Use-Case-Based Generation of Forms from a Class Diagram

by

Mohan Choodamani

A research paper submitted in partial fulfillment of the degree of
Master of Science

Major Professor	-	Dr. Toshimi Minoura
Minor Professor	-	Dr. Gregg Rothermel
Committee Member	-	Dr. Margaret Burnett

Dept. Of Computer Science
Oregon State University
Corvallis
Oregon - 97331

October 6, 1999

Acknowledgements

I am deeply indebted to my major professor, Dr. Toshimi Minoura, for giving me an opportunity to work on this project. I am grateful to him for all his valuable guidance, encouragement, and suggestions.

I would like to thank my minor professor, Dr. Gregg Rothermel, for his suggestions in my report. I also thank Dr. Margaret Burnett for accepting to serve in my committee.

I am thankful to my family, for their support and encouragement during my graduate program. I would like to thank all my friends, for their help during my graduate studies.

Use-Case-Based Generation of Forms from a Class Diagram

Mohan Choodamani
Dept. of Computer Science
Oregon State University
Corvallis, OR 97331
`choodamo@cs.orst.edu`

Abstract

Forms are an easy-to-use interface to access a database, including a remote database on the Internet. *An entity-relationship (ER) diagram,* which is a pictorial representation of a database schema, is widely used in designing a database. *A class diagram,* which shows set of classes, relationships among them, and associations, are equivalent to an ER diagram. We present a methodology for generating forms from a class diagram. An unique aspect of our approach is that we use a use-case to specify the information to be accessed by a form. To investigate the applicability of our approach, we developed a Web interface by using Java Servlets.

Key Words and Phrases: Forms, use case, class diagram, java servlet.

Contents

1 Introduction

The ER approach was first proposed by Peter Chen in 1976. Since then it has been extensively used in designing schemas for database systems and to represent the structures of systems in systems analysis. An ER diagram represents the logical structure of a database in a pictorial manner. An ER diagram not only provides an intuitive view of an application, but it is also possible to generate a relational schema automatically from it. The ER approach has been very effective in the areas of data management and systems analysis.

A *class Diagram* is a notation used in *Unified Modeling Language (UML)*[1]. A class diagram shows a set of classes and associations among them. Since a class diagram is more compact than an ER diagram, we consider a class diagram in place of an ER diagram.

A *Use Case Diagram*, a notation used in *Unified Modeling Language (UML)*[1], shows a set of use cases and actors and their relationships. A use case is a description of set of sequences of actions a system perform to yield an observable result to an user.

Data stored in a database are mostly retrieved or updated through *forms*. If forms are not used, then users must learn a query language to access data. As forms constitute a simple interface, they are widely used. In database applications, each use case can be implemented by a form.

A product *Open Database Internet Connector (ODBiC)* allows SQL statements to be added to *HTML forms* to access a database. Currently, many companies, like *Microsoft and Sybase* are doing research on automatic form generation. There are a few products such as *Microsoft Access 2000* available on the market today to generate forms for a database. These products generate forms that can communicate with the database, even across the Internet. However, these products does not specify that these forms are built automatically from an ER diagram or a class diagram.

In this paper, we discuss a new approach for generating forms from a class diagram and use cases. The unique feature of our approach is to use a use-case to specify the information to be accessed by a form. We analyzed the different relationship types that exist among entity types, entity types that can be covered in a form, and the entities that may be covered in one form. Based on the analysis, we formulate a set of rules that needs to be followed to generate forms. We also discuss a method for generating HTML forms with Java servlets.

Section 2 covers the overview of our approach. In Section 3 our methodology for generating forms is discussed. Section 4 gives the implementation details for the e-order company, which is stored in an *Oracle database*. In this section, we also describe a method used to generate forms using *Java Servlets*. Section 5 concludes the paper and addresses some future research topics.

2 Overview of our approach

To describe our approach, we are using a simple e-order company database. In this section we show the ER diagram for an e-order company, and also develop an equivalent class diagram for it. Use cases, as given by UML Notation [5], are used to specify what information can be accessed by each form. In section 2.4, we specify our organization of a form for displaying the required information.

2.1 ER Diagram for an E-Order Company

An ER diagram pictorially shows how a database schema is organized. In an ER diagram the emphasis is on representing the schemas rather than the instances [3]. This is more useful because a database schema does not change, and is easier to display.

A small e-order company must maintain the following information:

1. The company must keep track of all its customers with their names, addresses, and the dates of their first orders. A unique customer number is assigned to each customer.
2. Each order placed by a customer may contain multiple orderlines. Each orderline is used to order one kind of product for some quantity. The date of the order must be recorded.
3. Each product has a product number, a product name, color and a unit retail price.
4. Each product may have multiple suppliers. A unique supplier number is assigned to each supplier. Different suppliers may offer the same product at different wholesale prices. A supplier may supply multiple products. The name, address and status of each supplier must be recorded.
5. The company owns multiple warehouses. Each warehouse is identified by the phone number of its warehouse. The city where the warehouse is located must also be recorded. The number of each product stocked at each warehouse must be recorded. A warehouse can stock different products, however each product is stocked at most at one warehouse.

An ER diagram for an e-order company is shown in Fig. 1. Entity-Types such as `CUSTOMER`, `ORDER`, and `PRODUCT` are shown in *rectangular boxes*. Relationship-Types such as `CONTAINS` and `PLACES` are shown in *diamond-shaped boxes* attached to participating entity-types.

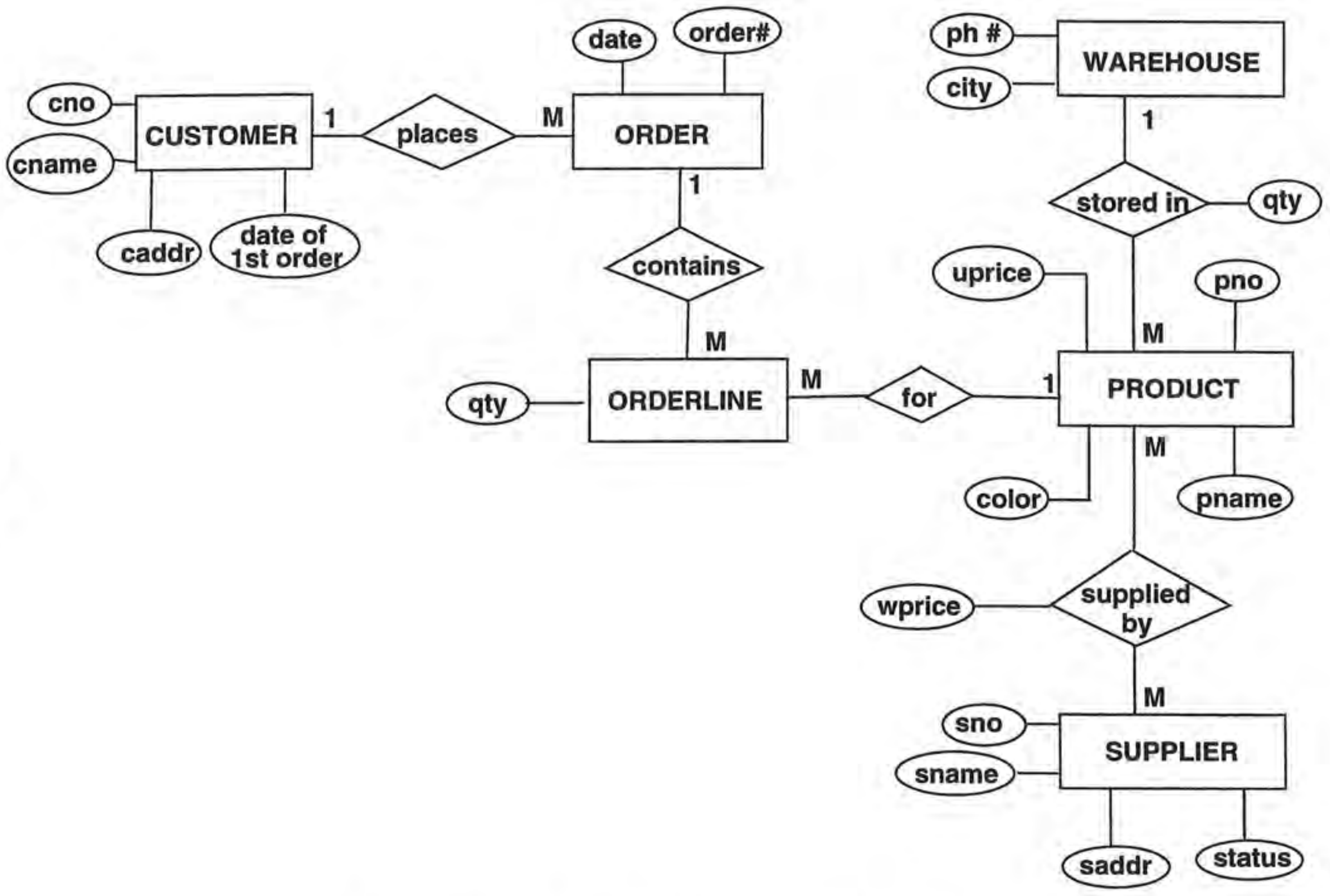


Figure 1: ER diagram for an e-order company.

2.2 Overview of a Class Diagram

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams address the static design view of a system [1]. The class diagram for the e-order company is shown in Fig. 2. The class diagram conforms to the notations specified in the UML notation guide [5].

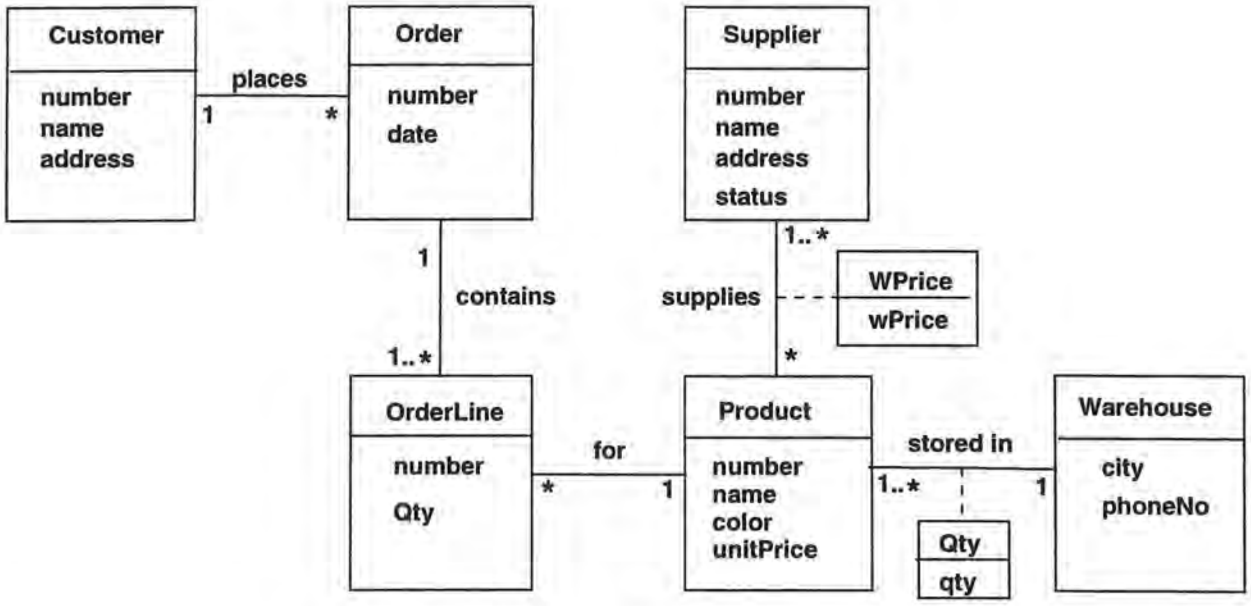


Figure 2: Class diagram for an e-order company.

2.3 Overview of a Use-Case Diagram

A use case is a unit of functionality as seen by an user. The users participating in a use case are called *actors*. *A use case diagram* shows a set of use cases and actors and the relationship among them. A use case diagram addresses the static view of a system. A use case can *use* other use cases and it may be *extended* to additional use cases to perform conditional operations [1]. The use case diagram conforms to the notations specified in the UML notation guide [5]. The following sections discuss use cases for the e-order database system.

2.3.1 Access Customer use Case

The customer can place a new order by entering the information needed for a `customer` and also the order number and order date for that `order`. The customer can delete his information using his `number`. The customer can also delete/update the information about his order using the `customer number`.

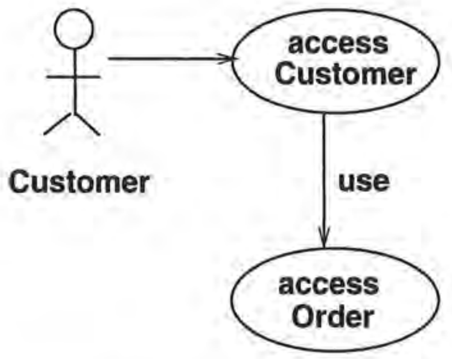


Figure 3: Access Customer Order Use Case Diagram.

2.3.2 Access Order Use Case

A customer can place a new order by inserting the product information, and the quantity he wishes to buy. If the information about an order is deleted using its `Number`, then the information about `Orderlines` for that order are also deleted. An order can be updated by updating the `orderline` for that order.

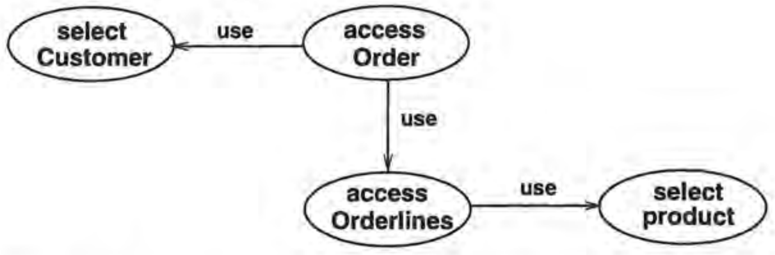


Figure 4: Access Order Information Use Case Diagram.

2.3.3 Access Product Use Case

A new product can be inserted with its `product number, name, color, and unit price` for that product. The information about the supplier who supplies that product, and the warehouse where that product is stored, is also inserted. The information about a product can be deleted/updated using its `number`. The information about the supplier who supplied this product can only be updated. The information in `Warehouse` can be deleted/updated for this product. The relationship `supplied-by` between supplier and product has to be deleted/updated to keep the database in a consistent state.

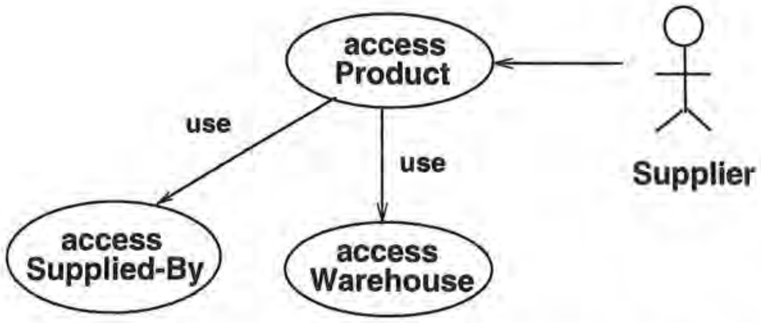


Figure 5: Access Product Information Use Case Diagram

2.3.4 Access Supplier Use Case

A Supplier can add a new product by entering the information about the product he wishes to supply and the details about the warehouse it is located. The information about a supplier can be deleted/updated using his `number`. The information about the `Product` supplied by that supplier can only be updated. The information in `Warehouse` is also deleted/updated for this product supplied by that supplier. The relationship `supplied-by` between supplier and product has to be deleted/updated to keep the database in a consistent state.

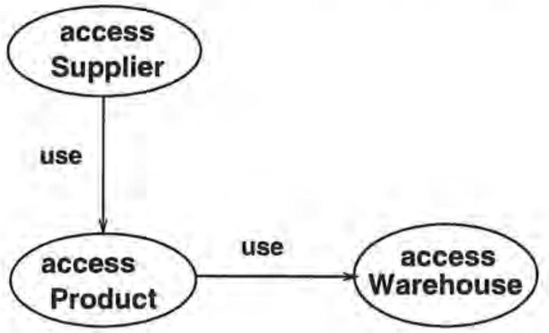


Figure 6: Access Supplier Information Use Case Diagram.

2.4 Organization of a Form

We have organized a form in such a way that all the needed information for that form can be displayed. The *main area or main form* is the place where the information about the entity we are interested in is displayed. The main form contains all the attribute values of the entity we are interested in. The related information for the entity of our interest is displayed in a *subform*. In the subform, the attribute values are displayed in a row of a table. This representation for a subform can be considered as a repeating group or a nested group. A main form can have as many subforms as the user needs, but we allow only nested subforms up to one level, by which we mean we can not have a subform within a subform. Our organization of a form is shown in Fig. 7.

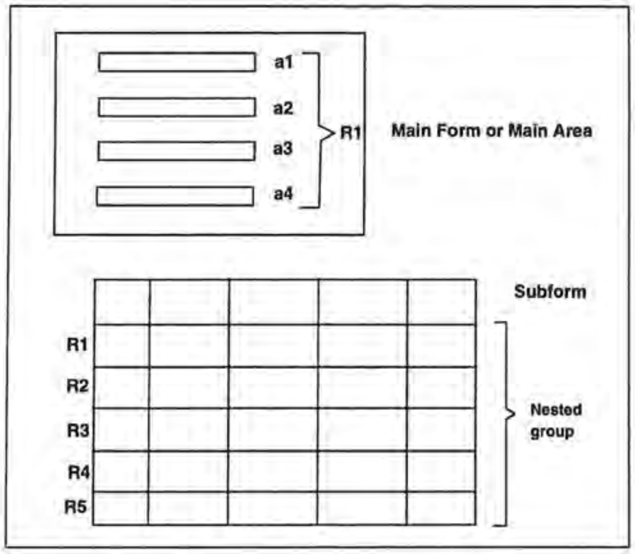


Figure 7: Organization in a Form.

3 Creating Forms from a Class Diagram

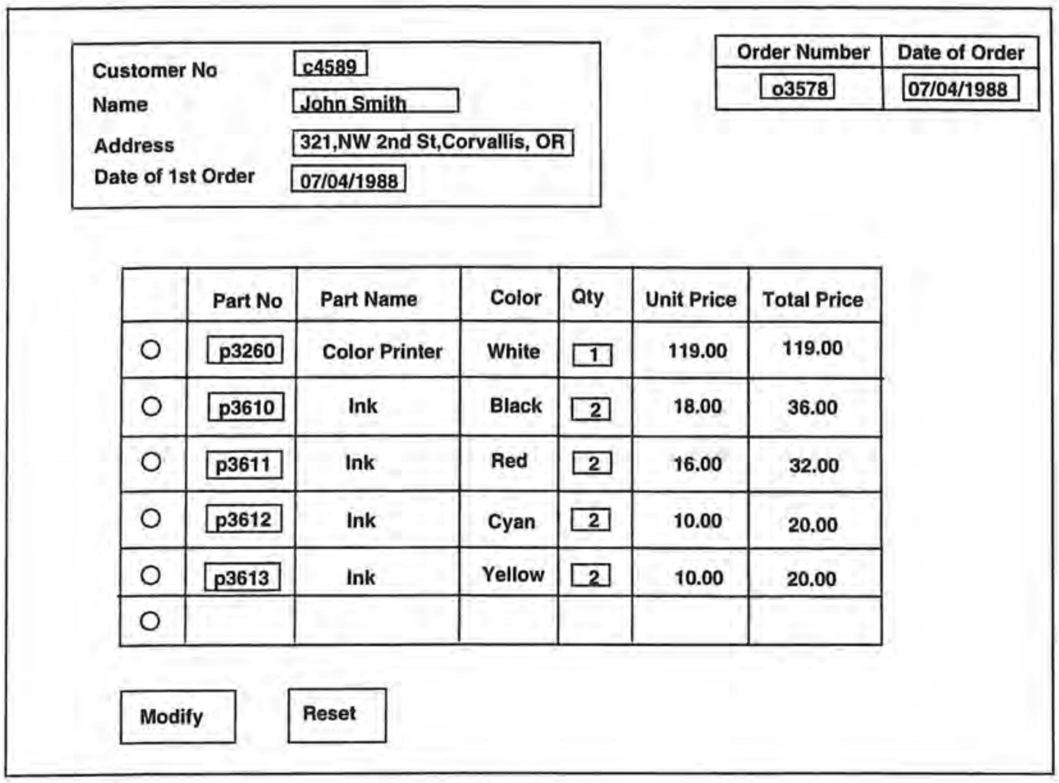
In this section we describe our methodology for building forms from a class diagram and use cases. For this purpose, we formulate a set of rules for entity types that will be covered by a form. The entities that may be covered by a form are also discussed. In order to explain our methodology, an order form shown in Fig. 8 is used.

The order form displays the attribute values, `order number` and `order date` of that `order`. The form also displays the attribute values of the customer to whom that order belongs and the orderlines for that order. An orderline consists of the quantity and the attribute values of the product ordered. For every order, the nested group of related orderline entities are displayed in the same form.

The use case diagram for this order that cover the entities in the class diagram is shown in Fig. 9.

3.1 Coverage of Entity-Types by a Form

The following rules state the restrictions on a set of entity-types to be displayed simultaneously in a form. Assume that a form uses an instance of entity type A as an *anchor*



Customer No	c4589
Name	John Smith
Address	321,NW 2nd St,Corvallis, OR
Date of 1st Order	07/04/1988

Order Number	Date of Order
o3578	07/04/1988

	Part No	Part Name	Color	Qty	Unit Price	Total Price
O	p3260	Color Printer	White	1	119.00	119.00
O	p3610	Ink	Black	2	18.00	36.00
O	p3611	Ink	Red	2	16.00	32.00
O	p3612	Ink	Cyan	2	10.00	20.00
O	p3613	Ink	Yellow	2	10.00	20.00
O						

Figure 8: Form for updating Order Information.

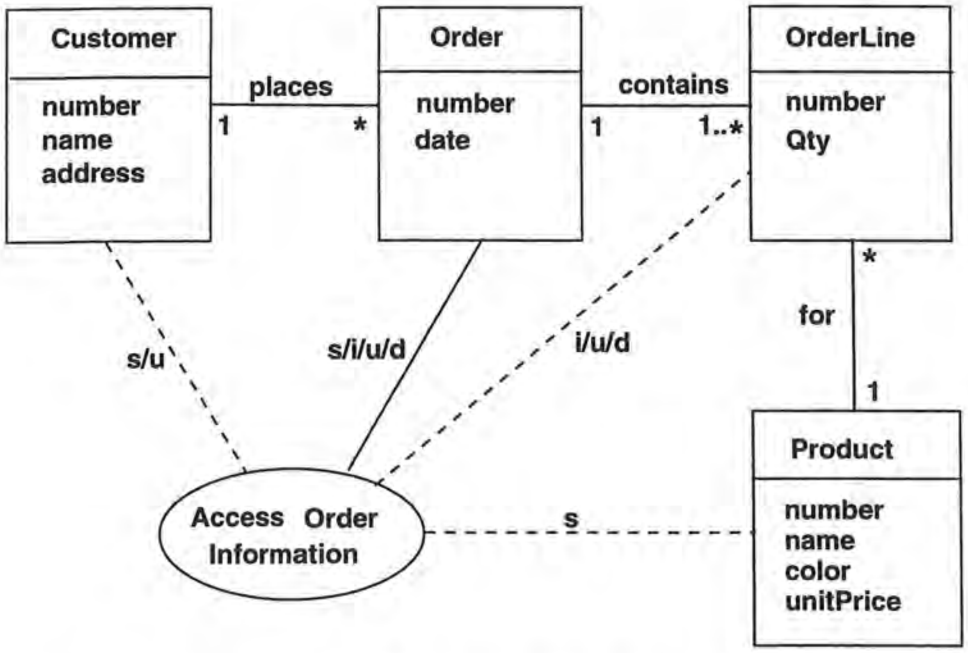


Figure 9: Objects accessed by Access Order Information Use Case.

entity, which is the main entity whose state the form displays, and that there is an relationship type between entity type A and entity type B, which is a *related entity*.

1. When the relationship-type between two entity-types A and B is one-to-one or many-to-one, the type B entity associated with a type A entity x can be *appended* to x. This is called *appending* and the type B entity is called an *appended entity*.
2. When the relationship-type between two entity-types A and B is one-to-many or many-to-many, many instances of a type B entity associated with one type A entity x can be *expanded*. This is called *expansion* and the type B entity is called an *expanded entity*.
3. An appending operation is possible for any entity, i.e., the anchor entity, an appended entity, or an expanded entity. Furthoremore, any number of appending operations are allowed within a form.
4. An expansion is possible only from the anchor entity or from an entity directly or indirectly appended to the anchor entity. We can show, within one form, only those entities that can be reached via at most one expansion.
5. The attribute values of the anchor entity and those of the entities appended to it directly or indirectly can be displayed in the main area of the form.
6. The attribute values of the expanded entity and those entities directly or indirectly appended to the expanded entity can be displayed as a row of a table in a subform within a form.
7. Entities that cannot be linked to an anchor entity must be able to become an anchor entity.
8. Sum of all operations *select, insert, update, and delete* allowed by all the forms must cover all the entities and all the relationships in the class diagram.

3.2 Structure for an Order Form

Consider a form for searching records in the e-order company database based on the anchor entity `ORDER`. The structure for this form is shown in Fig. 10 In this case, `CUSTOMER`, `ORDERLINES`, and `PRODUCT` belong to *related entity type*. The record of the `CUSTOMER` is appended with the record `ORDER` simultaneously in the main form, and the records of `ORDERLINES` are expanded for one particular *order* in the subform. The attribute values of the corresponding `PRODUCT` are appended with those of an orderline.

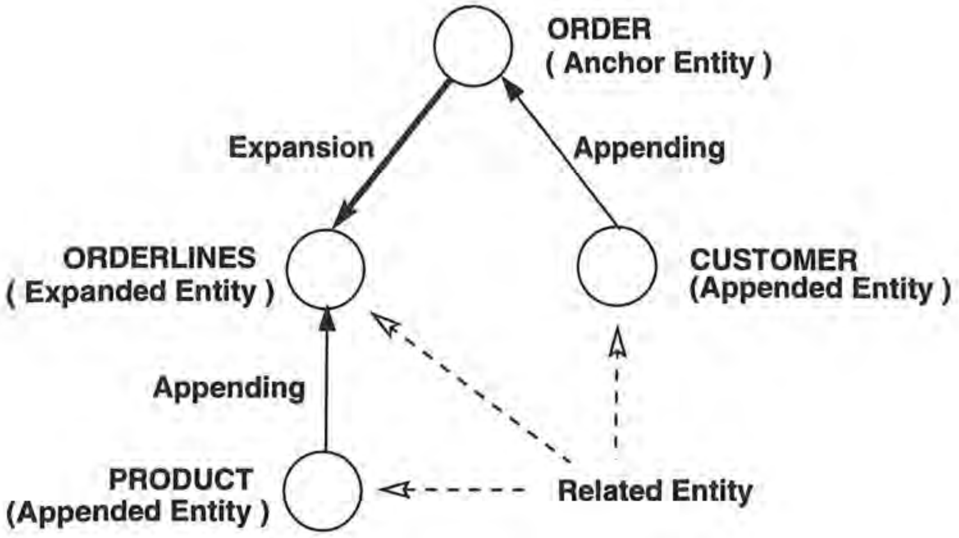


Figure 10: Structure in order form.

3.3 Explanation of Rules

Rule 1

According to this rule, the entity on the *one side* of a relationship is appended with the attribute values of the *many side* . Consider the example in Fig. 8 and Fig. 9 where anchor entity is `ORDER` and the related entity is `CUSTOMER` and the relationship type is *many-to-one*. The attribute value of a customer is appended to one particular order.

Rule 2

According to this rule, the entities on the *many side* of the relationship are expanded for the attribute value of the entity on the *one side*. Consider the example in Fig. 11 and Fig. 12, where anchor entity is `CUSTOMER` and the related entity is `ORDER` and the relationship type is *one-to-many*. The attribute values of order are expanded for one particular customer.

Rule 3

We do not enforce a restriction on the number of appending operations, because the appended entities can be displayed as a column in a table.

Rule 4

We enforce a restriction on the number of expanding operations, because more than one expansion can not be displayed in one form. The expanded entities are displayed as a row of a table within a form. In that case, we can not display another expanded entity for an already expanded entity simultaneously in one form.

Rule 5

Consider the example in Fig. 13 and Fig. 14 where anchor entity is `PRODUCT` and the related entity types is `WAREHOUSE`. The attribute values of `warehouse` are appended

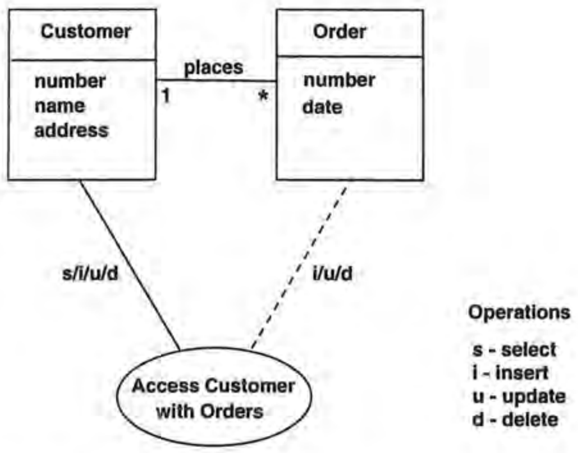
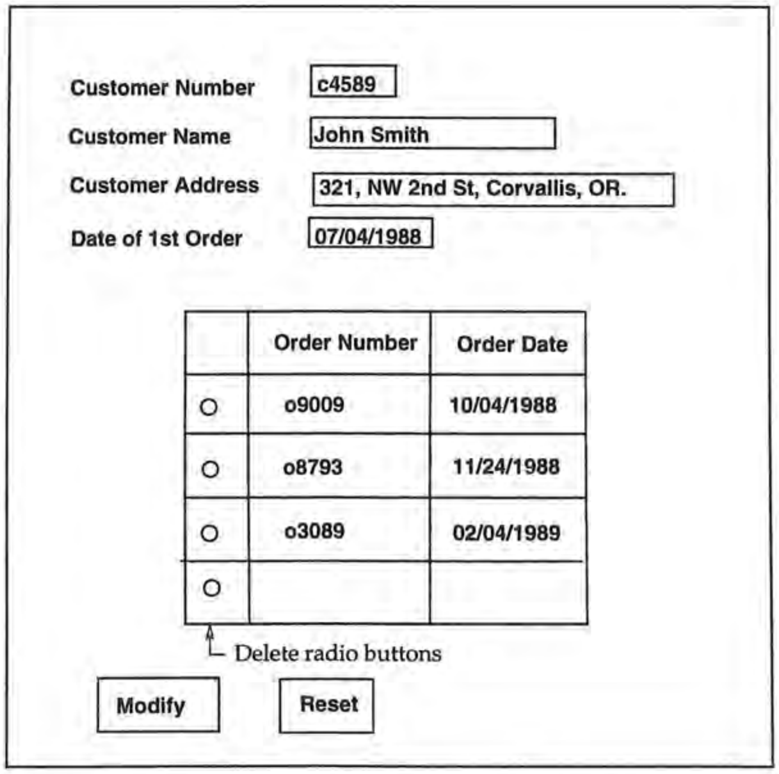


Figure 11: Objects accessed by Access Customer Information Use Case.



Customer Number	c4589
Customer Name	John Smith
Customer Address	321, NW 2nd St, Corvallis, OR.
Date of 1st Order	07/04/1988

	Order Number	Order Date
O	o9009	10/04/1988
O	o8793	11/24/1988
O	o3089	02/04/1989
O		

Figure 12: Form for updating Customer Information.

with those of a product, and are displayed in the main area of the form along with the attribute values of a product.

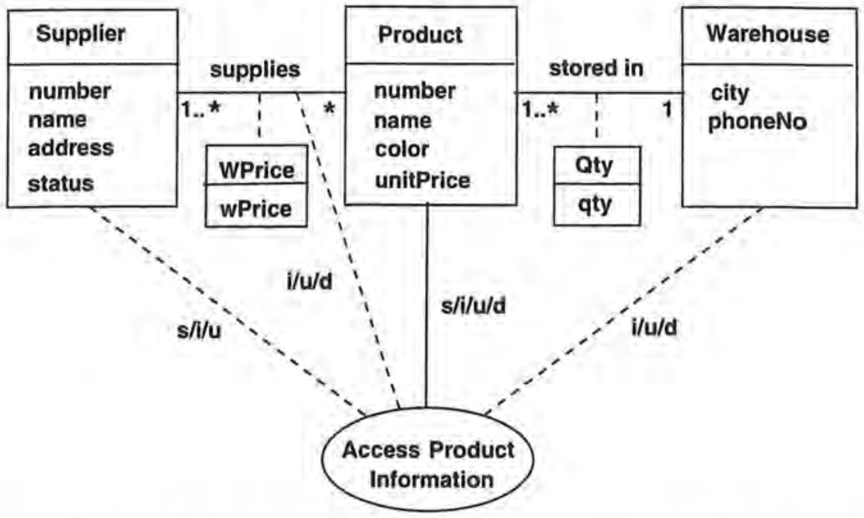


Figure 13: Objects accessed by Access Product Information Use Case.

Rule 6

In Fig. 15 and Fig. 16 the anchor entity is `SUPPLIER` and the related entity-types are `PRODUCT` and `WAREHOUSE`. The attribute values of the `PRODUCT` are expanded for one particular *supplier*, and are displayed as a subform showing the attribute values of a product in a row.

Rule 7

This rule can be clearly explained using the class diagram shown in Fig. 2. When we consider `ORDER` as an anchor entity for one form, then we can not reach `SUPPLIER` entity. So we need to have another form having `SUPPLIER` as an anchor entity.

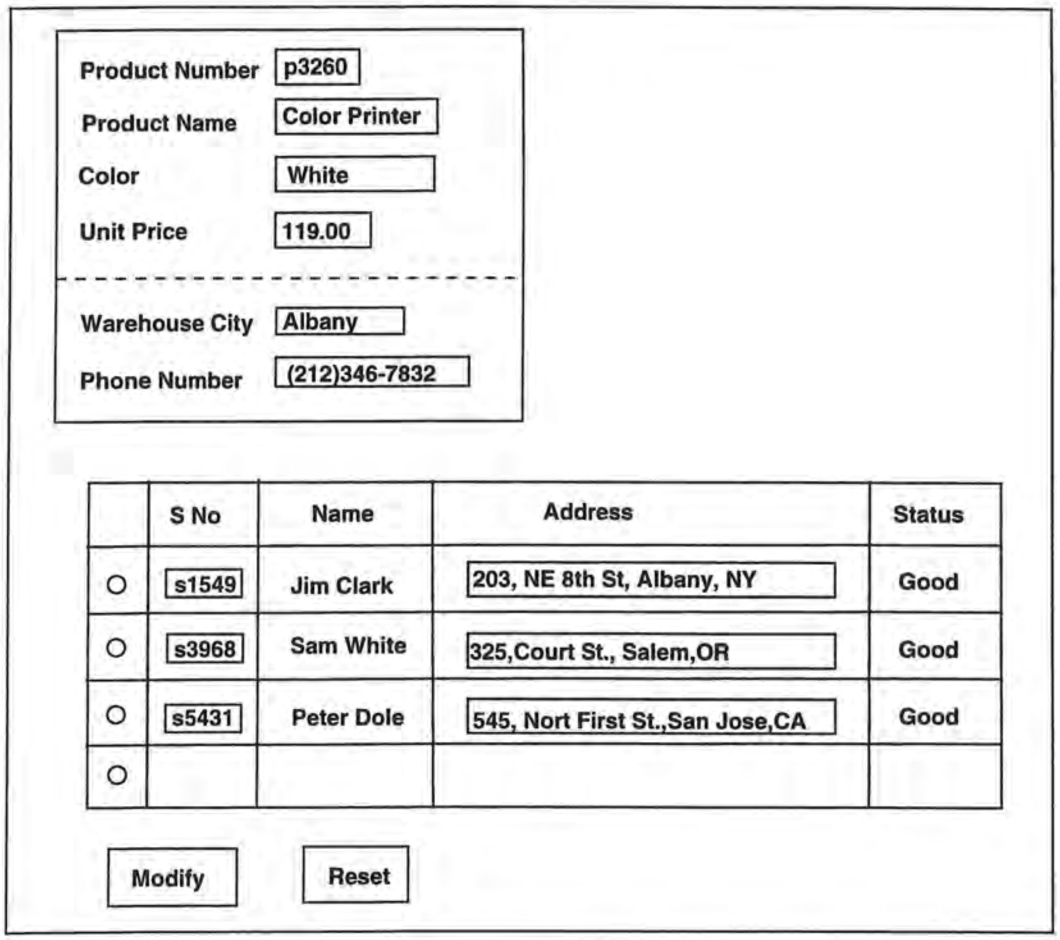
Consider the case, where `CUSTOMER` is an anchor entity, then the entity `PRODUCT` can not be reached by that form, so we need to have `PRODUCT` as an anchor entity in another form.

Rule 8

This rule says that all the entities and relationships in the class diagram should be select-able, insert-able, update-able, and delete-able by the set of forms generated for that particular application.

3.4 Soundness and Completeness of rules

As we have formulated a set of rules to generate forms, we discuss the soundness and completeness of those rules in this section.



Product Number: p3260

Product Name: Color Printer

Color: White

Unit Price: 119.00

Warehouse City: Albany

Phone Number: (212)346-7832

	S No	Name	Address	Status
O	s1549	Jim Clark	203, NE 8th St, Albany, NY	Good
O	s3968	Sam White	325,Court St., Salem,OR	Good
O	s5431	Peter Dole	545, Nort First St.,San Jose,CA	Good
O				

Figure 14: Form for updating Product Information.

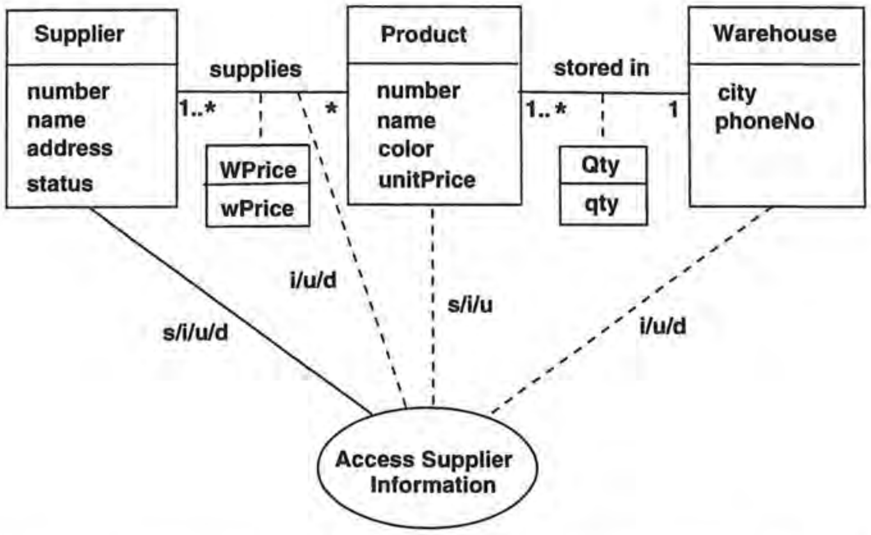
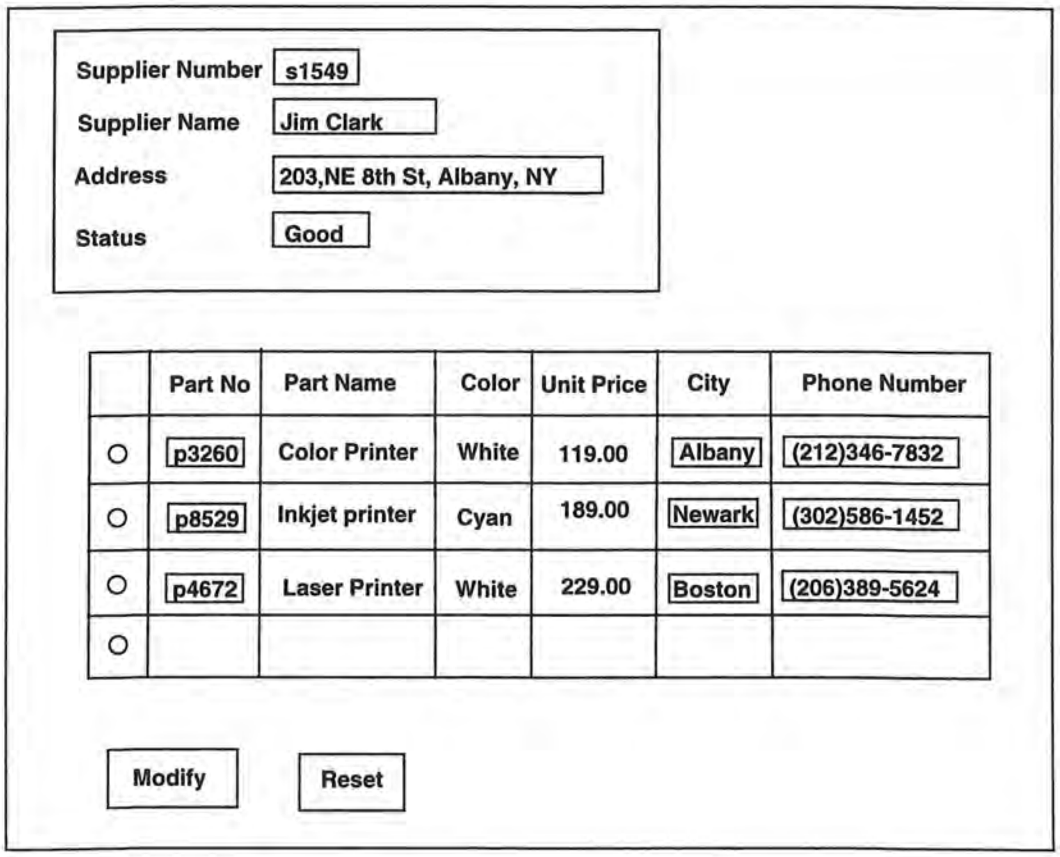


Figure 15: Objects accessed by Access Supplier Information Use Case.



Supplier Number	s1549
Supplier Name	Jim Clark
Address	203,NE 8th St, Albany, NY
Status	Good

	Part No	Part Name	Color	Unit Price	City	Phone Number
○	p3260	Color Printer	White	119.00	Albany	(212)346-7832
○	p8529	Inkjet printer	Cyan	189.00	Newark	(302)586-1452
○	p4672	Laser Printer	White	229.00	Boston	(206)389-5624
○						

Figure 16: Form for updating Supplier Information.

Soundness of rules: In order to show that the set of rules are sound, we show that the forms generated from those rules are correct. The set of rules identifies the locations of an anchor entity and appended and expanded entities. The anchor entities can be displayed in the main area of the form. The entities appended to the anchor entity can be displayed in the main area of the form. Since the rules allow only one expansion from the anchor entity, one level of nested entities is needed. The expanded entities can be displayed as rows in a subform. This is because as there may be many expanded entities, and as we need to display all of them, each can be displayed in one row of a table. Moreover as there are many entities displayed in one table, we call them as a nested group. The entities appended to an expanded entity should be displayed as a row of a table in a subform. This is correct as the entity that is appended to an already expanded entity has to be displayed along with the expanded entity. As all the entities selected by one form can be displayed correctly, we can say that the set of rules given in previous section is sound.

Completeness of rules: To show that rules are complete, we can generate forms using every entity type as an anchor type. This is very trivial, and hence there can be many forms generated which may not be needed.

This can be overcome by minimizing the number of anchors. The problem of minimizing the number of forms is probably an intracable problem, as it is similar to the vertex cover problem, which we already know is a NP-Complete problem. To minimize the number of anchors, what we have to do is select anchors such that only one expansion is allowed from the anchor.

Completeness of a set of forms: To show the completeness of the generated forms, we discuss about the operations applied to each entity type and relationship type. The possible operations for an entity and a relationship are *select, insert, update, and delete.* If we can show that all the entities and relationships are select-able, insert-able, update-able, and delete-able, then we can say that the set of forms provided by the rules are complete. We can do all the operations on the entity *customer* using the customer form, where the entity customer is an anchor. We can do all the operations on the entity *order* using the order form, where the entity order is an anchor. We can do all the operations on the entity *supplier* using the supplier form, where the entity supplier is an anchor. The entity product is select-able from order form, insert-able and update-able using supplier form, and delete-able using product form. So all the operations are possible on the entity *product* from all the forms for this e-order application. The relationship type *contains* between entities *order* and *orderline* is select-able, insert-able, delete-able, and update-able using order form. In this case, the entity *orderline* is a week entity, and so the operations applied to the relationship instance propagates to the related entity. So we are not showing the operations on the relationship, but directly showing them on the related entity. In general, when the cardinality at the anchor side of a relationship is 1, then the operations applied to the relationship cascades to the related entity. The relationship type *supplied-by* between entities *supplier* and *product* is also select-able, insert-able, delete-able, and update-able using either supplier form or product form. The relationship is a many-to-many relation, and so we have to explicitly apply the operations to the relationship. In general, if the cardinality at the anchor side is many, then the operations are applied separately to relationship and entity instances.

Assume that a form uses an instance of entity type A as an anchor entity and that there is an association type between entity type A and entity type B.

- *Complete Linking:* If the multiplicity at the A side of this association requires at least one instance, i.e., the multiplicity is 1, 1..*, and so on, then the insert and delete operations applied to the instances of B implies that those operations are also applied to the association. In this case, we say that the linking is *complete.*

  When the linking is complete, all the entities of B are covered by a form, in which entity A is an anchor. As the operations applied to the related entity implies to the relationship type as well, we are not explicitly showing the operations on the relationship types.

- *Partial Linking:* If the multiplicity at the A side of this association allows zero instance, i.e., the multiplicity is 0..1, *, and so on, some instances of B may not be associated with any instances of A, and hence the operations applied to the instances of B through the form may not be applicable to all the instances of B. However, the select, insert, and delete operations applied to the instances of B through the form imply that those operations are applied to the association also. In this case, we say that the linking is *partial.*

  When the linking is partial, not all entities of B are covered by a form for an anchor A. As the operations applied to the related entity does not imply to the relationship type, we have to explicitly show the operations applied to the relationship types.

In partial linking, consider the entity types A and B are connected by an many-to-many association type. Then having one form, e.g., that uses entity type A as an anchor entity type, and B as an expanded entity type cannot access all the entities of B. So we need another form that uses B as an anchor entity type. In the e-order example, the supplier and product have a many-to-many relation, and there exist 2 forms, one with product as an anchor and another with supplier as an anchor.

Entity types that are directly or indirectly connected to the anchor entity type by patial links can be included simultaneously in one form. Details about the operations allowed on a form and verification of forms [6] for an application are beyond the scope of this project report.

3.5 Preservation of Structural Integrity

The referential integrity rule for a relational database states that there must not be any unmatched foreign key values. This integrity constraint is specified on a database schema and is expected to hold on every database instance on that schema [2]. Referential integrity constriants typically arise from the relationships among the entities represented by the relational schema. Referential integrity and foreign key are both dependent on each other.

Foreign Key Rules for Delete/Update Operations:
The foreign key rules RESTRICTED and CASCADES for delete/update operations are supported by a relational database system.

In our schema, the entity ORDER and ORDERLINE have a one-to-many relationship type as shown in Fig. 9. The operation performed on the anchor entity ORDER is also performed on the entity ORDERLINE. We do not have to explicitly state that the relationship is also modified by the operation. The delete/update operation on the anchor entity, performs the same on the relationship and it gets propagated to the related entity ORDERLINE.

Consider the form shown in Fig. 15. As the relationship type between SUPPLIER and PRODUCT is many-to-many, we have to explicitly state the operation performed on the relationship. If we are to use supplier form to delete a product, supplied by the supplier, then we can get the product details only through the relationship. So we need to specify the operation on the relationship also.

4 Implementation Details

This section covers the details about creating a database for an e-order company in an Oracle 8.0 system. Forms for this application are developed using Java Servlets. Implementation of the Java Servlets is discussed in Section 4.2. In Section 4.3, we show that all the entities and relationships are covered by the forms generated for this application.

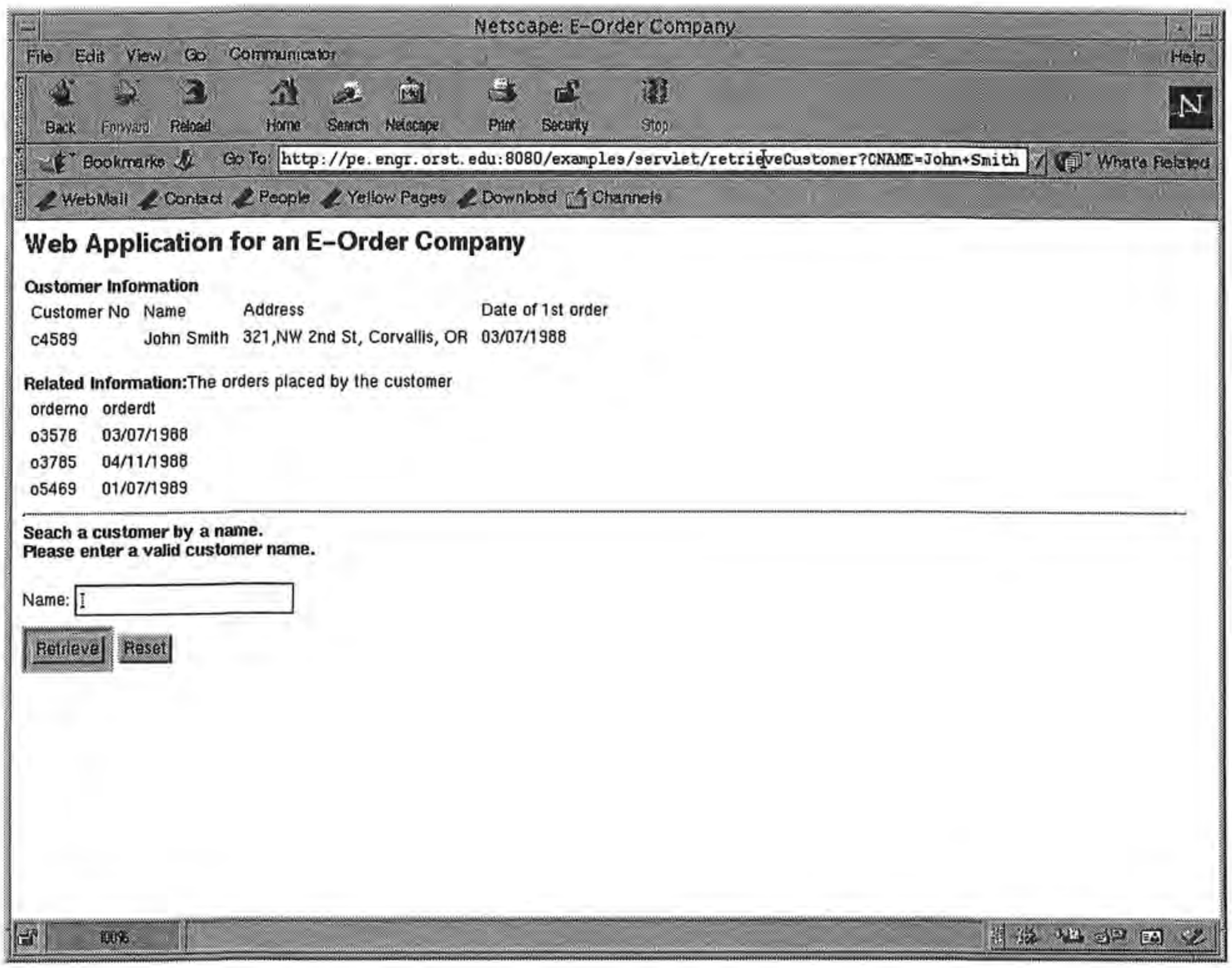
4.1 Oracle Database

The database created for the e-order application has 7 tables, which were created using following SQL commands.

```
create table CUSTOMER
  ( cno         char(6)     not null,
    cname       char(20)    not null,
    caddr       char(30)    not null,
    forderdt    char(10),
    primary key (cno) );

create table ORDERS
  ( orderno     char(6)     not null,
    orderdt     char(10),
    cno         char(10)    REFERENCES CUSTOMER(cno),
    primary key (orderno) );

create table ORDERLINE
  ( orderno   char(6)    REFERENCES ORDERS(orderno),
    pno       char(6)    REFERENCES PRODUCT(pno),
    qty       varchar2(3),
    primary key (orderno, pno) );

create table PRODUCT
  ( pno       char(6)    not null,
    pname     char(20)   not null,
    color     char(6),
    unitprice smallint,
    primary key (pno) );

create table SUPPLIER
  ( sno       char(6)    not null,
    sname     char(20)   not null,
    saddr     char(30),
    status    char(10),
    primary key (sno) );

create table SUPPLIED-BY
  ( sno       char(6)    REFERENCES SUPPLIER(sno),
    pno       char(6)    REFERENCES PRODUCT(pno),
    wprice    smallint,
    primary key (sno, pno) );

create table WAREHOUSE
  ( city      char(20)   not null,
    phoneno   char(15)   not null,
    pno       char(6)    REFERENCES PRODUCT(pno),
    primary key (phoneno) );
```

4.2 Java Servlets

Java Servlets dynamically generate HTML forms and also interact with the back-end ORACLE database via Java Database Connectivity (JDBC). The form shown in Fig. 17 is for retrieving customer information from the database. This form is generated by a Java Servlet code, which is explained in sections 4.2.1 and 4.2.2.


Netscape: E-Order Company
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Go To: http://pe.engr.orst.edu:8080/examples/servlet/retrieveCustomer?CNAME=John+Smith What's Related
WebMail Contact People Yellow Pages Download Channels
Retrieve Reset
100%

Web Application for an E-Order Company

Customer Information

Customer No	Name	Address	Date of 1st order
c4589	John Smith	321,NW 2nd St, Corvallis, OR	03/07/1988

Related Information: The orders placed by the customer

orderno	orderdt
o3578	03/07/1988
o3785	04/11/1988
o5469	01/07/1989

Seach a customer by a name.
Please enter a valid customer name.

Name:

Figure 17: Customer Information Retrieval/Request Form

4.2.1 JDBC Connection

The section of the code shown below describes how to establish a connection to the database using JDBC. `DriverManager` is the class responsible for loading database drivers and creating a new database connection. We need to load the driver manager code using the command `class.forName()`, so that the program can use it. Statement object `dispStmt` is created so that we can query the database.

```
public class retrieveCustomer extends javax.servlet.http.HttpServlet {
  Connection con;
  Statement dispStmt;
  public void init(ServletConfig conf) {
    super.init(conf);
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con = DriverManager.getConnection(
        "jdbc:oracle:thin:@pe:1521:cs440","user", "pswd");
      dispStmt = con.createStatement();
    }
    catch (Exception e) {
      throw(new UnavailableException(this, "Sorry"));
    }
  }
```

4.2.2 Method doGet()

The form shown in Fig. 17 displays the customer information from the database. The section of the code given below describes the service method `doGet()` provided by the servlet. This method is used to get the request from the webpage, process the request, and respond by sending the results back to webpage. The servlets allow the user to do a query on the database based on a attribute. In the section of the code described below for the example shown in Fig. 17, `CNAME` is the variable for the customer name. The value of the variable is the *request* and it is processed to produce the *response*. HTML tags can be used in the servlet, to produce a clear and organized output.

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {
  ServletOutputStream out = resp.getOutputStream();
  resp.setContentType("text/html");
  out.println("<body bgcolor = white>");
  out.println("<b>Customer Information</b>");
  out.println("<TABLE><TR>");
  out.println("<b><td> Customer No </td></b>");
  out.println("<b><td> Name </td></b>");
  out.println("<b><td> Address </td></b>");
  out.println("<b><td> Date of 1st order </td></b>");
  out.println("</tr>");
  String name = req.getParameter("CNAME");
  if (req.getParameter("CNAME") != null) {
    try {
      ResultSet rs = dispStmt.executeQuery( "select * from CUSTOMER " +
       "where customer.cname =" + "'" + name + "'");
      while(rs.next()){
        out.println("<tr>");
        String t = new String(rs.getString("cno"));
        String e = new String(rs.getString("cname"));
        String m = new String(rs.getString("caddr"));
        String p = new String(rs.getString("forderdt"));
        out.println("<TD>" + t + "<TD>" + e + "<TD>" + m + "<TD>" + p );
      }
      rs.close();
      out.println("</tr></TABLE>");
    }
    catch (SQLException e) {
      out.println("ERROR WITH DATABASE </table>");
    }
  }
  .        .        .
  out.println("<HR><B>Seach a customer by a name.</B><BR>");
  out.println("<B>Please enter a valid customer name.</B>");
  out.println("<FORM method=GET action=\"retrieveCustomer\">");
  out.println("Name:<INPUT TYPE=TEXT NAME=\"CNAME\">");
  out.println("<INPUT TYPE=SUBMIT VALUE=\"Retrieve\">");
  out.println("<INPUT TYPE=RESET VALUE=\"Reset\">");
  out.println("</FORM>");
  out.println("</BODY></HTML>");
  return;
 }
 public void dispose() {
   try {
     dispStmt.close();
     con.close();
   } catch(SQLException e) {}
 }
}
```

The related entity order information is obtained in a similar way, using a query to retrieve information about the order number and date for that particular customer. The SQL query for retrieving order information using `customer name` is

```
select orderno, orderdt from ORDERS, CUSTOMER
where ORDERS.cno = CUSTOMER.cno and
      customer.cname =" + "'" + name + "'" );
```

4.3 Forms for the database application

Based on the rules discussed in Section 3, we generated forms for the e-order database application. The generated forms from use cases covered all the entity types and the relationship types in the class diagram. Fig. 18 shows that all the entities and relationships are covered using the four forms generated for this application.

Form	Anchor Entity type
Form 1	Customer
Form 2	Order
Form 3	Supplier
Form 4	Product

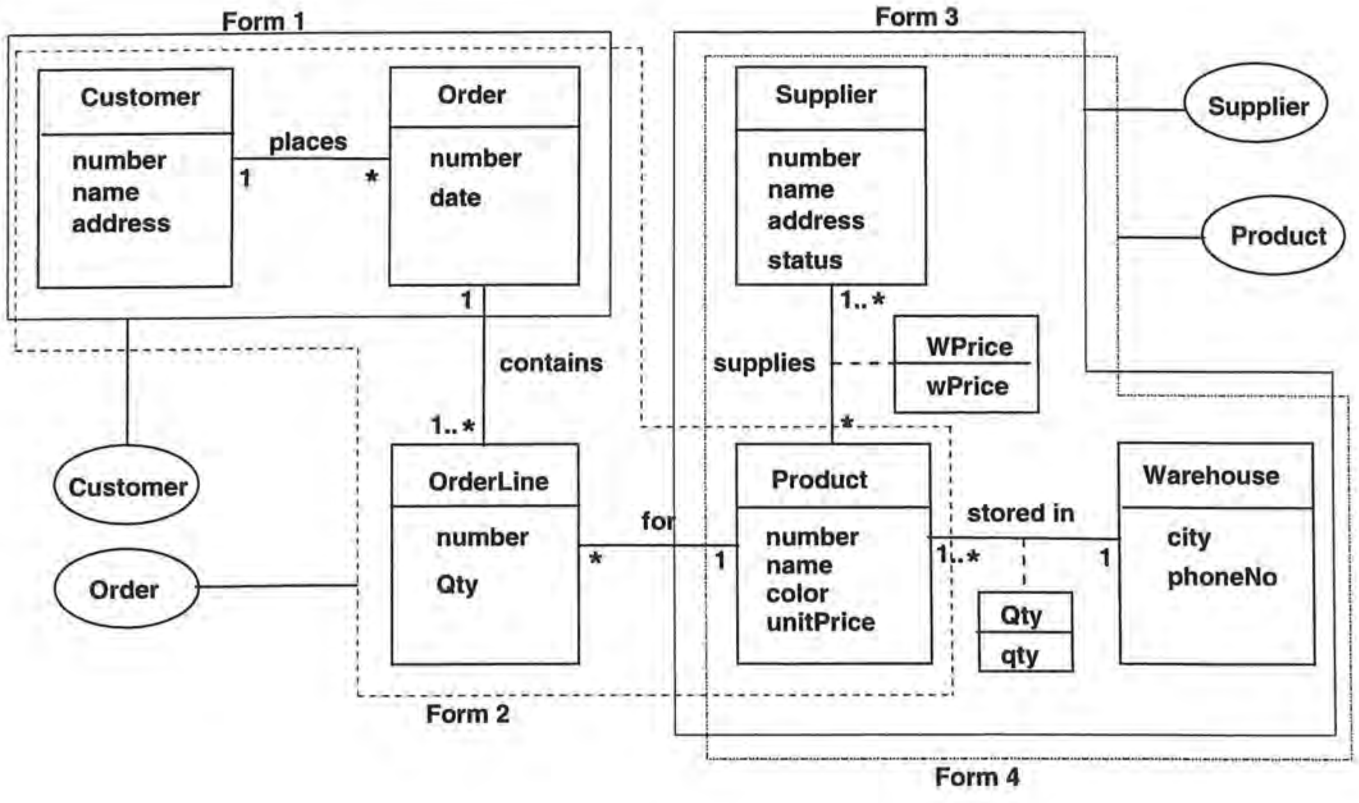


Figure 18: Coverage by forms.

5 Conclusions and Future Work

We proposed a new model for generating forms from class diagrams, using use cases. Use cases were used to specify what information will be accessible by a single form.

We introduced special concepts called *anchor entity*, which is the main entity in the main area of a form, and *expanded entity* and *appended entity*, which were used to show where the related entities will be displayed in one form. We further discussed the entity types covered in a form and the entities covered by a form. We also studied the operations that can be allowed in order to preserve the structural integrity for the proposed model.

We demonstrated that the forms can be built on the Internet using Java Servlets by building forms for the e-order company database. In Fig. 18, we show that all the entity types and relationship types are covered by the forms generated by the specified use cases for the e-order company database.

We hope that our model will become a basis for the future development of a software package for generating forms. Future work for this project includes automatic generation of forms from class diagram using use cases, and dynamically generating SQL by allowing the users to select the variables for performing a query.

References

[1] Grady Booch, James Rumbaugh, Ivar Jacobson *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[2] C.J.Date *An Introduction to Database Systems*, sixth edition, Addison-Wesley, 1995.

[3] Ramez Elmasri, Shamkanth B. Navathe *Fundamentals of Database Systems*, second edition, Benjamin/Cummings, 1994.

[4] Edward Honour, Paul Dalberth, Ari Kaplan, Atul Mehta, *ORACLE8 HOW-TO*, Waite Group Press, 1998.

[5] UML Notation Guide, version 1.1, Rational Group, 1997.

[6] Rashmi Dwarakanath, *Applying use-cases for Design and Verification for Database Applications*, Masters Project Report, Dept. of Computer Science, Oregon State University, 1999, in preparation.