Storage Model for CDA Documents

Zheng Liang, Peter Bodorik, Michael Shepherd
Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada B3H 1W5
zhengl@cs.dal.ca, bodorik@cs.dal.ca, shepherd@cs.dal.ca

Abstract

The Health Level 7 Clinic Document Architecture (CDA) is an XML-based document markup standard that specifies the hierarchical structure and semantics of "clinical documents" for the purpose of information exchange. In this research, issues arising with the design and implementation of a DB to support efficient retrieval from CDA documents and data mining for statistical analysis purposes are explored. Both an object-relational approach and a traditional relational approach were explored and compared in terms of design, implementation issues and efficiency. Although the object-relational approach results in a simpler design, implementation is more complicated as object methods must be programmed. In the relational design, queries were more complex to express than in the object-oriented design, but more efficient to execute. It was concluded that the DB design should use standard relational tables while using objects only when required for specialized processing, such as processing of graphs or scans.

1. Introduction

With the rapid development of the Internet and the World Wide Web, the computer-to-computer exchange of business documents in a structured, predefined standard format has become more and more significant. This exchange has been greatly facilitated by the development of extensible structured markup languages such as XML. A similar need for the electronic exchange of clinical documents exists in healthcare. Until recently, however, standards for clinical document exchange among clinical systems covered messaging of fielded data but did not meet the need for semantic processing of hierarchical, structured, clinical documents. This has now been addressed by the Clinical Document Architecture (CDA).

ANSI/HL7 CDA R1.0-2000 is the first nationally certified XML-based standard for healthcare [1]. It has been developed by Health Level 7 (HL7), an ANSI-accredited Standards Developing Organization operating in the healthcare arena. HL7's mission is to enable clinical interoperability through the provision of "… standards for the exchange, management and integration of data that support clinical patient care and the management, delivery and evaluation of healthcare services." [7].

As the CDA becomes used more widely and databases of CDA documents are created, the efficient retrieval of data from these documents and the statistical analysis of the contents of these documents become an issue. Right now, most hospitals and health related organizations are using relational databases due to their wide availability, powerful query and analysis tools and flexibility to integrate with existing business databases. These organizations may use, for various reasons such as legacy information systems, a hierarchical database to store operational data, and then dump this data into relational-based data warehouses for statistical analysis and data mining. However, when CDA documents are mapped into a relational database, the native hierarchical structure of the XML encoding and the hierarchical structure of the CDA may be lost. In addition, relational databases are weak in supporting the nesting and recursive structures inherent in CDA documents.

In this research, the relational database management (RDBMS) and the object-relational database management system (ORDBMS) are examined and compared for the purpose of storage and query of CDA documents. The ORDBMS is able to handle complex, object-centric, persistent data while keeping the powerful query and analysis tools of the RDBMS. ORDBMS are often considered to be a bridge between RDBMS and OODBMS (object-oriented DBMS) by combining the ease of use of RDBMS and the flexibility of OODBMS to handle complex data types. For the comparison, simplicity, flexibility, coverage, query performance and storage space are considered.

Section 2 of this paper briefly presents an overview of the Clinical Document Architecture with examples from the Structured Discharge Summaries used in previous research [10]. Section 3 briefly discusses the selection of DBs to support processing of CDA documents and also briefly contrasts the ORDBMS with the RDBMS. In Section 4, the DB design is presented by first developing an ER model and then translating it to the relational and

Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)
0-7695-1874-5/03 $17.00 © 2002 IEEE

object-relational data models. Section 5 discusses results of the preliminary comparison of the relational and object-relational DBs in terms of several characteristics, such as delays, for a set of queries. The final section offers a summary and concluding remarks.

2. CDA Overview

The Clinical Document Architecture has been in development since 1996, originally as the Kona Architecture [8], then as the Patient Record Architecture (PRA), and now as the CDA [4]. It is a document markup standard for the structure and semantics of exchanged "clinical documents". A CDA document is a defined and complete information object that can exist outside of a message and can include text, images, sounds, and other multimedia content.

CDA documents derive their meaning from the HL7 Reference Information Model (RIM) [5]. The RIM provides a coherent shared information model that contains all data content relevant to HL7 messages, and is an essential part of the HL7 Version 3 development methodology. It represents the semantic and lexical connections between the information carried in the fields of HL7 messages and has evolved to a flexible and general model of clinical information.

The Clinical Document Architecture is a three-layer architecture implemented in XML, where each level is defined by a DTD. Level One is the root of the hierarchy and each additional level adds further specificity and constraints to the architecture. Level One specifies the semantics of the header, codes for the document type and sections within the body of the document [6]. It consists of three technical specifications: the CDA Header, the CDA Level One Body, and the HL7 Version 3 data types. Level Two uses the same codes as Level One for the document type and sections but will allow further constraints to be imposed. Level Three will define observations and services within the document body. At this time, Levels Two and Three have not yet been fully defined by HL7.

As indicated above, CDA documents derive their meaning from the HL7 Reference Information Model (RIM). The elements and attributes and the relationships among these elements and attributes are drawn from the RIM and expressed in XML. For example, Figure 1 illustrates the use of the caption, coded caption and vocabulary domains in the Body of the document instance.

The caption for the section is "Most Responsible Diagnosis". The caption for the paragraph is "Unstable Angina". The RIM code representing Unstable Angina is I20.0 and is taken from the source vocabulary, ICD10 (2.16.840.1.113883.6.3). This permits the caption to be displayed and/or read by a human and the corresponding code to be processed by a computer application.

The CDA Header contains the metadata describing this clinical document. It consists of four logical components:

- Document information, including relationships to other documents
- Encounter data
- Service actors (such as providers)
- Service targets (such as patients)

```
<caption>Most Responsible Diagnosis</caption>
  <section>
    <caption>Unstable Angina
      <caption cdV=I20.0 S="2.16.840.1.113883.6.3"/>
    </caption>
    <paragraph>
      <content>Y</content>
    </paragraph>
  </section>
</caption>
```

Figure 1. Captions, coded captions and vocabulary

The major elements of the clinical_document_header include: id, set_id, version_nbr, document_type_cd, origination_dttm,confidentiality_cd, document_relationship, patient_encounter, legal_authenticator, originator, originating_organization, originating_device, provider, patient, local_header. Many of these elements may have sub-elements. For instance, as illustrated in Figure 2, the patient element may have sub-elements.

```
<patient>
  <patient.type_cd V="PATSBJ"/>
  <person>
    <id EX="12345" RT="2.16.840.1.113883.3.933"/>
    <person_name>
      <nm>
        <GIV V="John"/>
        <FAM V="Doe"/>
      </nm>
      <person_name.type_cd V="L"
          S="2.16.840.1.113883.5.200"/>
    </person_name>
  </person>
  <birth_dttm V="1932-09-24"/>
  <administrative_gender_cd V="M"
      S="2.16.840.1.113883.5.1"/>
</patient>
```

Figure 2. Example of coding sub-elements

The CDA Level One body is comprised of nested containers, including non_xml data, sections, paragraphs,

lists, and tables. The sections may be nested, in that a section may recursively have further sections, and may also contain paragraphs, lists, and tables. The containers may have captions and contents and may be coded. Each container also has confidentiality and origination attributes. This enables the sharing of information to groups with different confidentiality levels.

3. Database Type Selection

Before discussing what type of a storage system is most appropriate for management of CDA documents, some general comments on the business environment and the architecture of the information system are appropriate as they would have critical influence on the type of a DB system that is chosen. We assume that the information system must support two types of business processes, operational (statistical analysis) and also investigative in that querying facilities must be provided.

CDA (XML) documents are generated by various organizations, such as hospitals, clinics, and physicians, and then processed, resulting in the storage of relevant information in a DB – forming operational processing of health (business) documents. Many of the documents would have to be processed with transactional properties that provide for correct processing of concurrent requests and provide reliability in the face of various types of failures that can occur in such an environment. The DB is also accessed by users that wish to query the DB using DB facilities in order to find information of their interest. If a relational or an object-relational DB is used then the DB is accessed using SQL.

So far no published research has compared various storage strategy to determine which is the most suitable for CDA documents. One of the reasons, perhaps, is that the answer may be derived in a relatively straight-forward manner that we shall adopt here. There are two general alternatives for storing and manipulation of CDA documents: either utilize a file system together with a set of customized programs or use a DB system or use some type of a DBMS. Storing CDA documents in a file system can be viewed as relatively straightforward if all that is required is storage and retrieval while using the document ID as the key. However, such a system is not likely to be under serious consideration in any realistic scenario as it does not provide flexible query and data manipulation facilities and suffers from lack of the tools and features that are provided by database systems, features such as concurrency control and recovery management, and tools such as those for producing reports, forms, and DB statistics.

DB systems that can be considered for management of CDA documents can be categorized by the data model on which they are based and we shall briefly consider the following that we assert, without proof, that they apply here: native XML DBs, true Object-Oriented DBMS Systems (OODBMS), Relational DBMSs (RDBMS), and Object-Relational DBMS. We shall discuss briefly each one together with their strengths and weaknesses in their role in supporting management of CDA documents.

A native XML database is built specifically for storing XML data, supporting the DOM model and declarative querying. At first, this appears to be ideal for the application at hand. However, there are a number of factors that render this option as unlikely in the near future, and in the authors' opinion, in the distant future as well. That such systems can be ruled out in the near future is supported by the fact that currently no commercial-graded systems are available and also the fact that when such systems do become available it will take time for developers to become familiar and comfortable with them. In the long term, their viability to replace relational DB for general purpose business processing is also doubtful because of the hierarchical nature of the XML documents. It should be remembered that hierarchical DBMSs had been around for some time before relational DBs came about; yet, relational DBs are dominant today. Another big and simple reason is that business organizations have invested greatly in the relational DB technology. Thus, in our opinion, native XML DBs will not replace relational DBs in most business type settings, of which our environment is a small example. Yet, we do believe native XML DBs will play an important role in scenarios where their use is advantageous. For instance in our scenario, the CDA documents would likely be archived for auditing and also, perhaps, for data mining purposes. There might very well be an additional, native XML DB used to archive the CDA documents.

When CDA documents are mapped into a relational database, the native hierarchical structure of the XML encoding and the hierarchical structure of the CDA may be lost. This may be fine for operational purposes but not necessarily for the purposes of an audit trail. Furthermore, CDA documents exhibit a nested and recursive structure in their elements.

Object-relational DBs may be better suited than pure relational DBs because of the object-oriented extensions that support complex data types and also references. More importantly, some CDA documents by their nature will contain information in terms of objects such as graphs and pictures that require support of specialized methods for the purposes of querying. In such situations ORDBMs or OODBMSs are preferred.

To support the required storage and manipulation of complex objects and their methods, a true OODBMS, such as ObjectStore O2, may be used. It is our opinion that ORDMBSs, in comparison to OODBMSs, are preferred because they are based on the proven and well

understood and accepted relational DB technologies. It is likely that object-orientation purists would disagree, but a brief contemplation of the investments in resources, human and otherwise, that were made by companies in relational DB technology and its integration and pervasiveness in information systems should be convincing enough that ORDMBSs is the way to go.

Because we argue that ORDBMS is the DB type of choice to support processing of CDA documents and as they are relatively new, a brief description is in order. ORDBMS are often considered the bridge between RDBMSs and OODBMS (object-oriented DBMS). One of the biggest achievements of ORDBMS is to provide a flexible framework for organizing and manipulating software objects corresponding to real-world phenomenon without losing the advantages of RDBMS.

The ORDBMS is based on the relational model. It uses the same data storage structure approach and also the same data-access approach based on a standard object-oriented version of SQL [2]. ORDBMS introduces object-oriented concepts into the database management system and thus it has additional OO structural features, such as inheritance and polymorphism. Because objects can be stored in tables and objects can have methods defined on them, it embeds logic into the database. In comparison, traditional relational DB systems need to move data out of the database in order to apply the logic unless triggers or stored procedures are applied. These new OO features are made possible through new features implemented in ORDBMS [2]. These include table hierarchies (tables can include nested tables) and a number of new data types and features including structured user-defined types, attributes and behavior, functions and methods, typed hierarchies (single inheritance) and typed tables.

ORDBMS are particularly suitable for storage and manipulation of objects that require manipulations on them for the purposes of retrieval. Examples include objects in computer-aided design and the health field. Because the logic to be applied on the object can be embedded in the DB, the time for movement of objects from/to DB is reduced and hence efficiencies are gained. More fundamentally, development of applications becomes simpler as methods on objects that are natively stored in the DB are available. For example, with ORDBMS, one can ask a question such as "Find all graphs that satisfy a certain property" where the property is expressed in terms of a method execution on the object. In terms of the CDA processing and querying, this is fundamentally important when searching objects representing results of medical examinations, for instance, results of X-rays or various types of scans.

Although we claim that an ORDMS is the clear choice for storage and manipulation of data contained in CDA documents, the design of a DB for the ORDMS can take one of two approaches: in one, the design follows the natural object-oriented approach while in the second a relational DB schema is used as much as possible while objects are used only when necessary. We shall elaborate on these two approaches and compare them in the subsequent section.

4. Database Design and Implementation

We follow a traditional design that is based on the ANSI 3-Tier database model consisting of the conceptual or external level, logical or internal level and a physical level. The goal of the conceptual phase is to produce a conceptual schema for the database that is independent of a specific DBMS. During the logical level design, the conceptual schema is mapped into logical schema using a selected data model, in this case the Object-relational. The last step is the development of the specifications for the stored database in terms of its physical storage structures, low-level algorithms used to perform data retrieval and management.

4.1. ER Model for CDA Documents

For the conceptual model we use the widely used Entity Relationship (ER) model. The ER model for processing of CDA documents is relatively straight-forward consisting of just three entities and a number of relationships. Specifically, there are three entity types: Person, CDA-document, and Organization. There are eight relationships, each one with attributes: four N:M relationships and three 1:N relationships between CDA-document and Person entities and one 1:N relationship between CDA-document and organization entities. The specific relationships between the CDA-document and Person entities are:

- One CDA can be authenticated by 0-N person (authenticator) and one person can authenticate 0-M CDA documents (N:M).
- One CDA can be received by 0-N person (intended recipient) and one person can receive 0-M CDA documents (N:M).
- One CDA can be originated by 0-N person (originator) and one person can originate 0-M CDA documents (N:M).
- One CDA can be provided by 1-N person (provider) and one person can provide 0-M CDA documents (N:M).
- One CDA must have and only have one person (patient) and one person can be the patient for 0-N CDA documents (1:N).
- One CDA may be legally authenticated by 0-1 person (legal authenticator) and one person can legally authenticate 0-N CDA documents (1:N).

- One CDA may have 0-1 transcriptionist and one person can be the transcriptionist for 0-N CDA documents.

There is one 1:N relationship between the CDA-document and Organization entities:

- One CDA may have 0-1 originating organization and one organization could be the originator for 0-N

The ER diagram is shown in Figure 3. Square boxes represent entities, while their attributes are listed inside columns. For instance, the person entity is represented by a column with the two boxes labeled "Person" at the bottom and top of the column. Inside the column are Person attributes Id, Name, Address, and Phone. Attributes are listed using the following notation: "?" means that the attribute could be NULL, "*" means the attribute could have 0 to N values, "+" means the attribute must have at least one value. The diamonds represent relationships in the usual manner also showing the minimum and maximum cardinalities. Finally, relationships also have attributes; these are shown on the right-hand-side of the diagram using untraditional notation. Consider, for instance, the relationship Originator – it has attributes Type_cd and Prticipation_tmr.

Examination of the ER diagram and, in particular, the CDA documents that are supported at the Level One, reveals that a purely relational DB schema would be sufficient to support the CDA documents application in that there are no elements, such as graphs or scans, that require complex types and processing. Thus, if processing of CDA documents meant that only Level One documents are to be supported then a relational DB would be sufficient to support such as application. However, we do know that when eventually Level Two and Three CDA documents are defined that they will contain complex objects that will require support of an ORDBMS.

Because the Level One CDA documents do not require special methods for storage and retrieval, the resulting ER diagram can be translated into ORDMBS data model in two orthogonal ways, one using object orientation while the other using translation of the ER diagram into a pure relational data model. We shall explore these two approaches in order to find their advantages and disadvantages. For the purposes of the rest of the paper, we shall refer to one design approach as object-relational while we shall refer to the other as relational. It should be kept in mind, however, that both designs would eventually be supported by an ORDBMS.

4.2. Object-relational database design

When translating the ER model into an object-relational model [3], a natural way to proceed is to represent entities as objects. Thus there are three objects (classes of objects) corresponding to entities in the ER diagram. The object classes are shown in tables 1 to 3.

Table 1. Object oo_person and its attributes.

Person (oo_person)	
Name	***Type [: type of type]***
Id_list	oo_id_list : VArray (oo_id)
name_list	oo_name_list : VArray (oo_name)
address	oo_address_list : VArray (oo_address)
phone	oo_phone_list : VArray (oo_phone)

Table 1 represents the oo_person object, which has four attributes, id_list's type is oo_id_list which is an array of object oo_id, similarly for name_list, address and phone. Note that the object has three attributes/data-members that are arrays of objects. They are used to represent the multi-valued attributes of the entity Person of the ER model. A simple attribute (non-multi-valued) of an entity would be represented by a data member that is not an array.

Table 2 represents the object oo_organization. The structure of the object is similar to that of the Person object in that each of the object's array is used to represent one of the mutli-valued attributes of the entity Organization.

Table 2. Object oo_organization and its attributes

Organization (oo_organization)	
Name	***Type [: type of type]***
id_list	oo_id_list : Array (oo_id)
name_list	oo_name_list : Array (oo_name)
addr	oo_address_list : Array (varchar2(50))

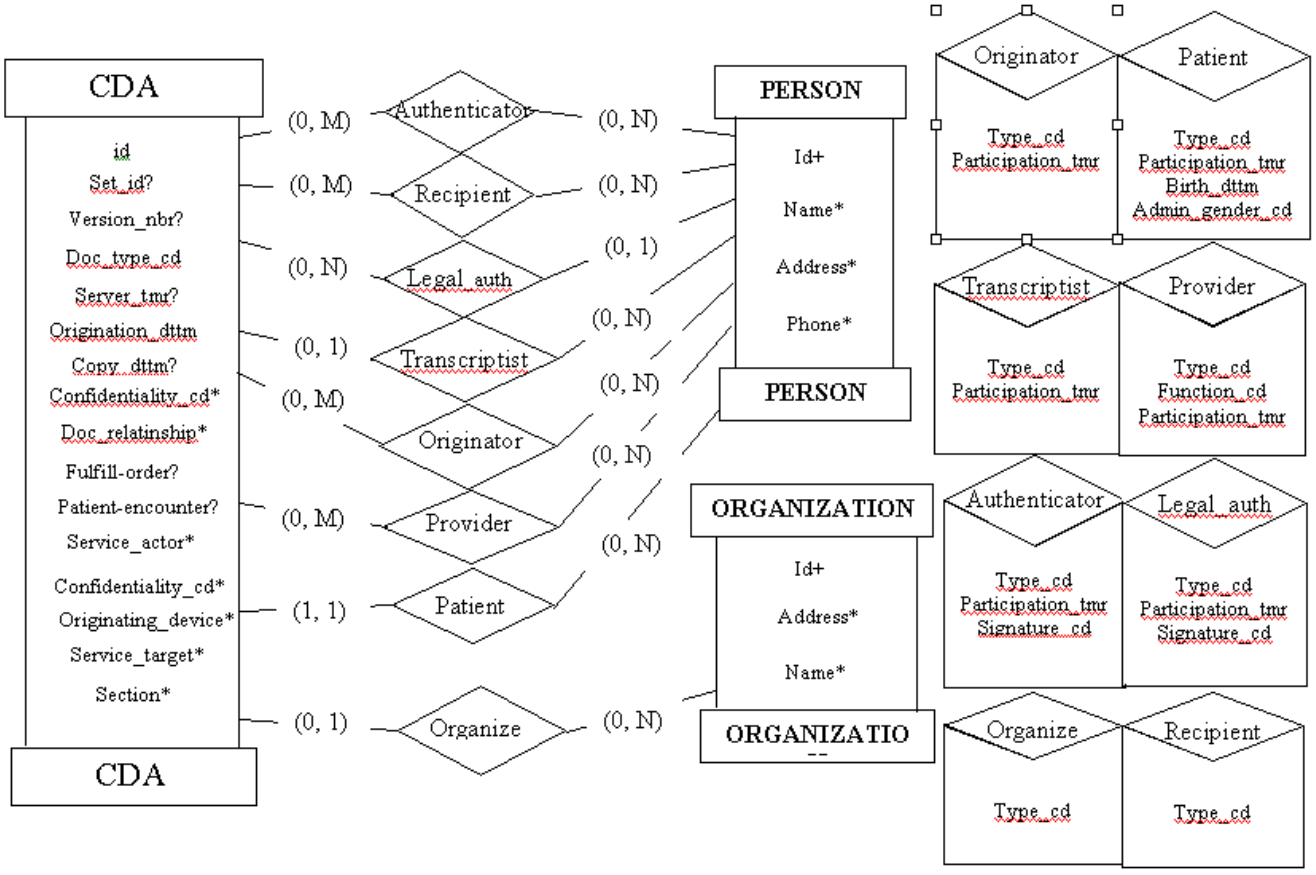

Figure 3. ER diagram for storage of CDA documents

In Table 3, which represents the object oo_cda, the first column is the name of data members/attributes while the second column is their type. Note that the first rows of the table up to and including the *section_list* are used to represent the attributes of the entity CDA-document. The remaining rows are used to represent the relationships, occurring in the ER model, between the CDA-document entity and the Person or Organization entities.

Table 3. Object oo_cda and its attributes.

CDA (oo_cda)	
Name	***Type [: type of type]***
cda_id	oo_id
set_id	oo_id
document_type_cd	oo_type_cd
service_tmr	Char(10)
origination_dttm	Char(10)
copy_dttm	Char(10)
confidentiality_cd_list	oo_confidentiality_cd_list : VArray (oo_confidentiality_cd)
document_relationship_list	oo_relationship_listtable : Nested table (oo_relationship)
fulfill_orders	oo_fulfill_order
patient_encounter	oo_patient_encounter
...	...
section_list	oo_section_listtable : Nested table (oo_section)
authenticator_list	oo_authenticator_listtable : Nested table (oo_authenticator)
intended_recipient_list	oo_recipient_listtable: Nested table (oo_recipient)
originator_list	oo_originator_listtable: Nested table (oo_originator)
originating_organization	oo_originating_organization
Transcriptionist	oo_transcriptionist
provider_list	oo_provider_listtable : Nested table (oo_provider)
service_actor_list	oo_service_actor_listtable : Nested table (oo_service_actor)
Patient	oo_patient_listtable : Nested table (oo_patient)

Attributes with multi-values (in the ER model) could be represented as either VARRAYS or nested tables. In general, if a collection needs to be queried then nested tables are used. If a whole collection needs to be retrieved with one operation, then VARRAY should be used.

In the object-relational model, an N:M relationship can be implemented by nested tables plus REF (a concept similar to a reference pointer). Using the relationship between provider and CDA as an example. Each oo_cda includes a nested table of objects oo_provider. One of the oo_provider's attributes is a reference to a person. Therefore, in the object-relational model, only two tables, cda and person, are enough for an N:M relationship. In the relational model, this case will result in cda, cda_person and person, totalling three tables. In fact, the nested table in oo_cda (oo_provider_list) for object-relational model is equivalent to cda_person table in the relational model.

In the object-relational model, the 1:N relationship implementation is similar to implementation in the relational model, but the object-relational model uses REF type, while the relational model uses foreign keys. For a better understanding, objects oo_provider and oo_patient are also list below (Tables 4 and 5). In Table 4, person_ref is a reference pointing to an oo_person instance. And in object oo_cda, provider_list is a nested table of oo_provider (Table 1). oo_person, oo_provider and oo_cda build a N:M relationship. In Table 5, person is a reference pointing to an oo_person instance. oo_patient is one of the attributes of oo_cda (Table 1), so it implements a 1:N relationship.

Table 4. Object oo_provider and its attributes

Provider (oo_provider)	
Name	***Type [: type of type]***
type_cd	oo_type_cd
function_cd	oo_type_cd
Participation_tmr	Char(10)
person_ref	REF oo_person

Table 5. Object oo_patient and its attributes

Patient (oo_patient)	
Name	***Type [: type of type]***
type_cd	oo_type_cd
Participation_tmr	Char(10)
Person	REF oo_person
birth_dttm	Char(10)
admin_gender_cd	oo_type_cd

We shall elaborate further on design alternatives later when comparing the object-relational and relational options.

4.3. Relational database design

We have used a standard algorithm to translate the ER model into the relational data model. The result of this translation is a set of 28 relations. Basically, the algorithm creates a relation/table for each entity. It also creates a table to represent each multi-valued attribute in the ER model – and there is a large number of them. A 1:N relationship is represented by attaching a key for the table from the one side of the relationship to the table from the many side of the relationship. Finally, each N:M relationship in the ER model is represented by using a separate table. Because the ER model has a large number multi-valued attributes and N:M relationships, the resulting number of tables in the relational model is also large when compared to the object-oriented design.

Because the relational DB design is pervasive in the in building information systems it shall not be elaborated on any further.

5. Comparison of the Object-relational and Relational Approaches

The previous section provided reasons for expecting that the DB of choice to support processing of CDA documents would very likely be of the ORDBMS type. It also showed that there are two approaches to the design of the object-relational DB. One approach uses the natural object relational design in which tables store objects that have data members and methods. In the second approach, tables are used to store objects only when it is necessary in that the processing requirements cannot be met by the pure relational facilities of the DB system but must resort to inclusion of object-specific methods to manipulate the objects.

We compare these two approaches from the perspectives of design and implementation issues and execution "performance".

5.1. Implementation Issues

The object-relational design results in a much smaller number of tables. Also, queries tend to be simpler as they use object methods that simplify expression of queries. On the other hand, object methods must be carefully prepared to support queries. Ad-hoc queries may not be possible for an object-relational design if appropriate methods have not been prepared at the time of implementation.

When designing and implementing an object-relational DB, for each object we must determine the types for the object's data members/attributes, the user-defined (**UDT**) types, and also methods that can be invoked on the objects, i.e., we must determine user-defined functions

(**UDF**) for objects. Thus, for each object, we must examine possible queries/operations on the object and create appropriate methods to support them. For instance, if we wanted to find the list of phones for the patient called Bob Smith, then we would issue the following objected-relational SQL statement:

```
SELECT o.phone  FROM oo_person_tab o
WHERE o.hasName ('Bob', 'Smith');
```

Note that the operation/method *hasName* must be provided as one of the methods on the object oo_person class. The UDFs are necessary to support search in nested tables. Thus, to support add-hoc queries, numerous methods/functions must be prepared a-priori. The programming is not difficult but it must be performed when the DB is implemented. If the method, such as *hasName*, that determines whether a Person class instance has a given name does not exist, we could not issue such a query. But once the method on the object is provided, the query itself is a simple one – retrieval is from one table only.

In pure relational SQL the query would be:

```
SELECT o.person_phone
FROM person p, person_person_name n
    person_person_phone o
WHERE n.person_name = 'Bob Smith'
    AND n.person_id = p.person_id
    AND o.person_id = p.person_id;
```

In this case, because of the flattened structure of the tables, no methods need to be implemented by the programmer, but the query is a join query spanning three tables.

The object-relational design may be viewed as simpler and as more intuitive than a pure relational approach – the result is simpler in terms of the number of tables. There are significant drawbacks, however. There is a smaller number of tables because mutli-valued attributes are represented as arrays or nested tables. However, methods to perform retrieval on these arrays and tables must be supplied through programming. More fundamentally, the smaller number of tables is also due to the fact that N:M relationships are also represented using nested tables possibly with additional objects. The designer, must make a choice in which object should the nested table be located. Consider a N:M relationship between entities A and B. This relationship can be represented by using a nested table in the object representing the entity A (referred to as object A) or by using a nested table in the object representing the entity B (referred to as object B). Appropriate methods to search the table must also be programmed. These two choices are not equivalent/symmetric because there are implications on delays of executing queries. If the table is included in the object A, then queries of the form, "find objects of class B that are related to a given instance of object A", are easily answered. By easily, we mean that they are not only easy to program but also efficient to execute. On the other hand, a query of the form, "given an instance of object B, find all instances of object A it (instance of B) is related to", is not easy to answer in terms of complexity and also efficiency. In the relational model, a relationship is represented by a separate table and hence both queries are equivalent/symmetric in terms of complexity and efficiency.

5.2. Execution Delays

We have implemented a relatively small DB, using both the pure relational and object-relational approaches, and used selected operations to obtain preliminary results in terms of execution delays. Both the relational and object-relational DB contained the same set of 1,000 CDA documents. For details on queries and creation of the CDA databases please see [11].

We shall just mention that storage requirements (in terms of the disk blocks) and also memory requirements when executing queries were much higher for the relational approach when compared to the object-relational approach. But, considering the continuous improvements in capacities and declining costs for both types of storage, they are not considered to be important.

To compare execution delays for the two approaches we have selected a set of queries that we considered to be representative. We chose the queries by examining the HL7 Version 2.4 CD-ROM published by Health Level Seven (received in Feb 2002), which is one of several ANSI-accredited standards developing organizations operating in the healthcare arena. The CD-ROM has nine chapters, most of the chapters provided typical queries for that particular area. We selected queries in the following categories:

- Queries that retrieve data on a single patient while using simple search criteria such as the name of a patient.
- Queries that retrieve data on multiple patients.
- Queries that retrieve information/data on CDA documents (not patients).
- Update operations to insert, delete, and update data stored in the DB.

We have implemented the queries for both the relational and object-relational DBs using Oracle 9i. Each query, such as find all patients with a given name "Bob Smith", has two equivalent version, one for the object-relational DB version and one for the relational version. They were equivalent in that they both retrieved the same result. Every query was run 5 times to get the average values. The differences between different runs were within 20% for all queries except those that had a very small CPU time. Care was taken to ensure that observed execution delays were not affected by

environment in which the experiments were conducted. For instance, it was ensured that no other applications were executing so that they would not affect the total execution delays of queries.

We used the TKPROF tool to find the execution characteristics of the queries. The characteristics provided by the tool include execution time, elapsed time, the number of table-rows accessed, etc. Table 6 shows the query execution delays. There are 13 queries, labeled as Q1 to Q13. Execution delays for the relational DB appear in the second column while the last two columns contain execution delays for two versions of queries for the object-relational case. The difference between the two versions is that in the second version we made sure to use UDF in queries only when it was necessary, that is only for queries that need to access the nested tables. For instance, a Version 1 query may have a clause

"WHERE o.hasCDAID ('a123', '2.16.840.1.113883.3.933') = 1"

that uses a UDF .hasCDAID() defined on an object. Instead of using this UDF the clause can be replaced by an equivalent clause that does not use the UDF:

"WHERE o.cda_id.ex='a123' AND
o.cda_id.rt='2.16.840.1.113883.3.933'"

As the table indicates, we were able to modify queries Q6 to Q10 to avoid the use of UDFs in five of the object-relational queries. It should be noted that queries Q1, Q2, and Q3 do not require the use of UDFs at all and thus they do not appear in the column for Version 1 queries.

Table 6. Execution delays

Query #	Relational	Object Relational	
		Version 1	Version 2
Q1	0.02		0.01
Q2	0.01		0.05
Q3	0.05		0.32
Q4	0.01	0.05	
Q5	0.01	0.02	
Q6	0.01	18.4	0.03
Q7	0.05	19.5	0.3
Q8	0.01	18.3	0.02
Q9	0.01	18.4	0.02
Q10	0.01	18.7	0.04
Q11	0.1	23.5	
Q12	0.05	5.2	
Q13	0.08	37.2	

Q1 to Q3 belong to query category 1, which do not use any UDFs. The execution delays were relatively small and comparable in magnitude.

Query category 2 includes queries Q4, Q5 and Q6 and are expressed on tables oo_person_tab, oo_organization_tab and oo_cda_tab respectively. Using Q4 of query version 1 as an example, this query needs to scan the whole oo_person_tab table, to execute a UDF hasName() for every tuple, and then collect the tuples that satisfied the condition, hasName ('Henry', 'Levin') = 1. The time complexity would be of O(n), where n is the number of tuples in the table. Therefore, execution delays of queries with UDFs such as hasName() and hasID() are dependent on the number of tuples that must be searched. Because the number of tuples searched by query Q6 is much larger than the number of tuples searched by queries Q4 and Q5, the execution delay of the query Q6 is also much higher. In the relational version the execution delay of the query Q6 is not sufficiently higher to be recorded using the precision used in table 6. Since relational databases perform sophisticated optimization and exploit fast access paths such as indices, the time complexity is likely to be of O(LOG(n)), where n is the number of tuples. For object-relational database, it is a difficult problem to attempt to optimize access to tuples through UDFs. When the version 1 of the object-relational query Q6 is modified to version 2 by avoiding the use of a UDF, its execution delay is improved dramatically. The same observation can be made about the category 3 of queries that includes Q7, Q8 and Q9.

The final category includes queries that entail retrieval from tables that are involved in N:M relationships. Again, the performance for object-relational database is not as good as relational database.

The average execution delays shown in the table lead to an obvious conclusion. Relational queries are far more efficient to execute. The table also shows that it is the UDFs in the object-relational approach that contribute most to the high execution delay of the object-relational approach. When comparing the two versions of queries for the object-relational DB, whenever the UDFs can be avoided when expressing a query, its execution delay is superior (lower).

For detailed discussion of execution delays please see [11]. We shall conclude that database vendors, and Oracle is no exception, have spent great resources ensuring that the execution of queries is as efficient as possible. Because the relational DB has been around for a long time, execution of SQL queries on relational tables of simple values (that is tables that do not store objects and that do not use nested tables) is very efficient. Experience with optimizing queries on tables that store objects is not extensive because it is a difficult to perform optimization when user-defined methods are involved. Furthermore, when searching through tables of objects, object functions are invoked and the overhead associated with their invocation is relatively high. Finally, object methods are programmed at design-time and they cannot be optimized automatically by the DB system itself. Thus, if they are not programmed with efficiency in mind, their execution delay will affect the overall delay of executing the query.

6. Comparison: Summary and Conclusions

To summarize, from the design point of view, the object-relational approach results in a simpler DB in terms of the number of tables. However, the design process is not as straight-forward as pure relational, because the designers have to be careful of how to represent the relationships among objects. When implementing the DBs, in the object-relational approach, methods to support search of multi-valued attributes and relationships, which are represented using arrays or nested tables, need to be provided. In the relational approach to the design, these are represented through separate tables accessible by SQL and hence programmed methods are not required.

In terms of queries and their execution, because of the smaller number of tables and object methods, queries are simpler to express in the object-relational approach. However, ad-hoc queries are not facilitated by the object-relational approach in situations where object-methods, which are required by the query, are missing. Execution of queries is more efficient in the relational-design DB. The reason is there is excellent query optimization for standard (non-object) SQL and also the fact that there is no invocation of methods on objects and thus the overhead of such invocation is avoided.

We conclude with an opinion that if a prototype were being developed, an object-relational approach would be appropriate as not all object methods would have to be developed and the over-all DB design would be simpler and thus easier to develop. Ad-hoc queries are not of concern under such a scenario either. For an operational DB that is stable in terms of the DB schema, however, a careful relational design would be preferred. Although the DB design is more complex, the schema is stable and supports flexible querying in that any relational SQL query can be issued on the DB and there are no restrictions on the queries depending on whether objects methods to support the search exist or not. Furthermore, execution of equivalent queries is more efficient in the relational as opposed to object-relational DB.

References

[1] L. Alshuler and R. Dolin. (Ed.) *Version 3 Standard: Clinical Document Architecture Release 1.0*. Canada HL7. November 6, 2000.

[2] Paul Brown. *Object-Relational Database Development: A Plumber's Guide.* First Edition. , Prentice Hall PTR. 2000.

[3] Paul Brown. *Developing Object Relational Database Applications*. 2000. [Available May 28, 2002: http://www.iiug.org/ver1/resources/articles.html]

[4] Robin Cover. "Health Level Seven XML Patient Record Architecture". *The XML Cover Pages*. [Available May 28, 2002: http://xml.coverpages.org/hl7PRA.html]

[5] R. Dolin. "Clinical Document Architecture", *e-Health 2001: The Future of Health Care Proceedings.* 26-29 May 2001, Toronto, Ontario. Canada HL7 Meeting.

[6] Robert H. Dolin, Liora Alschuler, Sandy Boyer, Calvin Beebe. "An Update on HL7's XML-based Document Representation Standards", *Proc. of the AMIA 2000 Annual Symposium*, November 4-8, 2000, Los Angelos, California. [Available May 28, 2002:: http://www.amia.org/pubs/symposia/D200113.pdf]

[7] HL7 Mission Statement. [Available May 28, 2002: http://www.hl7.org/about/hl7mission.htm]

[8] Kona Proposal Committee. "The Kona Proposal for Electronic Health Care Records", July 7, 1997. [Available May 28, 2002: http://www.hytime.org/ihc97/papers/harding/kona/kona.html]

[9] Oracle8 application Developer's Guide - Chapter 7. User-defined datatypes - an extended example. [Available May 28, 2002: http://www.oracle.com]

[10] Paterson, G., Shepherd, M., Wang., X., Watters, C. and D. Zitner. "Using the XML-based Clinical Document Architecture for Exchange of Structured Discharge Summaries". 35th *Hawaii International Conference on System Sciences*, Hawaii, 7-10 January 2002. CD-ROM publication.

[11] Zheng Liang, "Storage Models for CDA Documents, M.Comp.Sc. thesis, Dalhousie University, Halifax, Nova Scotia, Canada, 2002.

0-7695-1874-5/03 $17.00 © 2002 IEEE