Using UML Class Diagrams for a Comparative Analysis of Relational, Object-Oriented, and Object-Relational Database Mappings

Susan D. Urban and Suzanne W. Dietrich
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
s.urban@asu.edu dietrich@asu.edu

Abstract

This paper illustrates the manner in which UML can be used to study mappings to different types of database systems. After introducing UML through a comparison to the EER model, UML diagrams are used to teach different approaches for mapping conceptual designs to the relational model. As we cover object-oriented and object-relational database systems, different features of UML are used over the same enterprise example to help students understand mapping alternatives for each model. Students are required to compare and contrast the mappings in each model as part of the learning process. For object-oriented and object-relational database systems, we address mappings to the ODMG and SQL99 standards in addition to specific commercial implementations.

Categories & Subject Descriptors

H.2.1 [**Database Management**]: Logical Design – *Data Models, Schema and Subschema.*

General Terms

Design.

Keywords

Entity-Relationship Model, Unified Modeling Language, Relational Model, Object-Oriented Data Model, Object-Relational Data Model, Database Design, Schema Mappings.

1 Introduction

An important topic to address when teaching database concepts to undergraduates is the mapping of conceptual database designs to specific database implementation models. For example, a traditional course on relational database systems typically covers the Entity-Relationship (ER) model and how to use an ER design to generate a relational schema with primary key and foreign key constraints [4]. With recent advances in object-based technology, however, it is important for undergraduates to understand object-oriented techniques for modeling conceptual designs. As a result, it is also important to teach techniques for mapping object-oriented designs to traditional relational technology as well as object-oriented and object-relational database systems.

At Arizona State University, we have developed an advanced database course for undergraduates (CSE 494, http://www.eas.asu.edu/~cse494db) that begins with coverage of object-oriented database modeling using the Enhanced Entity Relationship (EER) model [4] and the Unified Modeling Language (UML) class diagrams [7]. Assuming a prerequisite course on relational database systems (http://www.eas.asu.edu/~cse412), we also cover advanced database topics related to object-oriented database systems [3], object-relational database systems, Web access to databases [2], and professionalism and ethics. The focus of this paper is on our approach to using UML as the basis for a comparative approach to teaching database mapping techniques for relational, object-oriented, and object-relational database designs.

Since students already have a basic understanding of the ER model, CSE 494 first introduces the EER model and its features for modeling inheritance, constraints on inheritance, and categories. We then introduce the equivalent modeling notation in UML, as well as additional features that UML provides for modeling behavior, aggregation, and abstract classes. Because of these additional modeling features in UML that do not exist in the EER model, we focus on the use of UML for a comparative analysis of mappings to different data models. We initially use UML to illustrate different mapping options for relational designs. As we introduce object-oriented and object-relational database technology, we use different features of UML over the same enterprise example to illustrate the mapping techniques that are specific to each model. This approach provides a basis for comparison of the relational, object-oriented, and object-relational technologies in terms of features, design issues, and constraint enforcement.

In the remainder of this paper, Section 2 provides an overview of the School Database Enterprise that is used as a running example to illustrate mapping techniques. The example is introduced using the EER model. Section 3 then describes the equivalent UML notation and the manner in which we teach conceptual mapping to the relational model. Sections 4 and 5 address mapping issues for the object-oriented model and the object-relational model, respectively. Section 6 concludes the paper with a discussion of the advantages of our approach.

2 The School Database Enterprise

Figure 1 gives the EER diagram of the School Database Enterprise. In an EER diagram, rectangles represent classes, diamonds denote relationships between classes, and ovals are attributes, which are linked by edges to the class or relationship that they describe. Edges are also used to denote the participation of the classes in a relationship. A double edge indicates that an instance of the class must participate in that relationship. The numbers 1, M or N on the edges indicate the cardinality or number of times that an instance of the class may participate in the relationship. A circle represents a specialization/generalization relationship between classes. A circle annotated with a 'd' indicates a disjoint specialization – an instance of the superclass can not be an instance of more than one of its subclasses. An 'o' annotation with the specialization circle denotes a possibly overlapping specialization.

In Figure 1, the Student and Faculty classes are generalized into a Person class. The specialization of Person into the Student and Faculty subclasses is disjoint, meaning that a person in this enterprise is either a student or a faculty but not both. The double edge from Person to the specialization circle indicates that every Person must participate in the specialization. Therefore, a Person in the enterprise must be either a student or a faculty member. The Student and Faculty classes inherit the properties from Person and have additional attributes of their own.

A Student has a status attribute, indicating whether the Student is a freshman, sophomore, junior, or senior. A Student also participates in two relationships: major and clubs. A Student must have exactly one major and a Department has many students that declare that department as a major. A Student may be a member of several Campus Clubs. A CampusClub has a unique identifier (cID) and is further described by its name, location and phone. A Campus Club has at most one Faculty advisor.

A Faculty has a rank attribute and participates in three relationships: advises, worksIn, and chair. A Faculty may be the faculty advisor for many CampusClubs but may not be the advisor of any club. A Faculty must be associated with the Department in which the Faculty works. A Faculty is associated with exactly one Department, and a Department has many Faculty working in that department. A Department is simply described with a unique code and name. There is at most one chair of a Department.

3 The Relational View

Figure 2 presents the UML version of the School Database Enterprise. In UML, classes are represented as rectangles with a name and a list of attributes. A class may also have a list of operations that define the behavior of the class.

Subclasses such as Student and Faculty are connected by a line that points to the Person superclass with an open arrowhead.

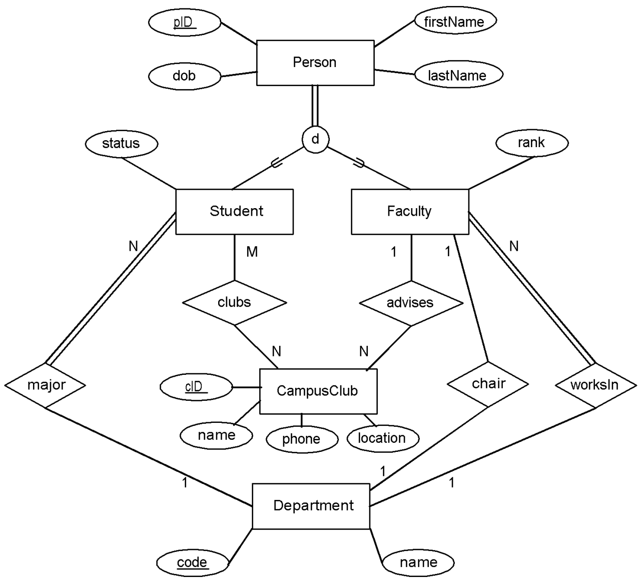

Figure 0: EER Diagram of the School Database

Specialization constraints are enclosed in curly braces. In Figure 2, the constraints on the specialization of Person into its subclasses Student and Faculty indicate that the specialization is disjoint and mandatory (i.e., participation in the subclasses is required).

Relationships in UML, which are referred to as associations, are drawn as lines between classes. Lines can be enhanced with relationship names, role names, and multiplicities. In Figure 2, clubs is a relationship name, with the black arrow indicating the direction in which the relationship is read. Role names provide additional semantics to the association. For example, members represents the role of students when traversing the clubs association from CampusClub to Student. Multiplicities are the same as cardinalities in the EER model. A star (*) represents the many side of a 1:N or M:N association. The number 1 indicates the one side of a total 1:1 or 1:N association. The notation 0..1 denotes partial participation in the association.

An association in UML can be refined by placing an arrow at one end of an association line. The use of an arrow is referred to as navigation and represents a uni-directional association, indicating that the association can only be traversed in the direction shown by the arrow. By default, an association without an arrow is bi-directional.

Figure 2 provides a UML diagram that represents one approach to implementing the School Database Enterprise in the relational data model, where the dog-eared rectangles are notes that summarize the mapping techniques used to design the corresponding relational schema shown in Figure 3. Relations are created for the main classes of Person, Student, Faculty, Department and CampusClub. The bi-directional clubs association is mapped to the separate Clubs table, allowing a user to traverse the association in either direction through queries over the Clubs relation.

The remaining associations (advises, chair, worksIn, majorsIn) are uni-directional, illustrating the mapping technique of embedding the primary key of one relation into the other relation of the association as a foreign key. For example, the majorsIn association is implemented by embedding the primary key of Department into

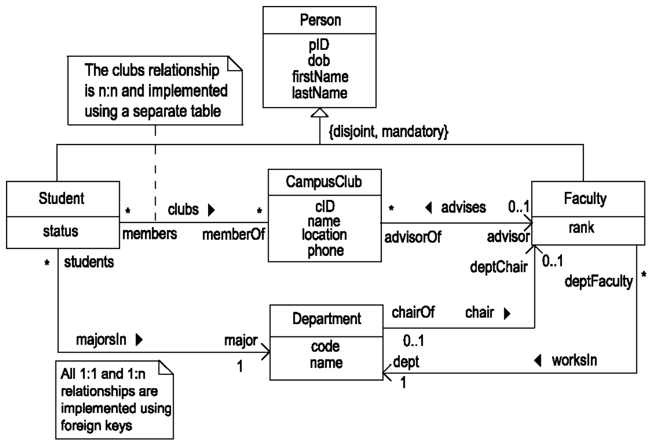

Figure 2: UML for Relational Implementation

Person (pID, dob, firstName, lastName)
Student (pID, status, major)
  foreign key (pID) references Person(pID),
  foreign key(major) references Department(code)
Faculty (pID, rank, dept)
  foreign key(pID) references Person(pID),
  foreign key(dept) references Department(code)
Department (code, name, chair)
  foreign key (chair) references Faculty(pID)
CampusClub (cID, name, phone, location, advisor)
  foreign key(advisor) references Faculty(pID)
Clubs(pID, cID)
  foreign key(pID) references Student(pID),
  foreign key(cID) references CampusClub(cID)

Figure 3: Relational Schema of School Database

Student as a foreign key (i.e., the major attribute). This mapping rule emphasizes the concept of a uni-directional mapping, illustrating how a user can directly navigate from Student to Department using the major attribute. In the opposite direction from Department to Student, navigation is not directly supported and must be indirectly achieved through a query over the Student relation. As a student exercise, the use of navigation can be removed from Figure 2 for the 1:1 and 1:M relationships so that students can experiment with different relational mappings. Removing navigation from majorsIn in Figure 2, for example, would require that the association be modeled as a separate relation.

Mapping techniques also address support for class hierarchies. The approach shown in Figure 3 creates a separate relation for each superclass and subclass, with the subclasses containing the key of the superclass. Views can then be created for Student and Faculty that join each relation with Person to access inherited attributes. Other approaches include flattening the hierarchy into one relation and creating relations for the subclasses only [4]. The constraints of the hierarchy must be considered when choosing the most appropriate mapping.

4 The Object-Oriented View

The coverage of object-oriented databases in the advanced database concepts class includes the Object Data Management Group (ODMG) [1] standard using the Object Definition Language (ODL) to specify object-oriented schemas. The students also have a hands-on assignment using the Objectivity/DB [6] object-oriented database product to reinforce the theoretical concepts.

Figure 4 provides a UML diagram that represents one approach to implementing the School Database Enterprise in an object-oriented data model. Specifically, the UML diagram illustrates an implementation of the application in Objectivity/DB. The choices of implementation for the associations are based on providing students with illustrations of the various alternatives that are available in mapping a conceptual model to an OODB.

Figure 5 provides an ODL specification for the UML diagram of Figure 4. The clubs and advises associations are modeled as bi-directional relationships, which are inherently supported in an OODB. For example, the memberOf relationship in Student represents the clubs association and its inverse is the members relationship in CampusClub. The relationship and its inverse are explicitly defined, and the consistency of the relationship instance is automatically maintained by the database system. A modification to one side of the relationship results in the automatic maintenance of the data on the other side of the relationship. The chair association illustrates a uni-directional relationship, which is specified as the deptChair attribute in Department. There is a method getChairOf() associated with a Faculty member to derive the chairOf role of the chair association. The majorsIn and worksIn associations are bi-directional in the UML diagram but they are not mapped to relationships in the ODL specification. Instead, these associations are represented as attributes on both sides of the associations. For example, the major of a Student is stored as the major attribute whose type is a single instance of the Department class. The students attribute of Department is a collection of Student. This alternative illustrates

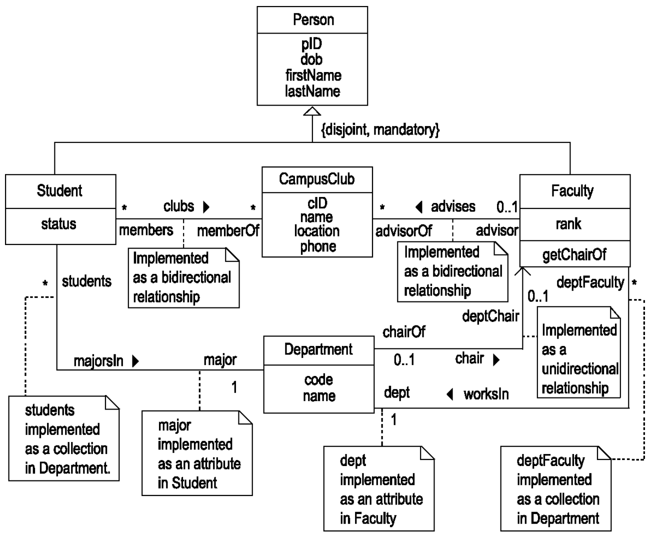
Person
pID
dob
firstName
lastName
{disjoint, mandatory}
Student
status
CampusClub
cID
name
location
phone
Faculty
rank
getChairOf
clubs
members
memberOf
advises
advisorOf
advisor
Implemented as a bidirectional relationship
Implemented as a bidirectional relationship
students
deptFaculty
deptChair
chairOf
chair
Implemented as a unidirectional relationship
Department
code
name
majorsIn
major
dept
worksIn
students implemented as a collection in Department.
major implemented as an attribute in Student
dept implemented as an attribute in Faculty
deptFaculty implemented as a collection in Department

Figure 4: UML for Object-Oriented Implementation

```
class Person
(   extent people
    key pID)
{   attribute string pID;
    attribute date dob;
    attribute string firstName;
    attribute string lastName;
    . . . }
class Student extends Person
(   extent students)
{   attribute string status;
    attribute Department major;
    relationship set<CampusClub> memberOf inverse CampusClub::members;
    . . . }
class Faculty extends Person
(   extent facultyMembers)
{   attribute string rank;
    attribute Department dept;
    relationship set<CampusClub> advisorOf inverse CampusClub::advisor;
    Department getChairOf();
    . . . }
class CampusClub
(   extent      campusClubs
    key cID)
{   attribute string cID;
    attribute string name;
    attribute string location;
    attribute string phone;
    relationship set<Student> members inverse Student::memberOf;
    relationship Faculty advisor inverse Faculty::advisorOf;
    . . . }
class Department
(   extent      departments
    key code)
{   attribute string code;
    attribute string name;
    attribute Faculty deptChair;
    attribute set<Student> students;
    attribute set<Faculty> deptFaculty;
    . . . }
```

Figure 5: ODL Specification of the School Database

that the many side of a 1:N or M:N relationship can be represented explicitly in an OODB, which has the ability to store collections. Using two attributes to store an association also illustrates the advantages of the automatically maintained relationship feature provided by an OODB. The example implementation provided to the student illustrates that the application programmer must maintain the consistency of an association that is not stored as an explicit binary relationship.

5 The Object-Relational View

In the object-relational section of the course, we begin with coverage of the object extensions that have been incorporated into the SQL99 standard [5] as well as advanced features of SQL99 such as triggers and stored procedures. We then provide a case study of how the School Database Enterprise can be mapped to the object-relational model of Oracle 8i. Figure 6 presents the SQL99 schema, demonstrating the object-relational features that must be considered in the mapping process. The corresponding UML diagram is shown in Figure 7. The notes in Figure 7 describe implementation details that are specific to Oracle 8i [8].

The object-relational features of SQL99 include the use of object tables, references between object tables to represent object relationships, and the use of arrays to represent multi-valued

```
CREATE TYPE person_udt AS (
  pID          VARCHAR(11),
  dob          DATE,
  firstName    VARCHAR(20),
  lastName     VARCHAR(20))
  NOT FINAL
  REF IS SYSTEM GENERATED;
CREATE TABLE person OF person_udt (
  CONSTRAINT person_pk PRIMARY KEY(PID),
  REF IS oid SYSTEM GENERATED);
CREATE TYPE faculty_udt UNDER person_udt AS (
  rank         VARCHAR(20),
  advisorOf    REF(campusClub_udt) SCOPE campusClub ARRAY[20],
  worksIn      REF(department_udt) SCOPE department,
  chairOf      REF(department_udt) SCOPE department)
  NOT FINAL
  METHOD    getClubsAdvised() RETURNS VARCHAR(25) ARRAY[20];
CREATE TABLE faculty OF faculty_udt under person;
CREATE TYPE student_udt UNDER person_udt AS (
  status       VARCHAR(20),
  clubs        REF(campusClub_udt) SCOPE campusClub ARRAY[20],
  major        REF(department_udt) SCOPE department)
  NOT FINAL
  METHOD    getClubs() RETURNS VARCHAR(25) ARRAY[20];
CREATE TABLE student OF student_udt under person;
CREATE TYPE campusClub_udt AS (
  cID          VARCHAR(11),
  name         VARCHAR(25),
  location     VARCHAR(25),
  phone        VARCHAR(25),
  advisor      REF( faculty_udt) SCOPE faculty,
  members      REF(student_udt) SCOPE student ARRAY[100])
  NOT FINAL
  REF IS SYSTEM GENERATED;
CREATE TABLE campusClub OF campusClub_udt (
  CONSTRAINT campusClub_pk PRIMARY KEY(cID),
  REF IS oid SYSTEM GENERATED);
CREATE TYPE department_udt AS (
   code         VARCHAR(3),
   name         VARCHAR(40),
   deptChair    REF(faculty_udt) SCOPE faculty)
   NOT FINAL
  REF IS SYSTEM GENERATED
   METHOD  getStudents() RETURNS  VARCHAR(40) ARRAY[1000],
   METHOD  getFaculty() RETURNS VARCHAR(40) ARRAY[50];
CREATE TABLE department OF department_udt (
  CONSTRAINT department_pk PRIMARY KEY(code),
  REF IS oid SYSTEM GENERATED);
```

Figure 6: SQL99 Schema for the School Database

associations. Object tables are created by first creating an object type, such as person_udt in Figure 6. Object types are user-defined types that establish the attributes, object relationships, and methods of a class. A type such as person_udt is then used to create the person table. Instances of the person table will have object identifiers as in the object-oriented model. Object types can be formed into hierarchies that support inheritance. In the faculty_udt type, the phrase "UNDER person_udt" defines faculty_udt to be a subtype of person_udt. The corresponding faculty object table will then be a subclass of the person object table, inheriting the attributes, relationships and methods from person.

References between objects are called REFs in SQL99. For example, the campusClub table contains a REF to objects of type faculty_udt to implement the advisor of a club. In the inverse direction, an array of REFs is used in faculty_udt to store the clubs that a faculty member advises. Since the array stores REFs to club objects, the getClubsAdvised method is used to return the names of

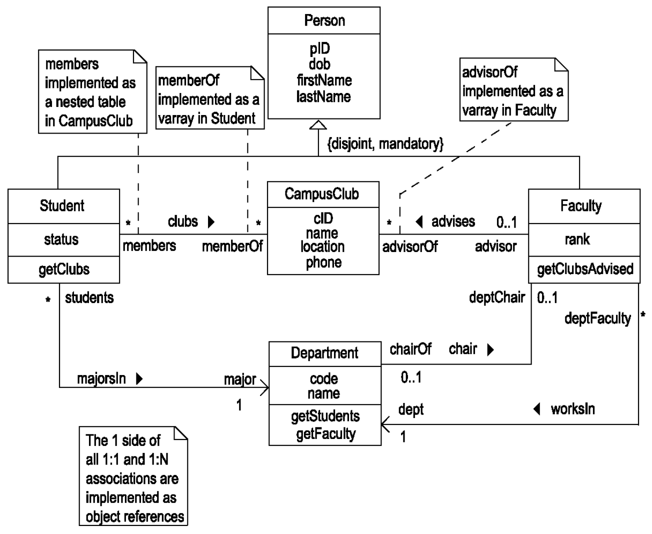

Figure 7: UML for Object-Relational Implementation

the club objects that are referenced in the array. The getClubs method in the student_udt is used for a similar purpose.

Although not shown in this paper, we teach the use of triggers for maintaining inverses in object-relational technology and compare this manual approach to maintaining inverses to the automatic approach provided by object-oriented databases. Also notice that in Figure 7, the majorsIn and worksIn relationships are uni-directional. As a result, methods are added to the department_udt to provide stored procedures that use queries over object tables to calculate the inverse of each relationship.

After students understand mapping to the object-relational features of SQL99, we then present a case study together with an implementation assignment using Oracle 8i. Oracle 8i does not directly support inheritance using the UNDER clause as in SQL99, so students must apply techniques that they learned from relational mappings to simulate inheritance. Oracle 8i also provides varrays (i.e, fixed-sized arrays) and nested tables (i.e., variable-sized collections) as alternatives for the implementation of arrays from SQL99. Figure 7 points out the way in which we use these features in the School Database Enterprise.

6 Summary

This paper has presented an approach for using UML as the basis for a comparative analysis of mappings to relational, object-oriented, and object-relational database designs. We have used this approach with success in three different offerings of the course. Students learn the intricacies of UML as a modeling alternative to the EER model. They also learn how the navigation feature of UML can be used to communicate implementation directives to the mapping process. Students experience the advantages of automatic maintenance of relationships in an object-oriented database model, compared to implementation alternatives in the relational and object-relational models that require the development of code to maintain inverse relationships. Students also address the mapping techniques from the point of view of the ODMG and SQL99 standards as well as specific commercial implementations of each standard.

We are currently updating the object-relational case study to the use of Oracle 9i. We are also revising the material to provide greater emphasis on the role that constraints play in the mapping and implementation process and the differences that exist between the enforcement of constraints in each model.

Acknowledgments

We would like to thank the students that supported this work: Yang Xiao, Nilan Yang, Mei Zheng, Arleen Wiryo, and Pablo Tapio. This research was supported by NSF Grant No. DUE-9980417.

References

[1] Cattell, R. G. G. *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann, 2000.

[2] Dietrich, S. W. and Urban, S. D., and Kyriakides, I., "JDBC Demonstration Courseware Using Servlets and Java Server Pages," *ACM SIGCSE Conference*, Kentucky, Feb. 2002, pp. 266-270.

[3] Dietrich, S. W., Suceava, D., Cherekuri, C., and Urban, S. D., "A Reusable Graphical Interface for Manipulating Object-Oriented Databases Using Java and XML," *ACM SIGCSE Conference*, North Carolina, Feb. 2001, pp. 362-366.

[4] Elmasri, R. and Navathe, S. *Fundamentals of Database Systems*, 3rd ed., Addison Wesley, 2000.

[5] Gulutzan, P. and Pelzer, T., SQL-99 Complete: Really!, R&D Books, Lawrence, Kansas, 1999.

[6] Objectivity/DB Version 7, Objectivity, Inc., Mountain View, CA.

[7] Rumbaugh, J., Jacobson, I., and G. Booch, *The Unified Modeling Language Reference Manual*, Addison Wesley, Upper Saddle River, New Jersey, 1999.

[8] Sunderraman, R., *Oracle 8 Programming: A Primer*, Addison Wesley, 2000.