GLOBAL JOURNAL OF PURE AND APPLIED SCIENCES VOL. 11, NO. 4, SEPTEMBER 2005: 561 - 565
COPYRIGHT(C) BACHUDO SCIENCE CO. LTD. PRINTED IN NIGERIA. ISSN 1118- 0579

OBJECT-RELATIONAL DATABASE DESIGN-EXPLOITING OBJECT ORIENTATION AT THE DATABASE LEVEL

BARILEÉ B. BARIDAM and O. OWOLABI

(Received 17 March, 2004; Revision accepted 8 September, 2004)

ABSTRACT

This paper applies the object-relational database paradigm in the design of a Health Management Information System. The class design, mapping of object classes to relational tables, the representation of inheritance hierarchies, and the appropriate database schema are all examined.

KEYWORDS: object relational, database schema, OID, instance variables, health system.

INTRODUCTION

The object paradigm is based on building application out of objects that have both data and behaviour. Relational paradigm is based on storing data on tables. The integration of database capabilities with object programming language capabilities results in an Object-Oriented Database Management System (OODBMS). An OODBMS makes database objects appear as programming language objects in one or more existing programming languages. The OODBMS extends the language with transparently persistent data, concurrency control, data recovery, associative queries, and other database capabilities (ODBMS, 2003).

Object-relational databases augment the concept of relations with object orientation. It is thus a ready way of migrating a relation database into an object-oriented database. There are many advantages to including the definition of operations with the definition of data. First, the defined operations apply ubiquitously and are not dependent on the particular database application running at the moment. Second, the data types can be extended to support complex data such as multi-media and imaging by defining new object classes that have operations to support the new kinds of information (DACS, 1997).

Other strengths of object-oriented modelling are well known (Wirfs-Brocks, et. al. 1990). For example, inheritance allows solutions to complex problems to be developed incrementally by defining new objects in terms of previously defined objects. Polymorphism and dynamic binding allow the definition of operations for one object and then to share the specification of the operations with other objects. These objects can further extend these operations to provide behaviours that are unique to them. Dynamic binding determines, at runtime, which of these operations is actually executed, depending on the class of the object requested to perform the operation. Polymorphism and dynamic binding are powerful object-oriented features that allow the composition of objects to provide solutions without having to write code that is specific to each object. All of these capabilities come together synergistically to provide significant productivity advantages to database application developers.

A significant difference between object-oriented databases and relational databases is that object-oriented databases represent relationships explicitly, supporting both navigational and associative access to information. As the complexity of interrelationships between information within the database increases, the greater the advantages of representing relationships explicitly. Another benefit of using explicit relationships is the improvement in data access performance over relational value-based relationships.

A unique characteristic of objects is that they have an identity that is independent of the state of the object. For example, if one has a car object and we remodel the car and change its appearance – the engine, the transmission, the tires so that it looks entirely different, it would still be recognised as the same object we had originally. Within an object-oriented database, one can always ask the question, is this the same object I had previously, assuming one remembers the object's identity. Object-identity allows objects to be related as well as shared within a distributed computing network.

All of these advantages point to the application of object-oriented databases to information management problems that are characterised by the need to manage: a large number of different data types, a large number of relationships between objects, and objects with complex behaviours.
All of these objects are seen in the information system infrastructure of the National Health Insurance Scheme (NHIS) as an integral part of the hospital management system. Beside healthcare, other application areas where this kind of complexity exists includes engineering, manufacturing, simulations, office automation and large information systems.

In this paper we apply the concepts of object-oriented and object-relational databases in designing a web-based administrative application tailored specifically for the registration process of the National Health Insurance Scheme. The goal is to identify the object classes that are required for the proper implementation of this system, and to map these classes to relational database tables. This system, when implemented, will make it possible for health care providers nationwide to register on-line with the NHIS Abuja Headquarters office. Detailed operational information and financial transactions are also to be allowed a two-way flow between the NHIS office and the providers. This will ensure the effective coordination of the scheme. Comparisons are also made between the relational and object-relational database designs, and object-oriented analysis and design for the system is also outlined.

Under the Formal Sector Social Health Insurance Programme, the National Health Insurance Scheme registers eligible employers, who are allowed to choose any Health Maintenance Organisation (HMO) from an NHIS approved list. These HMO's must in turn have been registered with the NHIS. Employees and their dependents are then registered with the Scheme and issued identity cards, after which they

BARILEE B. BARIDAM, Dept. of Mathematics/Computer Science, University of Port Harcourt, Port Harcourt, Nigeria.
O. OWOLABI, Department of Mathematics/Computer Science, University of Port Harcourt, Port Harcourt, Nigeria.

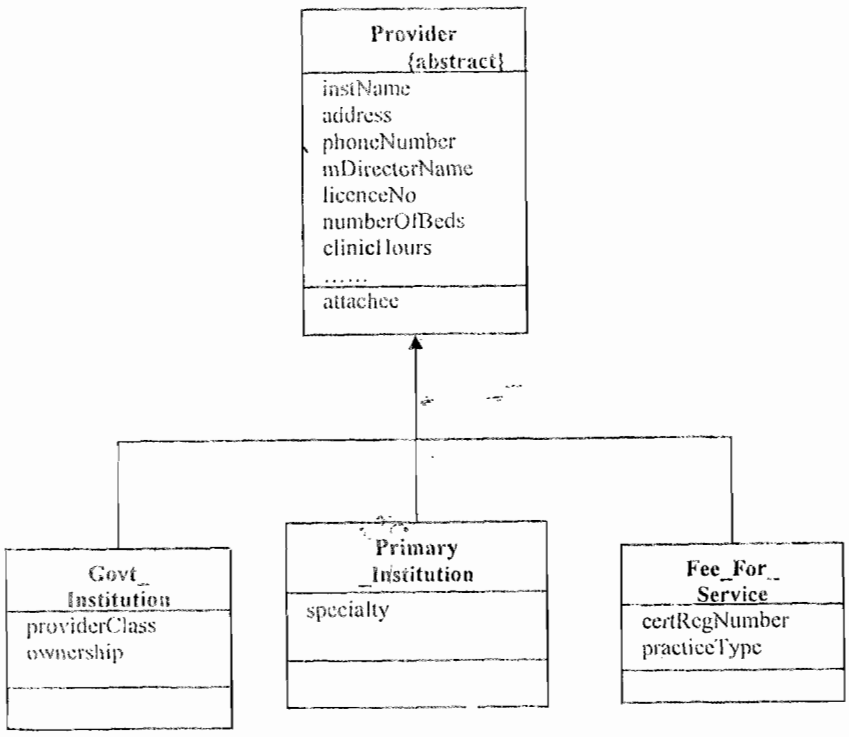

Fig. 1: A Unified Modeling Language (UML) class diagram of the *Provider* class hierarchy.

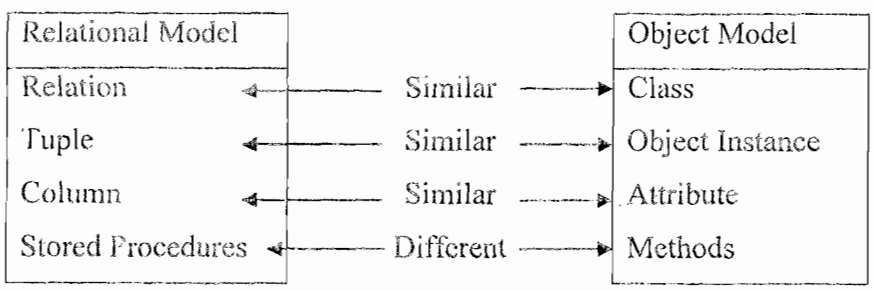

Figure 2: A Diagrammatic Comparison of object and relational terminology.

are required to register with an NHIS-approved Primary Health Care Provider of their choice, who they will consult for all their healthcare needs.

IDENTIFYING OBJECT CLASSES

The NHIS was set up to manage the provision of health facilities nationwide under an insurance scheme system. The health service providers could be classified into: government institutions, primary institutions and fee-for-service providers. These can thus be considered as subclasses of a *Provider* class. The UML class diagram of the class hierarchy is shown in Figure 1.

Another possible class that could be created for this project is the *HMO* class. A class definition lists all the parameters that are needed to define an object of that particular class. The parameters chosen are based on the need and objective of the work (Horton, 2002).

MAPPING OBJECTS TO RELATIONAL DATABASES

For relational databases to have the behaviour of object-oriented Databases there must exist a technique to bind or map these objects to the relational databases. Figure 2 shows the correspondences between the relational and the object model.

Objects are assigned identifiers for easy identification. OIDs are used to uniquely identify objects in a database. The assigning of OIDs is internal. In relational terminology a unique identifier is called a key, whereas in object terminology it is called an OID, although perhaps persistent object identifier would be the better term (Ambler, 2000). Every object in the system is automatically given an identifier that is unique and immutable during the object's life. One object can contain an OID that logically references, or points to, another object. These references prove valuable when associating objects with real-world entities, like in the information system under consideration. They also form the basis of features such as bi-directional relationships, versioning, composite objects, and distribution. In most ODBMSs, the OIDs become physical (the logical identifier is converted to pointers to specific memory addresses) once the data is loaded into memory (cached) for use by the object-oriented application. This conversion of references to memory pointers, sometimes called pointer swizzling, allows access to cached objects at native memory

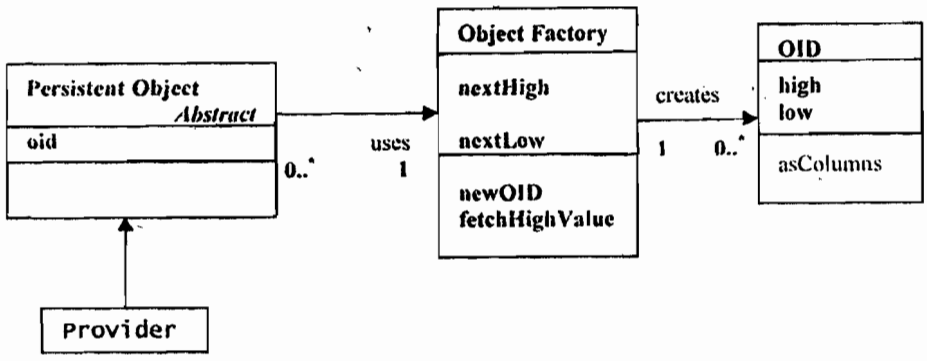

Figure 3: A Class diagram showing a possible way to implement an OID

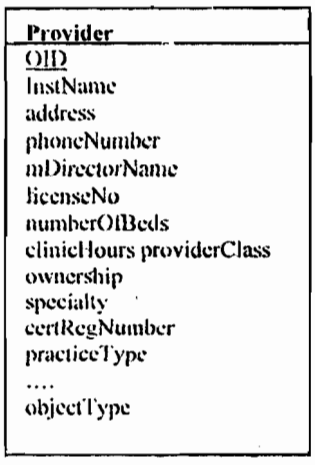

Figure 4: Strategy 1 - One Table Per Hierarchy

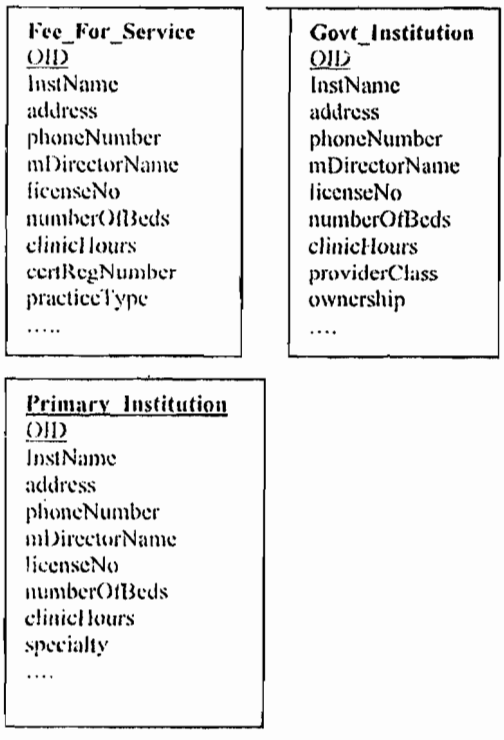

Figure 5: Strategy 2 - One Table Per Concrete Class

speeds (hundredths of microseconds) instead of the traditional approach of messages to the server, which takes milliseconds (McClure, 1997).

Object-identifiers are typically implemented as full-fledged objects in object-oriented applications and as large integers, or several large integers for larger applications in relational schema. The basic idea is that when a persistent object is created it is assigned an OID which is created by the single instance of *ObjectFactory*. This is shown in Figure 3. The sole responsibility of *ObjectFactory* is to create new OID objects. It does this by keeping track of the next HIGH and LOW values, having to fetch a HIGH value from the persistent mechanism (database) occasionally to do so. It creates an instance of OID based on the next values and returns it to be used as the unique OID for the new persistent object. The *asColumns* method returns a collection of data items that can be saved to a relational database to reprsent the instance of OID.

By using OIDs to uniquely identify objects in the database we greatly simplify our strategy for database keys – table columns that uniquely identify records – making it easier to implement inheritance, aggregation, and instance relationships.

MAPPING ATTRIBUTES TO COLUMNS

The attributes of a class will map to zero or more columns in a relational database (Ambler, 1998). Not all attributes are persistent. In this design, some classes have attributes that are used by instances for specific processes but are not saved to the database. Such attributes are regarded as non-persistent. It is worth noting that some attributes of an object are objects in their own right. Sometimes a single object attribute will map to several columns in the database (actually chances are that such a class will map to one or more tables in its own right). The important thing to note is that this is a recursive definition, since at some point the attribute will be mapped to zero or more columns (Ambler, 2000).

MAPPING CLASSES TO TABLES

Classes map to tables, although often not directly. This mapping has to take care of the inheritance hierarchy of the objects. The problem basically boils down to "How do you organize the inherited attributes within the database?" The way in which this question is answered can have a major impact on the system design. There are basically three solutions, shown in Figures 4 to 6, for mapping inheritance into a relational database:

Using one table for an entire class hierarchy, which involves mapping an entire class hierarchy into one table, with all the attributes of all the classes in the hierarchy stored in the table (Figure 4). The main advantage of this approach is its simplicity.

Using one table per concrete class, in which case each table includes both the attributes and the inherited attributes of the class that it

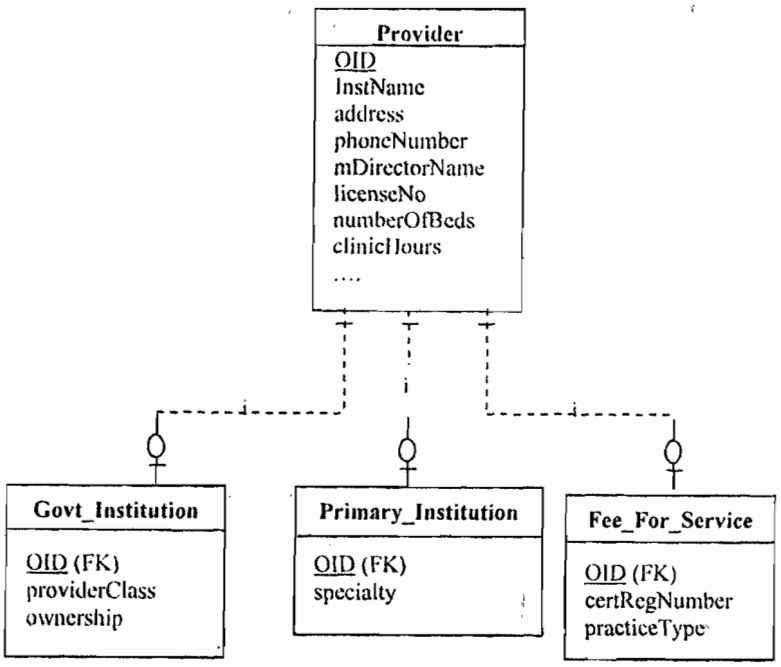

Figure 6: Strategy 3 - One Table Per Class

represents (Figure 5). The main advantage of this approach is that it is still fairly easy to do ad hoc reporting as all the data needed about a single class is stored in only one table.

Using one table per class. This involves creating one table per class, the attributes of which are the OIDs and the attributes that are specific to that class (Figure 6). That main advantage of this approach is that it conforms to object-oriented concepts the best.

Concerning the design trade offs between the three strategies, if another class is added which inherits from, say **Govt_Institution**, very little effort will be needed to update the one table per hierarchy strategy, although the obvious problem of space wastage in the database will increase. With the one table per concrete class strategy we only need to add one table, although the issue of how to handle objects that change their relationships now become more complex. With the third mapping strategy, mapping a single class to a single table, we need to add a new table, one that includes only the new attributes of the added class. The disadvantage of this approach is that it requires several database accesses to work with instances of the new class inheriting from **Govt_Institution.**

The first strategy was chosen in the design of the database. The reason is that it presents a better approach in comparison to the other two as it cuts down on the cost of creating tables. We consequently arrive at the following relational structure for storing persistent objects in the system.

RELATIONAL SCHEMA

Constructing a schema for the database **NhisDbs** we shall look at two tables, namely **Hmo** and **Provider.** The structure of the schema is simply a series of domain entries otherwise referred to as attributes as it applies to each table in the database.

Hmo(*HmoID*, Address, AnnualTurnover, AssetRatio, Branches, CashRatio, CertNumber, Computerisation, Coverage, DateSent, Deposit, Email, Fax, InstName, Insurers, MedName, Objective, PdShareCapital, PhoneNumber, PostalAddress, PrincipalOfficers, RegDate, RegOffice, ShareCapital, ZonalOffice)

Providers(*ProviderID*, Address, Beds, ClinicHours, CurrentLicence, DateSent, Email, Emergency, Employee, ExpiryDate, Facilities, Fax, InstName, IssueDate, Lga, MedicalDirector, MedName, Name, Ownership, PhoneNumber, PostalAddress, PracticeType, ProfQualification, ProfRegNumber, ProfRegYear, ProviderClass, RegAddress, RegNumber, Services, Speciality, State, StateLicence, TradeName)

CONCLUSION

We have been able to explore the potential benefits of the object-relational database model and apply its methods in the design of a relational database schema for a health-sector information system. With the approach of mapping objects to a relational database it becomes easier to develop object applications using a relational database. Programmers can thus enjoy the features offered by object-oriented database design.

REFERENCES

Ambler, S. W., 1998. *Building Object application That Work* SIGS Books/Cambridge University Press.

Ambler, S. W., 2000. Mapping Objexts to Relational Databases October 21, 2000. http://www.AmbySoft.com/mappingObjects.pdf

DACS, 1997. Object-Oriented Database Management Systems Revisited. An Updated DoD Data & Analysis Centre for Software state-of-the-Art Report, December 18, 1997. http://www.dacs.dtic.mil

Horton, I., 2002. Ivor Horton's Beginning Java 2 SDK 1.4 Edition, Wrox Press Ltd.

McClure, S., 1997. Object database vs. Object-Relational Databases. IDC Bulletin #14821E August 1997.

ODBMS, 2003. Object Database Management System *(ODBMS)* definition http://www.odbmfacts.com (June, 2003).

Wirfs-Brock, Wilkerson, R, B. and Weiner, L., 1990. Designing object-oriented software. Prentice-Hall , Englewood Cliffs, NJ