDO is a generic object-relational database, and as such, it does not have the necessary facilities to store and manage DICOM information. This project takes a generic IDS-UDO and extends it with a DICOM DataBlade module that provides DICOM specific facilities. The DICOM DataBlade provides new data type definitions for all DICOM value representations, a schema based on the E-R diagram developed in Chapter 4, and a compiled version of the API presented in Chapter 5. The DICOM DataBlade consists of four files: `prepare.sql`, `prepare.en_us.1252.sql`, `objects.sql`, and `DICOM.bld`. Their roles are explained below:

- The `prepare.sql` and `prepare.en_us.1252.sql` files register the DICOM DataBlade and its version number within the database's system tables. These file should not be changed.
- The `objects.sql` file performs several functions. First, it creates distinct data types for all value representations (VR) shown in Table 4-2 (based on their SQL data types) and then creates a bi-directional cast between each VR and its based SQL type. During this process, this file also creates two other distinct types, the dcm_tag distinct

* Formerly known as the Informix Universal Server. The prototype uses version 9.13 UC1 of the database engine running on a Sun UltraSparc I (170Mhz).

type and the dcm_vm distinct type. dcm_tag is required by the dcm_deref() function (see Section 5.7), while dcm_vm is required by the dcm_create_dict() function (see Section 5.3). After these type declarations are performed, the `objects.sql` file registers all API functions as external functions or external procedures. The registration process dynamically links the DICOM.bld shared library into the database environment's shared memory space. The last duty the `objects.sql` file performs is the execution of two SQL files. The first of these files calls the dcm_create_schema() function to create the DICOM schema. The second SQL file calls the dcm_create_dict() function to populate the dcm_data_dict table with entries from the DICOM data dictionary.

- The `DICOM.bld` file is the compiled shared library of all API functions presented in Chapter 5. Functions within this file are registered by the `objects.sql` file as described above.

In order to install DICOM DataBlade, one places a copy of `prepare.sql`, `prepare.en_us.1252.sql`, `objects.sql`, and `DICOM.bld` into the `$INFORMDIR/extend`* directory of the computer running the database. The DataBlade is then registered using BladeManager†.

6.2. Prototype Setup

To demonstrate the storage and extraction of DICOM images from the prototype, an access application is required to interact with the database. This project uses an HTTP server running a Common Gateway Interface (CGI) program as the access application because this approach leverages the user-interface of existing web browsers, and it allows the demonstration to reach a wider audience. The demonstration requires interaction between a client using a web browser, an intermediate server running an HTTP server, and the prototype image archive. The configuration of the demonstration is illustrated in Figure 6-1.

* `$INFORMIXDIR` is the directory where the IDS-UDO is installed.
† This product is provided by Informix and accompanies all releases of IDS-UDO.

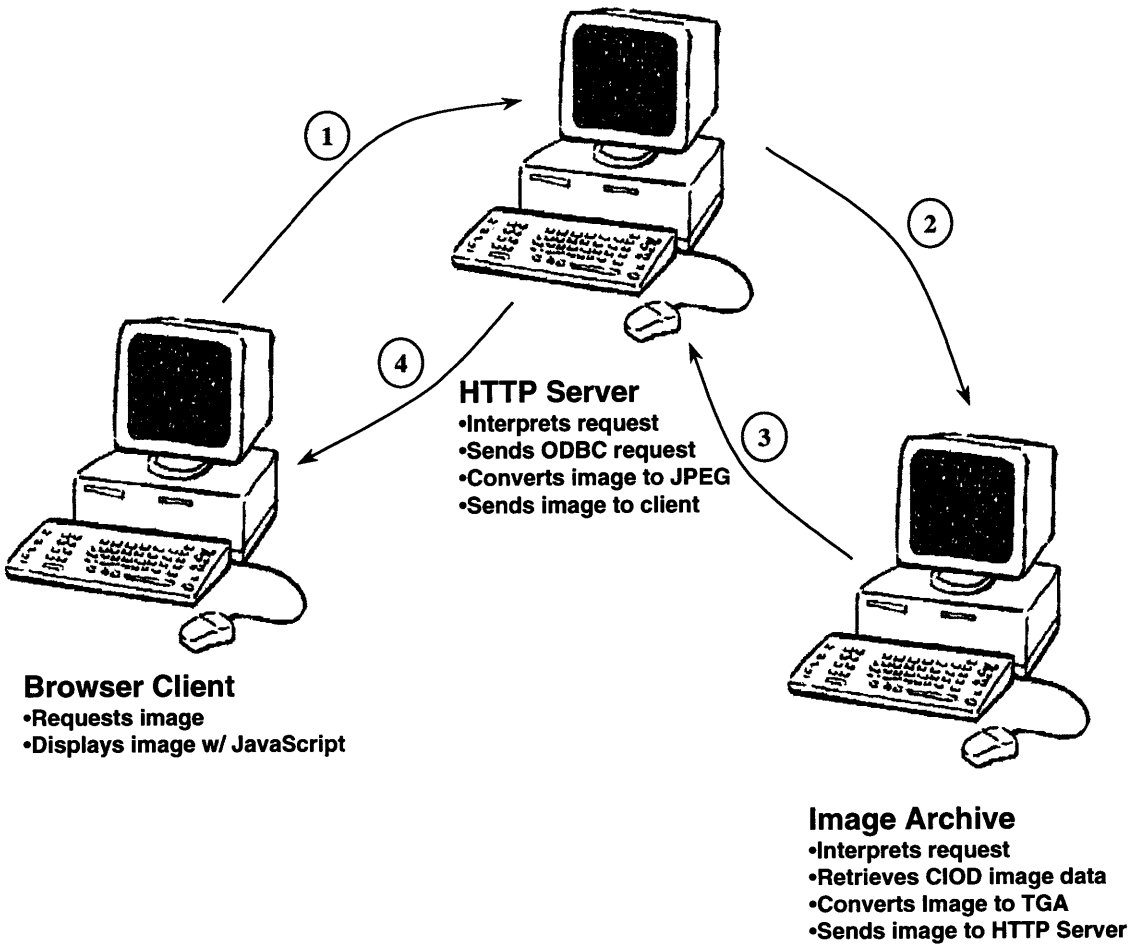


Figure 6-1 Configuration of the prototype demonstration consisting of three computers: a client using a web browser, a server running an HTTP daemon, and the prototype image archive.

Images are retrieved from the image archive when a web browser* submits a query to the HTTP server requesting a DICOM image. This request is sent via the Hypertext Transfer Protocol (HTTP), and when the HTTP server receives it, the server presents the request to a CGI application running on the same machine. This CGI application parses the request, translates it into an SQL query, and then sends the SQL query to the image archive via an ODBC 3.0 call. Once the image archive receives the SQL requests, it issues several DICOM DataBlade API calls. The first API call is to reconstruct the requested image's CIOD from its normalized representation within the schema. The

* The browser client can be any web browser that supports JavaScript 1.2.

second API call extracts the image component of the CIOD and passes this information to a third and final call that converts the image's pixel data to lossless TGA format. At the completion of the third API call, the image archive sends the converted image to the HTTP server where it undergoes another conversion from TIF to JPEG. This final image format is then sent to the client and displayed. Other scenarios, including delivery of the entire DICOM CIOD without conversion can easily be implemented. Figure 6-2 shows the client user-interface before and after a request for an ultrasound image.

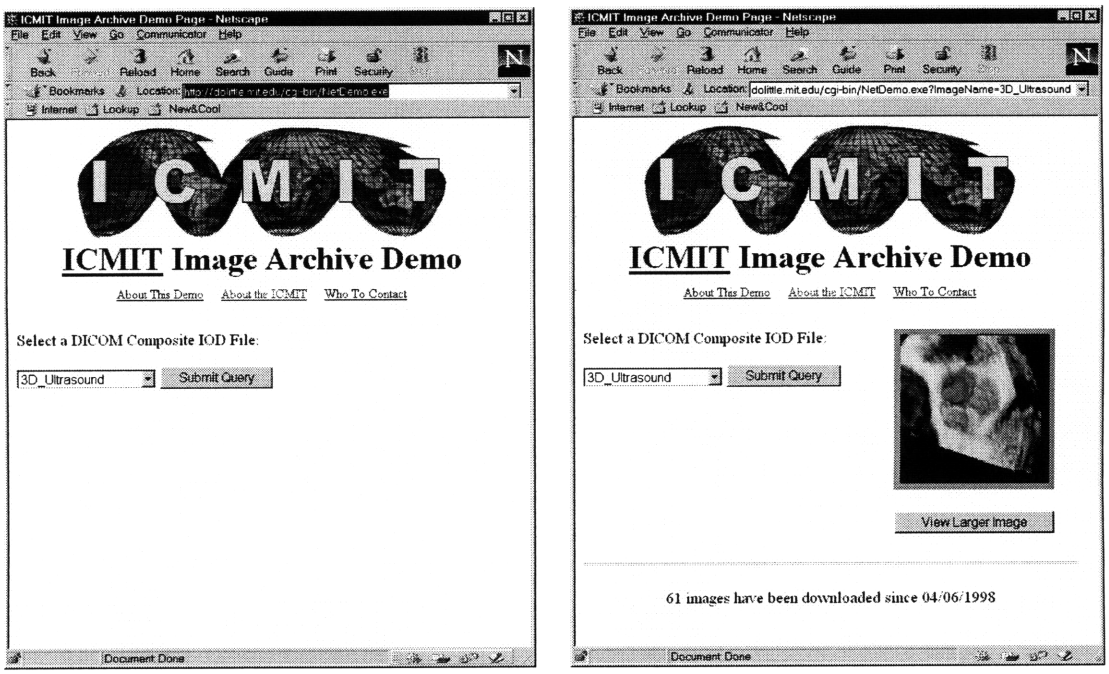


Figure 6-2 The user-interface developed for the browser client before (left) and after (right) image retrieval.

Chapter 7 Conclusion

This thesis has demonstrated the feasibility of extending a generic object-relational database with facilities for storing and managing DICOM information. The design and prototype presented in Chapters 4 through 6 provide a unified method for treating the simple, complex, and unstructured information objects specified in DICOM.

The schema design presented in Chapter 4 is based on an object-relational data model (ORDM). This data model was chosen because it is based on proven relational technologies and because it supports object-oriented concepts. The ORDM provides intrinsic support for type inheritance and data encapsulation. This project uses inheritance in two ways. First, inheritance is used to capture the implied notion (present in the DICOM information model) of objects specializing other objects. This notion is implemented as inheritance hierarchies within the schema, and these hierarchies serve as rules for expanded query scope. The schema also uses inheritance as the basis of its modular design that allows code re-use and transparent schema extensions. The ORDM also provides intrinsic support for data encapsulation, but this project did not implement user-define methods to enforce DICOM defined data access interfaces*.

DICOM information object definitions (IOD) and information entities are stored with the schema as normalized objects. Normalized representations of composite IODs are lossless decompositions of their original definitions. This approach requires that the schema be supported by user-defined methods for creating DICOM views, providing DICOM input/output services, and facilitating schema extensions. This project has developed these user-defined methods and several others in the form of an application-programming interface.

* These interfaces were not implemented as access methods because they are generally very application-environment specific. This project was not intended for any specific application environment.

Appendix A

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

Dcm_ciod_prt	
(PK) tag_0008_0018	UI NOT NULL
Dcm_curve	
(FK) tag_0008_1140	set(dcm_tag_0008_1155 not null),
tag_0008_0016	UI NOT NULL,
tag_0008_0005	CS,
tag_0008_0012	DA,
tag_0008_0013	TM,
tag_0008_0014	UI,
tag_0020_0024	IS,
tag_0008_0025	DA,
tag_0008_0035	TM,
tag_50xx_0005	US,
tag_50xx_0010	US,
tag_50xx_0020	CS,
tag_50xx_0103	US,
tag_50xx_3000	OB,
tag_50xx_0022	LO,
tag_50xx_0030	multiset(SH not null),
tag_50xx_0040	multiset(SH not null),
tag_50xx_0106	multiset(SH not null),
tag_50xx_0110	US,
tag_50xx_0112	US,
tag_50xx_0114	US
Dcm_equipment	
(PK) tag_0018_1000	LO,
tag_0008_0070	LO,
tag_0008_0080	LO,
tag_0008_0081	ST,
tag_0008_1010	SH,
tag_0008_1040	LO,
tag_0008_1090	LO,
tag_0018_1020	LO,
tag_0018_1050	DS,
tag_0018_1200	DA,
tag_0018_1201	TM,
tag_0028_0120	DS
Dcm_nm_equipment	
(PK) tag_0018_1000	LO,
tag_0008_0070	LO,
tag_0008_0080	LO,
tag_0008_0081	ST,
tag_0008_1010	SH,
tag_0008_1040	LO,
tag_0008_1090	LO,
tag_0018_1020	LO,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0018_1050 DS, tag_0018_1200 DA, tag_0018_1201 TM, tag_0028_0120 DS, tag_0018_1145 DS, tag_0018_1147 CS, tag_0018_1149 multiset(IS not null), tag_0018_1180 SH, tag_0018_1181 CS, tag_0018_1182 DS, tag_0018_1183 DS, tag_0018_1184 DS
Dcm_sc_equipment	
	(PK) tag_0018_1000 LO, tag_0008_0070 LO, tag_0008_0080 LO, tag_0008_0081 ST, tag_0008_1010 SH, tag_0008_1040 LO, tag_0008_1090 LO, tag_0018_1020 LO, tag_0018_1050 DS, tag_0018_1200 DA, tag_0018_1201 TM, tag_0028_0120 DS, tag_0008_0064 CS, tag_0008_0060 CS, tag_0018_1010 LO, tag_0018_1016 LO, tag_0018_1018 LO, tag_0018_1019 set(LO not null), tag_0018_1022 SH, tag_0018_1023 LO
Dcm_frame_of_ref	
	(PK) tag_0020_0052 UI,
Dcm_us_frame_of_ref	
	(PK) tag_0020_0052 UI, tag_0018_6018 UL, tag_0018_601A UL, tag_0018_601C UL, tag_0018_601E UL, tag_0018_6024 US, tag_0018_6026 US, tag_0018_602C FD, tag_0018_602E FD, tag_0018_6020 SL, tag_0018_6022 SL, tag_0018_6028 FD, tag_0018_602A FD
Dcm_hospital	

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	(PK) inst_code_value SH,
	(PK) inst_code_scheme SH,
	tag_0008_0080 LO,
	tag_0008_0081 ST
Dcm_image	
	(PK) tag_0008_0018 UI NOT NULL,
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL),
	(FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0020_0013 IS,
	tag_0020_0020 set(CS not null),
	tag_0008_0023 DA,
	tag_0008_0033 TM,
	tag_0008_0008 set(CS not null),
	tag_0020_0012 IS,
	tag_0008_0022 DA,
	tag_0008_0032 TM,
	tag_0008_2111 ST,
	tag_0020_4000 LT,
	tag_0028_0002 US,
	tag_0028_0004 CS,
	tag_0028_0010 US,
	tag_0028_0011 US,
	tag_0028_0100 US,
	tag_0028_0101 US,
	tag_0028_0102 US,
	tag_0028_0103 US,
	tag_7FE0_0010 OB,
	tag_0028_0006 US,
	tag_0028_0034 set(IS not null),
	tag_0028_1101 multiset(US not null),
	tag_0028_1102 multiset(US not null),
	tag_0028_1103 multiset(US not null),
	tag_0028_1201 multiset(US not null),
	tag_0028_1202 multiset(US not null),
	tag_0028_1203 multiset(US not null)
Dcm_cr_image	
	(PK) tag_0008_0018 UI NOT NULL,
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL),
	(FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0020_0013 IS,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0020_0020	set(CS not null),
	tag_0008_0023	DA,
	tag_0008_0033	TM,
	tag_0008_0008	set(CS not null),
	tag_0020_0012	IS,
	tag_0008_0022	DA,
	tag_0008_0032	TM,
	tag_0008_2111	ST,
	tag_0020_4000	LT,
	tag_0028_0002	US,
	tag_0028_0004	CS,
	tag_0028_0010	US,
	tag_0028_0011	US,
	tag_0028_0100	US,
	tag_0028_0101	US,
	tag_0028_0102	US,
	tag_0028_0103	US,
	tag_7FE0_0010	OB,
	tag_0028_0006	US,
	tag_0028_0034	set(IS not null),
	tag_0028_1101	multiset(US not null),
	tag_0028_1102	multiset(US not null),
	tag_0028_1103	multiset(US not null),
	tag_0028_1201	multiset(US not null),
	tag_0028_1202	multiset(US not null),
	tag_0028_1203	multiset(US not null),
	tag_0018_0010	LO,
	tag_0018_1040	LO,
	tag_0018_1041	DS,
	tag_0018_1042	TM,
	tag_0018_1043	TM,
	tag_0018_1044	DS,
	tag_0018_0060	DS,
	tag_0018_1004	DS,
	tag_0018_1110	DS,
	tag_0018_1111	DS,
	tag_0018_1150	IS,
	tag_0018_1151	IS,
	tag_0018_1152	IS,
	tag_0018_1170	IS,
	tag_0018_1400	LO,
	tag_0018_1401	LO,
	tag_0018_1402	CS,
	tag_0018_1403	CS,
	tag_0018_1404	US,
	tag_0018_1405	IS,
	tag_0018_6000	DS
Dcm_ct_image		
	(PK) tag_0008_0018	UI NOT NULL,
	(FK) tag_0008_1140	set(dcm_tag_0008_1155 not NULL),

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

(FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL),
tag_0008_0016 UI NOT NULL,
tag_0008_0005 CS,
tag_0008_0012 DA,
tag_0008_0013 TM,
tag_0008_0014 UI,
tag_0020_0013 IS,
tag_0020_0020 set(CS not null),
tag_0008_0023 DA,
tag_0008_0033 TM,
tag_0008_0008 set(CS not null),
tag_0020_0012 IS,
tag_0008_0022 DA,
tag_0008_0032 TM,
tag_0008_2111 ST,
tag_0020_4000 LT,
tag_0028_0002 US,
tag_0028_0004 CS,
tag_0028_0010 US,
tag_0028_0011 US,
tag_0028_0100 US,
tag_0028_0101 US,
tag_0028_0102 US,
tag_0028_0103 US,
tag_7FE0_0010 OB,
tag_0028_0006 US,
tag_0028_0034 set(IS not null),
tag_0028_1101 multiset(US not null),
tag_0028_1102 multiset(US not null),
tag_0028_1103 multiset(US not null),
tag_0028_1201 multiset(US not null),
tag_0028_1202 multiset(US not null),
tag_0028_1203 multiset(US not null),
tag_0028_0030 multiset(DS not null),
tag_0020_0037 multiset(DS not null),
tag_0020_0032 multiset(DS not null),
tag_0018_0050 DS,
tag_0020_1041 DS,
tag_0018_0010 LO,
tag_0018_1040 LO,
tag_0018_1041 DS,
tag_0018_1042 TM,
tag_0018_1043 TM,
tag_0018_1044 DS,
tag_0028_1052 DS,
tag_0028_1053 DS,
tag_0018_0060 DS,
tag_0018_0022 set(CS not null),
tag_0018_0090 DS,
tag_0018_1100 DS,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0018_1110 DS,
	tag_0018_1111 DS,
	tag_0018_1120 DS,
	tag_0018_1130 DS,
	tag_0018_1140 CS,
	tag_0018_1150 IS,
	tag_0018_1151 IS,
	tag_0018_1152 IS,
	tag_0018_1160 SH,
	tag_0018_1170 IS,
	tag_0018_1190 multiset(DS not null),
	tag_0018_1210 multiset(SH not null)
Dcm_mr_image	
	(PK) tag_0008_0018 UI NOT NULL,
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL),
	(FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0020_0013 IS,
	tag_0020_0020 set(CS not null),
	tag_0008_0023 DA,
	tag_0008_0033 TM,
	tag_0008_0008 set(CS not null),
	tag_0020_0012 IS,
	tag_0008_0022 DA,
	tag_0008_0032 TM,
	tag_0008_2111 ST,
	tag_0020_4000 LT,
	tag_0028_0002 US,
	tag_0028_0004 CS,
	tag_0028_0010 US,
	tag_0028_0011 US,
	tag_0028_0100 US,
	tag_0028_0101 US,
	tag_0028_0102 US,
	tag_0028_0103 US,
	tag_7FE0_0010 OB,
	tag_0028_0006 US,
	tag_0028_0034 set(IS not null),
	tag_0028_1101 multiset(US not null),
	tag_0028_1102 multiset(US not null),
	tag_0028_1103 multiset(US not null),
	tag_0028_1201 multiset(US not null),
	tag_0028_1202 multiset(US not null),
	tag_0028_1203 multiset(US not null),
	tag_0028_0030 multiset(DS not null),
	tag_0020_0037 multiset(DS not null),

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

tag_0020_0032	multiset(DS not null),
tag_0018_0050	DS,
tag_0020_1041	DS,
tag_0018_0010	LO,
tag_0018_1040	LO,
tag_0018_1041	DS,
tag_0018_1042	TM,
tag_0018_1043	TM,
tag_0018_1044	DS,
tag_0018_0020	set(CS not null),
tag_0018_0021	set(CS not null),
tag_0018_0022	set(CS not null),
tag_0018_0023	CS,
tag_0018_0080	DS,
tag_0018_0081	DS,
tag_0018_0091	IS,
tag_0018_0082	DS,
tag_0018_1060	DS,
tag_0018_0024	SH,
tag_0018_0025	CS,
tag_0018_0083	DS,
tag_0018_0084	DS,
tag_0018_0085	SH,
tag_0018_0086	multiset(IS not null),
tag_0018_0087	DS,
tag_0018_0088	DS,
tag_0018_0089	IS,
tag_0018_0093	DS,
tag_0018_0094	DS,
tag_0018_0095	DS,
tag_0018_1062	IS,
tag_0018_1080	CS,
tag_0018_1082	IS,
tag_0018_1083	IS,
tag_0018_1084	IS,
tag_0018_1085	LO,
tag_0018_1086	IS,
tag_0018_1088	IS,
tag_0018_1090	IS,
tag_0018_1094	IS,
tag_0018_1100	DS,
tag_0018_1250	SH,
tag_0018_1251	SH,
tag_0018_1310	multiset(US not null),
tag_0018_1312	CS,
tag_0018_1314	DS,
tag_0018_1316	DS,
tag_0018_1315	CS,
tag_0018_1318	DS,
tag_0020_0100	IS,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0020_0105 IS,
	tag_0020_0110 DS
Dcm_nm_image	
	(PK) tag_0008_0018 UI NOT NULL,
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL),
	(FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0020_0013 IS,
	tag_0020_0020 set(CS not null),
	tag_0008_0023 DA,
	tag_0008_0033 TM,
	tag_0008_0008 set(CS not null),
	tag_0020_0012 IS,
	tag_0008_0022 DA,
	tag_0008_0032 TM,
	tag_0008_2111 ST,
	tag_0020_4000 LT,
	tag_0028_0002 US,
	tag_0028_0004 CS,
	tag_0028_0010 US,
	tag_0028_0011 US,
	tag_0028_0100 US,
	tag_0028_0101 US,
	tag_0028_0102 US,
	tag_0028_0103 US,
	tag_7FE0_0010 OB,
	tag_0028_0006 US,
	tag_0028_0034 set(IS not null),
	tag_0028_1101 multiset(US not null),
	tag_0028_1102 multiset(US not null),
	tag_0028_1103 multiset(US not null),
	tag_0028_1201 multiset(US not null),
	tag_0028_1202 multiset(US not null),
	tag_0028_1203 multiset(US not null),
	tag_0028_0030 multiset(DS not null),
	tag_0020_0037 multiset(DS not null),
	tag_0020_0032 multiset(DS not null),
	tag_0018_0050 DS,
	tag_0020_1041 DS,
	tag_0018_1063 DS,
	tag_0018_1065 multiset(DS not null),
	tag_0008_2142 IS,
	tag_0008_2143 IS,
	tag_0008_2144 IS,
	tag_0018_0040 IS,
	tag_0018_1066 DS,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0018_0073 DS,
	tag_0018_1242 IS,
	tag_0028_0008 IS,
	tag_0028_0009 AT,
	tag_0028_0031 multiset(DS not null),
	tag_0018_0070 IS,
	tag_0018_0071 CS,
	tag_0018_1100 DS,
	tag_0018_1110 DS,
	tag_0018_1130 DS,
	tag_0018_1131 DS,
	tag_0018_1141 DS,
	tag_0018_1142 multiset(DS not null),
	tag_0018_1210 multiset(SH not null),
	tag_0018_5020 LO,
	tag_0018_5021 LO,
	tag_0020_0015 IS,
	tag_0020_0016 IS,
	tag_0020_0017 IS,
	tag_0020_0018 IS,
	tag_0028_0032 multiset(DS not null),
	tag_0028_0051 CS,
	tag_0018_1061 LO,
	tag_0018_1060 DS,
	tag_0018_1062 IS,
	tag_0018_1064 LO,
	tag_0018_1080 CS,
	tag_0018_1081 IS,
	tag_0018_1082 IS,
	tag_0018_1083 IS,
	tag_0018_1084 IS,
	tag_0018_1085 LO,
	tag_0018_1086 IS,
	tag_0018_1088 IS,
	tag_0018_1090 IS,
	tag_0018_1144 IS,
	tag_0018_1143 DS,
	tag_0018_1140 CS,
	tag_0018_1146 multiset(DS not null)
Dcm_us_image	
	(PK) tag_0008_0018 UI NOT NULL,
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL),
	(FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0020_0013 IS,
	tag_0020_0020 set(CS not null),

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

tag_0008_0023	DA,
tag_0008_0033	TM,
tag_0008_0008	set(CS not null),
tag_0020_0012	IS,
tag_0008_0022	DA,
tag_0008_0032	TM,
tag_0008_2111	ST,
tag_0020_4000	LT,
tag_0028_0002	US,
tag_0028_0004	CS,
tag_0028_0010	US,
tag_0028_0011	US,
tag_0028_0100	US,
tag_0028_0101	US,
tag_0028_0102	US,
tag_0028_0103	US,
tag_7FE0_0010	OB,
tag_0028_0006	US,
tag_0028_0034	set(IS not null),
tag_0028_1101	multiset(US not null),
tag_0028_1102	multiset(US not null),
tag_0028_1103	multiset(US not null),
tag_0028_1201	multiset(US not null),
tag_0028_1202	multiset(US not null),
tag_0028_1203	multiset(US not null),
tag_0018_0010	LO,
tag_0018_1040	LO,
tag_0018_1041	DS,
tag_0018_1042	TM,
tag_0018_1043	TM,
tag_0018_1044	DS,
tag_0018_6011	set(tag_0018_6011 not null),
tag_0028_0009	AT,
tag_0008_2124	IS,
tag_0008_212A	IS,
tag_0008_2120	SH,
tag_0008_2122	IS,
tag_0008_2128	IS,
tag_0008_2129	IS,
tag_0008_2130	set(DS not null),
tag_0008_2132	set(LO not null),
tag_0008_2200	CS,
tag_0008_2204	CS,
tag_0008_2208	CS,
tag_0018_1060	DS,
tag_0018_1062	IS,
tag_0018_1080	CS,
tag_0018_1081	IS,
tag_0018_1082	IS,
tag_0018_1088	IS,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0018_5000 set(SH not null), tag_0018_5010 multiset(LO not null), tag_0018_6031 CS, tag_0018_5012 DS, tag_0018_5020 LO, tag_0018_5022 DS, tag_0018_5024 DS, tag_0018_5026 DS, tag_0018_5027 DS, tag_0018_5028 DS, tag_0018_5029 DS, tag_0018_5050 IS, tag_0018_5210 DS, tag_0018_5212 DS
Dcm_us_mf_image	
	(PK) tag_0008_0018 UI NOT NULL, (FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL), (FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL), tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0020_0013 IS, tag_0020_0020 set(CS not null), tag_0008_0023 DA, tag_0008_0033 TM, tag_0008_0008 set(CS not null), tag_0020_0012 IS, tag_0008_0022 DA, tag_0008_0032 TM, tag_0008_2111 ST, tag_0020_4000 LT, tag_0028_0002 US, tag_0028_0004 CS, tag_0028_0010 US, tag_0028_0011 US, tag_0028_0100 US, tag_0028_0101 US, tag_0028_0102 US, tag_0028_0103 US, tag_7FE0_0010 OB, tag_0028_0006 US, tag_0028_0034 set(IS not null), tag_0028_1101 multiset(US not null), tag_0028_1102 multiset(US not null), tag_0028_1103 multiset(US not null), tag_0028_1201 multiset(US not null), tag_0028_1202 multiset(US not null), tag_0028_1203 multiset(US not null),

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0018_0010	LO,
	tag_0018_1040	LO,
	tag_0018_1041	DS,
	tag_0018_1042	TM,
	tag_0018_1043	TM,
	tag_0018_1044	DS,
	tag_0018_6011	set(tag_0018_6011 not null),
	tag_0028_0009	AT,
	tag_0008_2124	IS,
	tag_0008_212A	IS,
	tag_0008_2120	SH,
	tag_0008_2122	IS,
	tag_0008_2128	IS,
	tag_0008_2129	IS,
	tag_0008_2130	set(DS not null),
	tag_0008_2132	set(LO not null),
	tag_0008_2200	CS,
	tag_0008_2204	CS,
	tag_0008_2208	CS,
	tag_0018_1060	DS,
	tag_0018_1062	IS,
	tag_0018_1080	CS,
	tag_0018_1081	IS,
	tag_0018_1082	IS,
	tag_0018_1088	IS,
	tag_0018_5000	set(SH not null),
	tag_0018_5010	multiset(LO not null),
	tag_0018_6031	CS,
	tag_0018_5012	DS,
	tag_0018_5020	LO,
	tag_0018_5022	DS,
	tag_0018_5024	DS,
	tag_0018_5026	DS,
	tag_0018_5027	DS,
	tag_0018_5028	DS,
	tag_0018_5029	DS,
	tag_0018_5050	IS,
	tag_0018_5210	DS,
	tag_0018_5212	DS,
	tag_0018_1063	DS,
	tag_0018_1065	multiset(DS not null),
	tag_0008_2142	IS,
	tag_0008_2143	IS,
	tag_0008_2144	IS,
	tag_0018_0040	IS,
	tag_0018_1066	DS,
	tag_0018_0072	DS,
	tag_0018_1242	IS,
	tag_0028_0008	IS
Dcm_sc_image		

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

(PK) tag_0008_0018 UI NOT NULL, (FK) tag_0008_1140 set(dcm_tag_0008_1155 not NULL), (FK) tag_0008_2112 set(dcm_tag_0008_1155 not NULL), tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0020_0013 IS, tag_0020_0020 set(CS not null), tag_0008_0023 DA, tag_0008_0033 TM, tag_0008_0008 set(CS not null), tag_0020_0012 IS, tag_0008_0022 DA, tag_0008_0032 TM, tag_0008_2111 ST, tag_0020_4000 LT, tag_0028_0002 US, tag_0028_0004 CS, tag_0028_0010 US, tag_0028_0011 US, tag_0028_0100 US, tag_0028_0101 US, tag_0028_0102 US, tag_0028_0103 US, tag_7FE0_0010 OB, tag_0028_0006 US, tag_0028_0034 set(IS not null), tag_0028_1101 multiset(US not null), tag_0028_1102 multiset(US not null), tag_0028_1103 multiset(US not null), tag_0028_1201 multiset(US not null), tag_0028_1202 multiset(US not null), tag_0028_1203 multiset(US not null), tag_0018_1012 DA, tag_0018_1014 TM
Dcm_interpret
(PK) tag_0008_0018 UI NOT NULL, tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_4008_0200 SH, tag_4008_0202 LO, tag_4008_0210 CS, tag_4008_0212 CS, tag_4008_0100 DA, tag_4008_0101 TM,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_4008_0102 PN, tag_4008_0103 LO, tag_4008_0108 DA, tag_4008_0109 TM, tag_4008_010A PN, tag_4008_010B ST, tag_4008_010C PN, tag_4008_0111 set(dcm_tag_4008_0111 not null), tag_4008_0115 LT, tag_4008_0117 set(dcm_tag_4008_0117 not null), tag_4008_0118 set(dcm_tag_4008_0118 not null)
Dcm_modality_lut	
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not null), tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0028_3000 set(dcm_tag_0028_3000 not null), tag_0028_1052 DS, tag_0028_1053 DS, tag_0028_1054 LO, tag_0020_0026 IS
Dcm_overlay	
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not null), tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0020_0022 IS, tag_0008_0024 DA, tag_0008_0034 TM, tag_60xx_0010 US, tag_60xx_0011 US, tag_60xx_0050 multiset(SS not null), tag_60xx_0100 US, tag_60xx_0102 US, tag_60xx_3000 OW, tag_60xx_1301 IS, tag_60xx_1100 US, tag_60xx_1101 US, tag_60xx_1102 US, tag_60xx_1103 US, tag_60xx_1200 multiset(US not null), tag_60xx_1201 multiset(US not null), tag_60xx_1202 multiset(US not null), tag_60xx_1203 multiset(US not null)
Dcm_mf_overlay	
	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not null),

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0020_0022 IS, tag_0008_0024 DA, tag_0008_0034 TM, tag_60xx_0010 US, tag_60xx_0011 US, tag_60xx_0050 multiset(SS not null), tag_60xx_0100 US, tag_60xx_0102 US, tag_60xx_3000 OW, tag_60xx_1301 IS, tag_60xx_1100 US, tag_60xx_1101 US, tag_60xx_1102 US, tag_60xx_1103 US, tag_60xx_1200 multiset(US not null), tag_60xx_1201 multiset(US not null), tag_60xx_1202 multiset(US not null), tag_60xx_1203 multiset(US not null), tag_60xx_0015 IS
Dcm_patient	
	(PK) tag_0010_0020 LO, (FK) tag_0038_0004 SQ, tag_0008_0018 UI, tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0010_0010 PN, tag_0010_0021 LO, tag_0010_1000 set(LO NOT NULL), tag_0010_1001 set(PN NOT NULL), tag_0010_1005 PN, tag_0010_1060 PN, tag_0010_1090 LO, tag_0010_1040 LO, tag_0010_2152 LO, tag_0010_2150 LO, tag_0010_2154 set(SH NOT NULL), tag_0010_0030 DA, tag_0010_0032 TM, tag_0010_2160 SH, tag_0010_0040 CS, tag_0010_1020 DS, tag_0010_1030 DS,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0010_1080 LO), tag_0010_1081 LO, tag_0010_0050 set(dcm_tag_0010_0050 NOT NULL), tag_0010_21f0 LO, tag_0010_4000 LT, tag_0038_0500 LO, tag_0010_21C0 US, tag_0010_2000 set(LO NOT NULL), tag_0010_2110 set(LO NOT NULL), tag_0038_0050 LO, tag_0010_21D0 DA tag_0010_21A0 CS, tag_0010_21B0 LT
Dcm_physician	
	(PK) tag_mitx_phyx UI, tag_0008_0090 PN, tag_0008_0092 ST, tag_0008_0094 set(SH NOT NULL)
Dcm_results	
	(PK) tag_0008_0018 UI NOT NULL, tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_4008_0040 SH, tag_4008_0042 LO, tag_4008_0300 ST, tag_4008_4000 ST
Dcm_series	
	(PK) tag_0020_000E UI, (FK) tag_0018_1000 LO, (FK) tag_0020_0052 UI, tag_0020_0011 IS, tag_0020_0060 CS, tag_0008_0021 DA, tag_0008_0031 TM, tag_0008_1050 set(PN NOT NULL), tag_0018_1030 LO, tag_0008_103E LO, tag_0008_1111 set(PN NOT NULL), tag_0018_0015 CS, tag_0008_5100 CS, tag_0028_0108 US, tag_0028_0109 US
Dcm_cr_series	
	tag_0018_5101 CS, tag_0018_1160 SH, tag_0018_1180 SH, tag_0018_1190 set(DS NOT NULL),

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0018_1260 SH,
	tag_0018_1261 LO
Dcm_nm_series	
	tag_0018_0030 set(LO not Null),
	tag_0008_0042 CS,
	tag_0018_1300 IS,
	tag_0018_1301 set(CS not Null),
	tag_0018_1302 IS,
	tag_0018_0031 set(LO not null),
	tag_0018_0032 DS,
	tag_0018_0033 multiset(DS not null),
	tag_0018_0034 LO,
	tag_0018_0035 TM,
	tag_0018_0072 DS,
	tag_0018_1045 multiset(IS not null),
	tag_0018_1061 LO,
	tag_0018_1070 multiset(LO not null),
	tag_0018_1071 multiset(DS not null),
	tag_0018_1072 multiset(TM not null),
	tag_0018_1073 multiset(TM not null),
	tag_0018_1074 multiset(DS not null),
	tag_0018_1120 DS,
	tag_0020_0014 IS
Dcm_stdy_compnt	
	(PK) tag_0008_0018 UI NOT NULL,
	(FK) tag_0008_1110 set(dcm_tag_0008_1155 not null),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0008_1030 LO,
	tag_0008_1032 dcm_tag_0008_1032,
	tag_0008_1050 set(PN NOT NULL),
	tag_0032_1055 CS
Dcm_study	
	(PK) tag_0020_000D UI,
	tag_0008_0018 UI NOT NULL,
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0008_0050 SH,
	tag_0020_0010 SH,
	tag_0032_0012 LO,
	tag_0020_1070 set(IS NOT NULL),
	tag_0032_000A CS,
	tag_0032_000C CS,
	tag_0032_4000 LT,

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	tag_0032_1000 DA, tag_0032_1001 TM, tag_0032_1010 DA, tag_0032_1011 TM, tag_0032_1020 LO, tag_0032_1021 multiset(AE not null), tag_0032_1030 LO, tag_0032_1033 LO, tag_0032_1060 LO, tag_0032_1064 set(dcm_tag_0032_1064), tag_0032_1070 LO, tag_0032_1040 DA, tag_0032_1041 TM, tag_0008_0020 DA, tag_0008_0030 TM, tag_0032_1050 DA, tag_0032_1051 TM, tag_0032_0032 DA, tag_0032_0033 TM, tag_0008_1060 PN, tag_0032_0034 DA, tag_0032_0035 TM
Dcm_visit	
	(PK) tag_0008_0018 UI NOT NULL, tag_0008_0016 UI NOT NULL, tag_0008_0005 CS, tag_0008_0012 DA, tag_0008_0013 TM, tag_0008_0014 UI, tag_0038_0010 LO, tag_0038_0011 LO, tag_0038_0008 CS, tag_0038_0300 LO, tag_0038_0400 LO, tag_0038_4000 LT, tag_0038_0020 DA, tag_0038_0021 TM, tag_0038_0016 LO, tag_0008_1080 LO, tag_0008_1084 dcm_tag_0008_1084, tag_0038_0030 DA, tag_0038_0032 TM, tag_0038_0040 LO, tag_0038_0044 dcm_tag_0038_0044, tag_0038_001A DA, tag_0038_001B TM, tag_0038_001C DA, tag_0038_001D TM, tag_0038_001E LO
Dcm_voi_lut	

Table A- 1 Attributes of all entities of the schema. (PK) represents primary keys, (FK) represents foreign keys. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

	(FK) tag_0008_1140 set(dcm_tag_0008_1155 not null),
	tag_0008_0016 UI NOT NULL,
	tag_0008_0005 CS,
	tag_0008_0012 DA,
	tag_0008_0013 TM,
	tag_0008_0014 UI,
	tag_0028_3010 set(dcm_tag_0028_3010 not null),
	tag_0028_1050 multiset(DS not null),
	tag_0028_1051 multiset(DS not null),
	tag_0028_1055 set(LO not null)

Table A- 2 Relationship tables within the schema. The naming convention used is the following: dcm_entity1_entity1 represents a relationship table between entity1 and entity2. (PK) represents primary key, (FK) represents foreign key. Next to each attribute is its value representation as defined in Part 6 of the DICOM 3.0 standard.

dcm_patient_visit
(PK)(FK) tag_0008_1120 UI (PK)(FK) tag_0008_1125 UI
dcm_patient_study
(PK)(FK) tag_0008_1120 UI, (PK)(FK) tag_0008_1110 UI
dcm_study_results
(FK) (PK) tag_0008_1110 UI, (FK) (PK) tag_0008_1100 UI
dcm_reslt_intprt (results and interpret)
(FK) (PK) tag_0008_1100 UI, (FK) (PK) tag_4008_0050 UI
dcm_phys_visit (physician and visit)
FK) (PK) tag_0008_1125 UI, FK) (PK) tag_mitx_phyx UI
dcm_visit_hospital
(PK)(FK) tag_0008_1125 UI, (PK)(FK) inst_code_value SH, (PK)(FK) inst_code_scheme SH
dcm_visit_study
(FK) (PK) tag_0008_1125 UI, (FK) (PK) tag_0008_1110 UI
dcm_stdy_stdycmp (study and study component)
(FK) (PK) tag_0008_1110 UI, (FK) (PK) tag_0008_1111 UI
dcm_stdycmp_ser (between study component and series)
(FK) (PK) tag_0008_1111 UI, (FK) (PK) tag_0008_1115 UI,
dcm_ser_ciod_prt (series and ciod_prt)
(PK)(FK) tag_0008_1115 UI, (PK)(FK) tag_0008_1140 UI
dcm_img_emb_obj
(PK)(FK) tag_0008_1140 UI, (PK) tag_mitx_obj UI
Dcm_img_curve
(PK)(FK) tag_0008_1140 UI, (PK)(FK) tag_mitx_obj UI
dcm_img_mod_lut
(PK)(FK) tag_0008_1140 UI, (PK)(FK) tag_mitx_obj UI
dcm_img_overlay
(PK)(FK) tag_0008_1140 UI, (PK)(FK) tag_mitx_obj UI
dcm_img_voi_lut
(PK)(FK) tag_0008_1140 UI, (PK)(FK) tag_mitx_obj UI

References

[1] C. P. Waegmann. "The five levels of electronic health records," *MD computing*, vol. 13, no. 3, pp. 199-203, 1996.

[2] American College of Radiologists and the National Electrical Manufacturers Association, *Digital imaging and Communications in Medicine (DICOM) Version 3.0*, Washington, DC: National Electrical Manufacturers Association 1993.

[3] American College of Radiologists and the National Electrical Manufacturers, *Digital Imaging and Communications ACR-NEMA 300-1985*. Washington, DC: National Electrical Manufacturers Association, 1985.

[4] American College of Radiologists and the National Electrical Manufacturers, *Digital Imaging and Communications ACR-NEMA 300-1988*. Washington, DC: National Electrical Manufacturers Association, 1988.

[5] S. C. Horiil et al., "Introduction to the ACR-NEMA DICOM standard," *Radiographics*, vol. 12, no. 2, Mar., pp. 345-355, 1992.

[6] W. Kim, *Introduction to Object-Oriented Databases.* Cambridge, MA: The M.I.T Press, 1990.

[7] R. K. Taira, N. J. Mankovich, M. I. Boechat, H. Kangarloo, and H. K. Huang, "Design and implementation of a picture archiving and communication-system for pediatric radiology," *American Journal Of Roentgenology*, vol. 150, no. 5, May, pp. 1117-1121, 1988.

[8] H. K. Huang and R. K. Taira, "Infrastructure design of a picture archiving and communication system," *American Journal of Roentgenology*, vol. 158, no. 4, Apr., pp. 743-749, 1992.

[9] H. K. Huang, R. K. Taira, S. L. Lou, A. W. K. Wong, C. Breant, B. K. T. Ho, K. S. Chuang, B. K. Stewart, K. Andriole, R. Tecotzky, T. Bazzill, S. L. Eldredge, J. Tagawa, Z. Barbaric, M. I. Boechat, T. Hall, J. Bentson, and H. Kangarloo, "Implementation of a large-scale picture archiving and communication-system," *Computerized Medical Imaging And Graphics*, vol. 17, no. 1, Jan-Feb, pp. 1-11, 1993.

[10] S. T. C. Wong and H. K. Huang, "Design methods and architectural issues of integrated medical image data base systems," *Computerized Medical Imaging and Graphics*, vol. 20, no. 4, pp. 285-299, 1996.

[11] "Image Server DICOM Conformance Statement," Duke University Medical Center-Department of Radiology, Durham, NC. 1996.

[12] "DICOM Conformance Statement IMPAX Storage and Print Services," AGFA Medical Division, Mortsel Belgium, 1997.

[13] "SIENET MagicStore DICOM Conformance Statement," Siemens AG, 1996.

[14] "GE PACS Conformance Statement for DICOM v3.0," General Electric, Mt. Prospect, IL. Technical Publication IIS P/N 4361668 Revision 01, 1998.

[15] S. T. C. Wong and H. K. Huang, "A hospital integrated framework for multimodality image base management," *IEEE Transactions on Systems Man and Cybernetics Part A-Systems And Humans*, vol. 26, no. 4, Jul., pp. 455-469, 1996.

[16] I. T. Hawryszkiewycz, *Relational Database Design: An Introduction*. New York: Prentice Hall, 1990.

[17] I. T. Hawryszkiewycz, *Database Analysis and Design*. Chicago : Science Research Associates, 1984.

[18] W. Kent, "Limitations of record-based information models," *ACM Transactions on Database Systems*, vol. 4, no. 1, pp. 107-131, 1979.

[19] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, June, pp. 377-387, 1970.

[20] P. Turner and A. M. Keller, "Reflections on object-relational applications," OOPSLA workshop on object and relational databases, Austin, TX, October 1995.

[21] A. M. Keller, "Penguin: Objects for programs, relations for persistence," submitted for publication, April 1994.

[22] J. Lee and D. Forslund, "Coexistence of Relational and Objects in Distributed Object Computing," [Online document], June 1995, [cited 1996 Oct 15], Available HTTP: http://www.acl.lanl.gov/sunrise/dbms/index.html.

[23] W. Kim, "Object-relational database technology: a UniSQL whitepaper," UniSQL Inc., Austin, TX. 1996.

[24] R. G. G. Cattell, Ed., *The Object Database Standard: ODMG-93 Release 1.2*. San Francisco: Morgan Kaufmann, 1996.

[25] A. M. Keller, R. Jensen, and S. Agarwal, "Persistence Software: Bridging Object-Oriented Programming and Relational Databases," appeared in *SIGMOD*, May 1993.

[26] A. M. Keller, "Updating relational databases through views," Ph.D. dissertation, Stanford University, 1985.

[27] A. M. Keller and C. Hamon, "A C++ Binding for Penguin: a System for Data Sharing among Heterogeneous Object Models," appeared in Foundations on Data Organization (FODO) 93, Chicago, October 1993.

[28] M. Stonebraker and G. Kemnitz, "The Postgres next-generation database-management system," *Communications of the ACM*, vol. 34, no. 10, pp. 78-92, 1991.

[29] M. Stonebraker and D. Moore, *Object-Relational DBMS: The Next Great Wave*. San Francisco, CA: Morgan Kaufmann, 1996.

[30] S. Gala and W. Kim, "Database design methodology: Converting a relational schema into an object-relational schema," presented at ACM Symposium on Advanced Database Systems and Their Integration, Japan, October 1994.

[31] W. Kim, "Completeness criteria for object-relational database systems," UniSQL Inc., Austin, TX. 1996.

[32] Informix Software, Inc., *Informix Guide to SQL: Tutorial*, Version 9.1, Menlo Park: Informix Press, 1997.

[33] "Leveraging digital image assets using the Oracle8 Image Cartridge and the Oracle8 Visual Information Retrieval Cartridge: an Oracle business white paper," Oracle Corporation, Redwood Shores, CA., June 1997.

[34] "Oracle8 Data Cartridge development: an Oracle technical white paper," Oracle Corporation, Redwood Shores, CA., June 1997.

[35] "Oracle8 Image Data Cartridges: an Oracle technical white paper," Oracle Corporation, Redwood Shores, CA., June 1997.

[36] "DB2 Universal Database: Everything your database has been missing," IBM Corporation, San Jose, CA., 1997.

[37] J. Melton, Ed., ANSI SQL3 Papers SC21 N9463 through SC21 N9467, ANSI SC21 Secretariat, +1.212.642.4900, New York, 1995.