On Mapping between UML and Entity-Relationship Model

Yongzhen Ou

Fakultät für Mathematik und Informatik,
Universität Konstanz, Fach D188, 78457 Konstanz, Germany

Abstract: Nowadays, the Entity-Relationship Model (ERM) is the most important and widely used method for modeling data and designing databases. On the other hand, the Unified Modeling Language (UML) is expected to become more and more popular in object-oriented analysis and design (OOA/OOD). As a by-product of OOA/OOD, a database design can be derived by mapping of objects to entities. The purpose of this paper is to define a mapping between UML and ERM. The translation of a UML class diagram to and from an ER diagram is also elaborated. This work is part of a multi-model multi-tool database application engineering framework that is being designed and prototyped at University Konstanz.

1 Motivation

Since its introduction by Chen (1976), the Entity-Relationship Model (ERM) has found its wide acceptance in the area of database design and related fields. In the ER model, all data are viewed as stating facts about entities and relationships, i.e., connections or associations between entities. These simple and clear structuring concepts allow users to naturally model the "things" in real world. As a result, most of the existing databases are first designed conceptually using some variant of ERM and then got their conceptual schemata translated into logical schemata of the target system (Batini (1992), Elmasri and Navathe (1994)).

In the beginning of this year, the Unified Modeling Language (UML), which unifies the Booch (1994), OMT (Rumbaugh et al. (1991)) and OOSE (Jacobson et al. (1992)) methods and incorporates ideas from other methods and industry practice, was submitted to OMG (Object Management Group) for adoption as a standard. That means the UML may become a standard modeling language for object-oriented development soon. For UML is meant to be applicable to the modeling of all types of systems, it can also be applied to conceptual database design. How to map the constructs between UML and ERM? Is it possible to automate the translation of a UML class diagram to an ER diagram? This paper tries to answer these questions. A good solution to these problems is very useful for forward engineering and reverse engineering of database design.

In forward engineering, a software application can be developed by using the UML notations. The persistent classes underlying the application can then be stored in an object-oriented database or in a relational database.

In the latter case, a mapping from the object model to the relational model is required. The ER model is the best choice to represent the relational model visually. Moreover, many database designers and database users have been using ER model for a long time and have mastered all the modeling techniques of ER models. They prefer viewing database schemata as ER diagrams. Furthermore, it can also happen that some preexisting parts of the UML class diagrams have originally been modeled as entities and relationships in some ER diagrams. If the UML class diagram can also be shown as ER diagram, it would be a useful first step towards schema integration and database integration (Batini (1992)). In all these cases, it is desirable to transform UML class diagrams to ER diagrams automatically. Even though almost all OO design tools -including Rational Rose- provide some mapping of their class diagrams to database schema definitions (e.g., in SQL), these transformations are typically very limited in functionality (coverage of integrity constraints) and for flexibility (e.g., choices for representing generalization hierarchies). Therefore, the goal of our work is to develop a multi-model, multi-tool design platform that allows the use of different analysis/design/implementation tools, possibly based on different models (such as UML and ER variants). Such an environment will give the designer a broad choice of models and tools to use for the individual tasks during the forward and also reverse engineering tasks for database applications.

In reverse engineering, if an existing ER diagram can be viewed as a UML class diagram, one can easily change, update, or enhance the existing database design under the OO paradigm. Showing the ER diagrams as UML diagrams also make it possible to integrate the existing database designs with other OO designs. In these cases, the mapping of ER model to UML is needed. The automatic transformation of ER diagrams to UML class diagrams is useful.

The layout of this paper is as follows. Section 2 gives an overview of UML, Section 3 discusses the mapping of UML to ERM. In Section 4, the translation of a UML Class Diagram to an ER Diagram is elaborated. Section 5 briefly discusses the mapping of ERM to UML. Finally Section 6 concludes this work.

2 The Unified Modeling Language (UML)

What is the UML? As its designers pointed out, the UML is a language for specifying, visualizing and documenting the artifacts of an object-oriented system under development (Booch et al. (1997)). It was developed jointly by Grady Booch, Ivar Jacobson, and Jim Rumbaugh at Rational Software Corporation, with contributions from other leading methodologists, software vendors, and many users. It provides the application modeling language for business process modeling with use cases, class and object modeling, component modeling, distribution and deployment modeling (Booch et al. (1997)). The UML semantics defines model elements, relationships among the model

elements, mechanisms organizing the model elements, and diagrams showing the projection of model elements. The UML also defines notations for its various types of diagrams. These notations provide a very convenient way to describe the various modeling aspects. The diagrams supported by UML are:

- Class diagram and object diagram
- Use case diagram
- Interaction diagram
  - Sequence diagram
  - Collaboration diagram
- State diagram
- Implementation diagram
  - Component diagram
  - Deployment diagram

In the process of object-oriented analysis and design, a model and a few diagrams can be produced. The model contains all of the underlying elements of information about a system under consideration, while diagrams capture different perspective, or view, of the system model. A class diagram shows types, classes, and their relationship; it shows the building blocks of the model. An object diagram describes instances and their relationship; it shows the sample data structure. Both class diagram and object diagram project the static structure of a system model. A use case diagram encompasses actors, use cases, and their relationship. It provides a natural high-level view of the intended functionality of the system. Interaction diagrams, including sequence diagram and collaboration diagram, display instances and their relationships (including messages), organized either by space (collaboration diagram) or by time (sequence diagram). A state diagram shows states and their relationships, organized by state, and is used to specify the overall behavior of a type. A component diagram encompasses components and their relationships; it reveals the dependencies among software units. A deployment diagram depicts components, nodes, and their relationships; it shows the distribution and interaction of components and objects on computational nodes.

In this paper, we concentrate on the UML semantics and notations describing the static structure aspects of a model, namely, types, classes, associations, etc., and the class diagram visually presenting these aspects.

3 Mapping of UML to ERM

The class diagram is core to a UML model. It presents the important abstractions in a system and shows how they relate to each other. This section

will discuss the concepts underlying the class diagram and their mappings to ER model. As there exist many variations of ER model, we take the one presented in (Elmasri and Navathe (1994)) as our ER model. This ER model has the following concepts: entities and attributes (include composite attributes and multivalued attributes), primary key; regular and weak relationships (they can also have attributes), roles, and cardinality; and weak entities. The graphical notation of the ER model is the same as that in the DB-MAIN CASE tool (Hainaut (1994)), which uses rectangle to represent entity type and hexagon to represent relationship type. Weak entity type is represented by double rectangles and its identifying relationship by double hexagons. If we do not explicitly indicate, we mean an entity an entity type and a relationship a relationship type in the following discussion.

3.1 Classes, Attributes, and Operations

According to the UML semantics, a class is the descriptor for a set of objects with similar structure, behavior, and relationships. A class is the implementation of type. A class can have attributes defining its structure and operations defining the behaviors of instances of the class. Obviously, an object is essentially an entity in ER model. Therefore, a class is mapped to an entity type in ERM. Attributes of the class are mapped to attributes of the resultant entity type. Mapping of operations is the future research. In UML, it is assumed that each object (instance of a class) is uniquely identified by its object identifier (OID). In the ER model, an entity is distinguished from other entities by the value of its key attribute. To accommodate this, a surrogate key is added to the resultant entity type which has a name of the entity type name concatenating "-ID" (such as window-ID). For example, Figure 1 shows the mapping of a class `Window` to an entity `Window`.

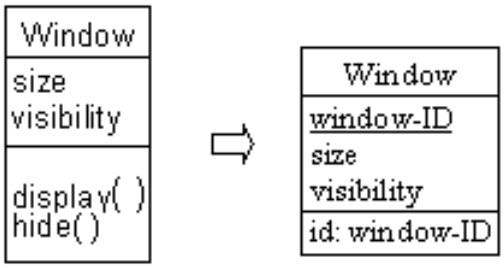

Figure 1: The mapping of a class to an entity

3.2 Relationships

A relationship is a semantic connection among classes. There exist several different kinds of relationships in UML, namely the bi-directional association, uni-directional association, dependency, aggregation association, inher-

itance and template instantiation.

Association

An association is used to represent a structural dependency among objects. A bi- directional association indicates that the involving objects are mutually dependent on each other, while a uni-directional association indicates the one directional dependency. Both bi-directional and uni-directional associations are mapped to relationships in the ER model, though the relationship in ER model has more mathematical sense. The UML supports n-ary (n > 2) associations, so does ER model. Therefore, an n-ary association is mapped to a n-ary relationship in ER model. Some associations may have association classes. An association class is just an association that has class properties (such as attributes) as well. Therefore, an association class is also mapped to a relationship with attributes. For example, in Figure 2, the association class `Job` in UML is mapped to the relationship `Job` in ER model.

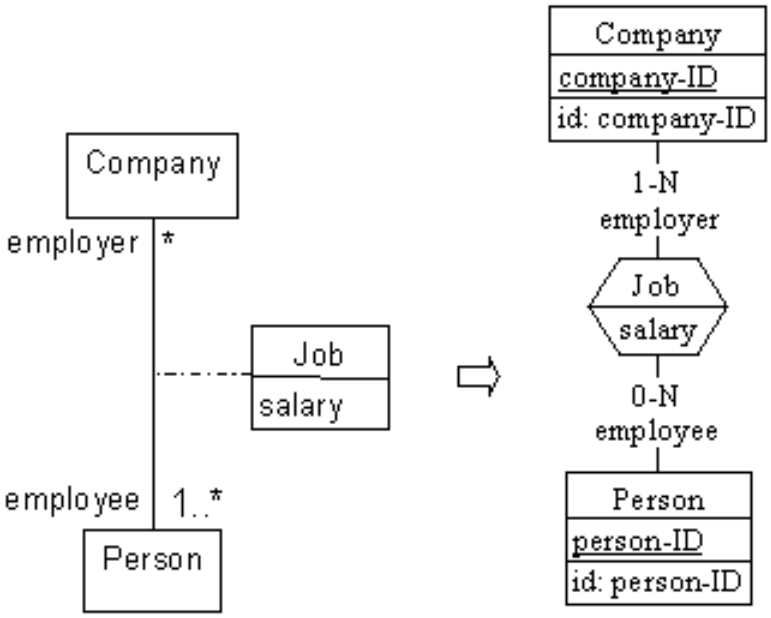

Figure 2: The mapping of an association class to a relationship

Qualified Association

A qualifier is sometimes used to qualify an association and reduces the effective multiplicity of an association. Usually only one-to-many and many-to-many associations may be qualified. The qualifier separates the objects on the "many" side into some subsets. A qualified association is also mapped to a relationship in ERM, just like the normal associations. However, depend-

ing on the multiplicity, the multiplicity and primary key of the resultant relationship are different.

For example, Figure 3 shows the mapping of an one-to-one qualified association to an one-to-many relationship. In this example, the qualified association indicates that a company may have one or more departments with each having a unique dept-name. A department belongs to only one company while departments from different companies may have the same dept-name. Here the qualifier "dept-name" happens to be an attribute of the class Department. Consequently, the resultant relationship R is an one-to-many relationship, and the key of this relationship can be the key of entity type Department (department-ID) or the combination of the key of entity type Company and dept-name (company-ID, dept-name).

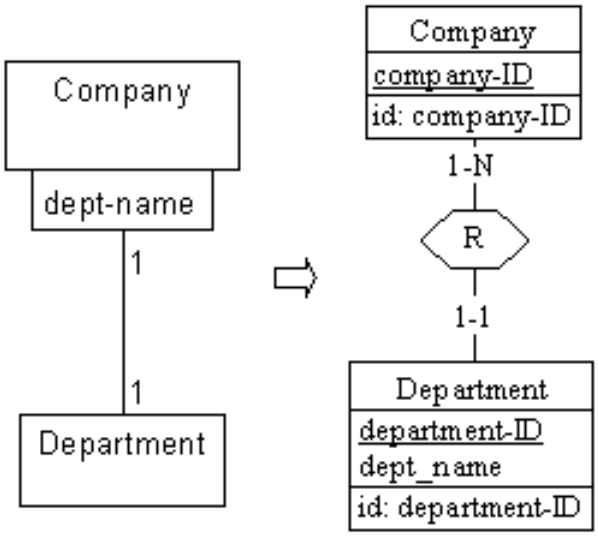

Figure 3: The mapping of a qualified association to a relationship – 1

Figure 4 shows an one-to-many qualified association. In this example, a bank may have many persons as their customers with each uniquely identified by his account#. A person may have zero or more accounts in zero or more banks. Therefore, the resultant relationship has the multiplicity of many-to-many. The qualifier "account#" becomes the attribute and part of the key of the relationship. The primary key of the relationship is then the combination of the key of bank and "account#" (bank-ID, account#).

Figure 5 illustrates the mapping of a many-to-many qualified association to a many-to-many relationship. In this example, a company may have employed many persons and a person may be employed by many companies. Each company may possess many offices with each headed by one or more officers (persons). The qualifier "office" divides the persons into a few subsets with each attaching to an office. Therefore, the resultant relationship has the multiplicity of many-to-many. The qualifier "office" becomes the attribute and part of the key of the relationship. The primary key of the

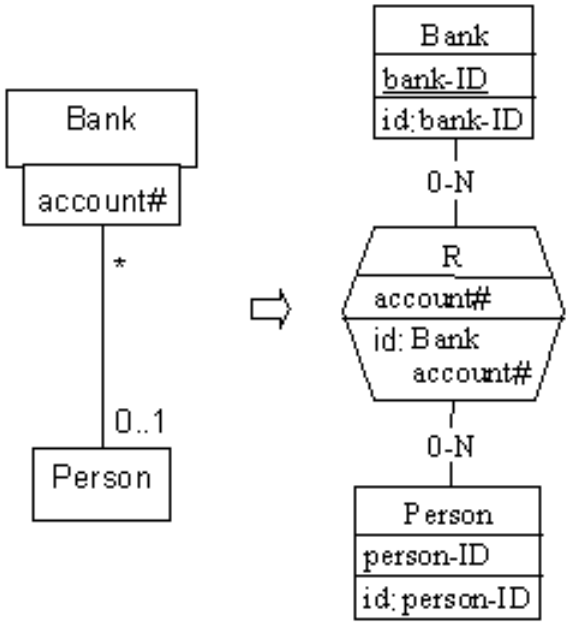

Figure 4: The mapping of a qualified association to a relationship – 2

relationship is "office" combined with the keys from entity type Company and entity type Person (company-ID, person-ID, office).

Aggregation

Aggregation is a special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. There are two kinds of aggregation depending on the containment of the parts in its whole: one is by reference (shared aggregation), the other is by value (composite aggregation). The composite aggregation is equivalence to the attribute composition.

Aggregation indicates that the existence of the parts is dependent on the whole. This existent dependency leads to the mapping of an aggregation to a relationship in the ER model with a total participation constraint on the entity representing the component part. For example, in Figure 6, the aggregation `Contains` is mapped to relationship `Contains`. There is no mapping for the UML adornment `ordered`. In such cases, a note or a comment can be added to the ER diagram or to its attaching documentation to accommodate the constraint definitions.

Inheritance

Inheritance is used when a more specific class is to incorporate structure and behavior of a more general class. The more specific class (the subclass)

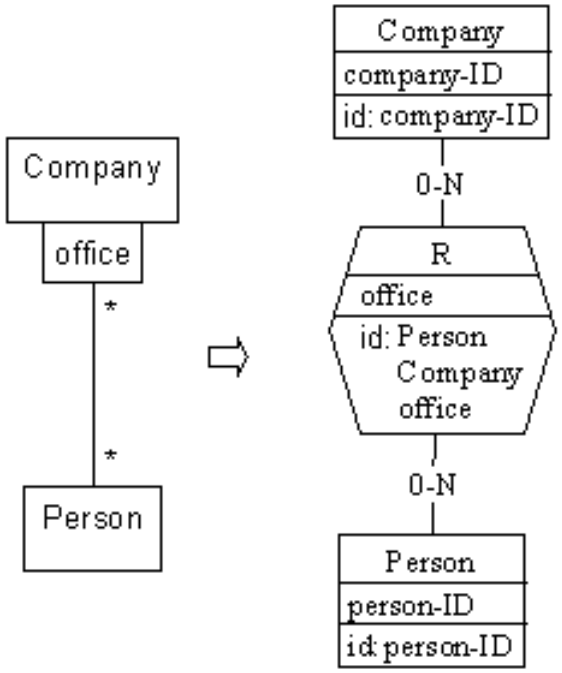

Figure 5: The mapping of a qualified association to a relationship – 3

is a specialization or extension of the more general one (the superclass). The original ER model does not directly support the concept of inheritance. However, an inheritance can be simulated by an isa relationship in the ERM, which indicates that the entity representing the subclass "is-a" kind of entity representing the superclass. For example, Figure 7 shows the mapping of a generalization in UML to some isa relationships in ER model.

Dependency

Dependency is a relationship between two classes, in which one of them (the client class) depends on some service(s) of another class (the supplier class).There are four kinds of predefined Dependency: *trace*(Trace), *refine*(Refinement), *uses*(Usage), and *bind*(Binding). The ER model has no direct representations for these concepts. However, the dependencies between model elements (the trace, refine and uses dependencies) in a UML class diagram can be documented and attached to the resultant ER diagram. The bind dependency is the same as the instantiates relationship and is discussed below.

Instantiates

An instantiates relationship between two classes indicates that one of them

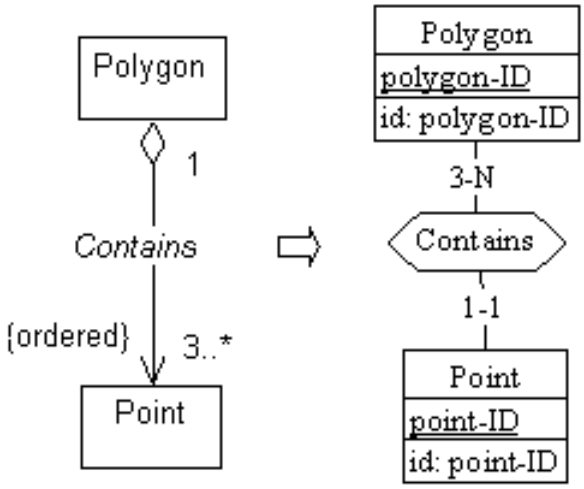

Figure 6: The mapping of an aggregation to a relationship

(the instantiated class) is created as a result of instantiation of the other class (the parameterized class, or the template class). The parameterized class has a formal parameter and the instantiated class replaces the formal parameter with an actual one. The instantiates relationship is actually a kind of inheritance as well. Therefore, an instantiates relationship is also mapped to an isa relationship in ERM. Moreover, the identification and existence of the entity type representing the instantiated class depends on the entity type representing the parameterized class. Therefore, the former entity type should be a weak entity and should totally participate in the relationship. For example, Figure 8 shows the mapping of an instantiates relationship to isa relationships.

4 Translation of a UML Class Diagram to an ER Diagram

The translation of a UML class diagram to an ER diagram requires the following steps:

1. For each regular class (not association class) in the class diagram, create a corresponding entity type. The name of the entity type is directly taken over from the class. The attributes of the original class are all carried over to the resultant entity type as well. An attribute with the name of the class name plus "-ID" is added to the entity type and is marked as the primary key. If the class is an instantiated class, mark the resultant entity type as a weak entity type. See Figure 1 and Figure 8 for examples.

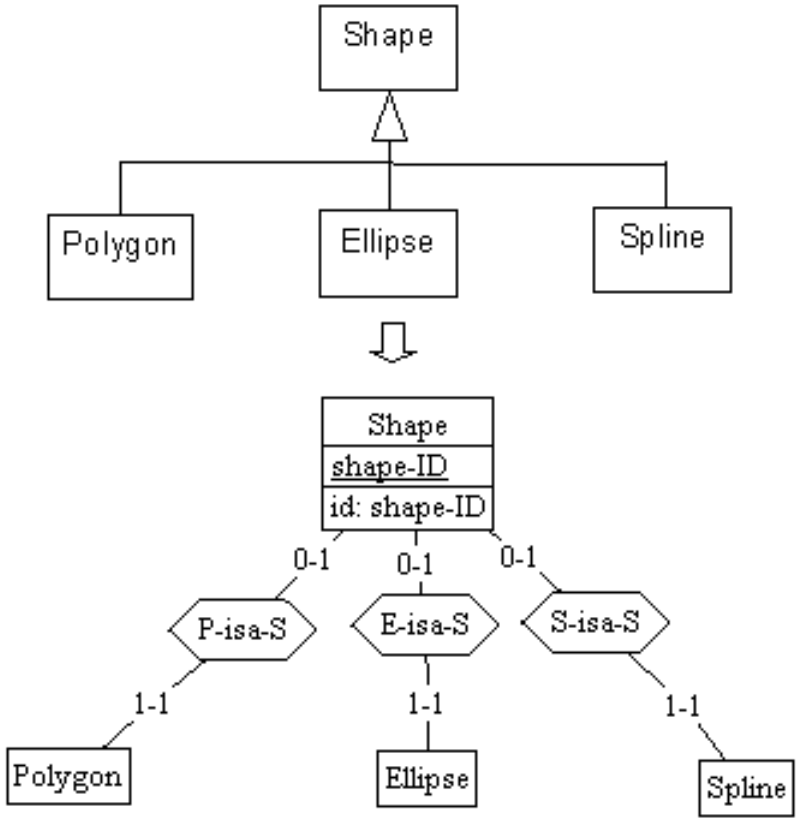

Figure 7: The mapping of generalization to isa relationships

2. For each inheritance relationship, create an isa relationship type between the resultant entity types of the two participating classes. See Figure 7 for an example.

3. For each instantiates relationship, create a weak isa relationship type between the resultant entity types of the two participating classes. See Figure 8.

4. For each aggregation relationship, create a relationship type between the resultant entity types of the two participating classes. Add the cardinality constraints to the resultant relationship properly. See Figure 6.

5. For each association, draw a relationship type among the entity types representing the participating classes. The association name is the name of the resultant relationship. If the association possesses an association class, add all the attributes of the association class to the resultant relationship type. See Figure 2.

6. For each association, do:

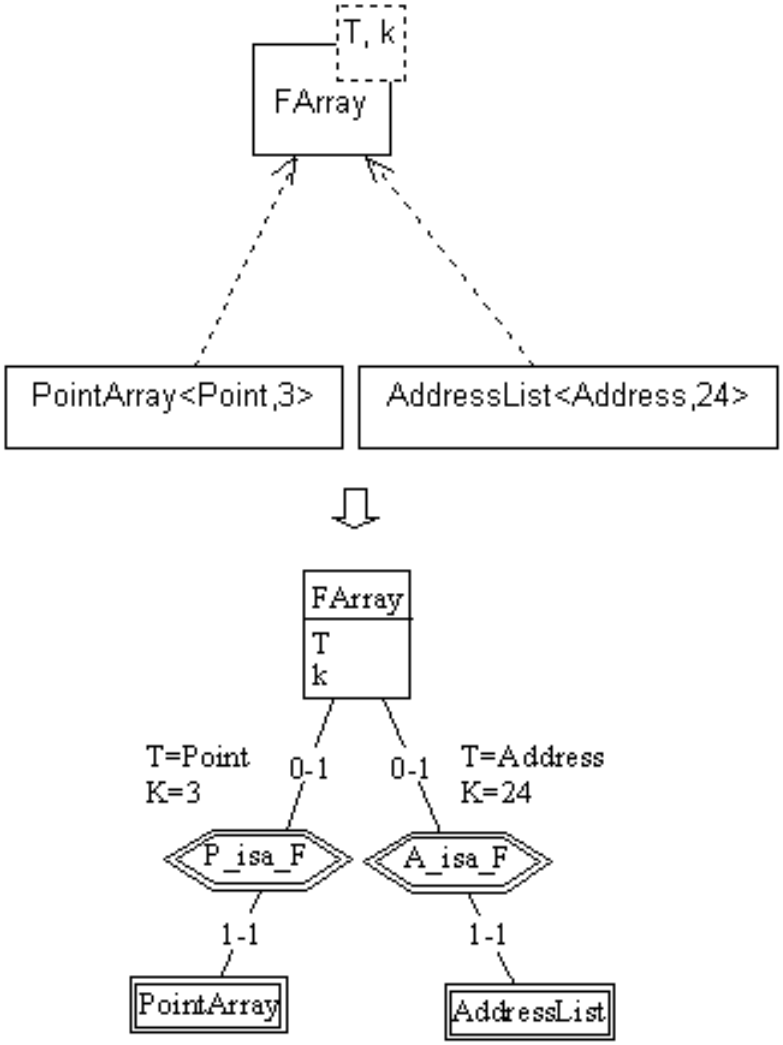

Figure 8: The mapping of instantiates relationship to isa relationships

- If there are any role names, then carry them over to the resultant relationship type. See Figure 2.
- If there is any cardinality specified and the association is not qualified, then put the specification on the resultant relationship type but exchange the positions of the specification. See Figure 2 and Figure 6.
- If a qualifier is defined for the association, depending on the cardinality specification of the qualified association, put the proper specification and attributes on the resultant relationship type and define the primary key for the relationship type. See Figure 3, Figure 4, Figure 5 for examples.

After these steps, an ER diagram equivalent to the original UML class diagram is created. However, some refinement must be done by the designer. For instance, the names of the relationships should be added if missing.

5 Mapping of ERM to UML

As Rumbaugh et al. (1991) pointed out, the OMT (Object Modeling Technique) is an enhanced form of ER that includes some new concepts (such as qualification). The UML is an enhanced form of OMT and thus an enhanced form of ER model. Therefore, the mapping of ER model to UML is quite straightforward. An Entity type can be mapped to a class. The attributes of the entity type are then carried over to be the attributes of the resultant class. A relationship type is mapped to an association. More accurately, the isa relationship type should be mapped to a generalization, an identifying relationship type to an uni-directional association. A relationship type with attributes must mapped to an association class. The role names of the relationship type become role names of the resultant association. The cardinality definitions in ERM are mapped to the multiplicity definitions in UML, but the specification positions are exchanged (cf. Figure 2).
In another words, to translate an ER diagram to a UML class diagram, the following steps are needed.

1. For each entity type in the ER diagram, create a corresponding UML class with the same name. The attributes of the entity type become attributes of the resultant class.

2. For each relationship type:

   - If it is an `isa` relationship, create a generalization between the classes representing the participating entity types.
   - If it is an identifying relationship type, create a uni-directional association between the classes representing the participating entity types.
   - Otherwise, create an association among the classes representing the participating entity types.

3. For each relationship type:

   - If there are any role names specified, add the role names to the resultant generalization/association as well.
   - If there are any cardinalities specified, add the specifications to the resultant association properly.

6 Conclusion

The UML is a semantically rich and expressively powerful modeling language. Many CASE tools will soon support it. The ER model is essentially *the* model used in database design, especially in relational database design. This paper has tried to define the mapping between the UML and the ER model. Due to the simplicity of the ER model, some of the UML concepts and notations, like operations, visibility of classes and some adornments attached to an association role, are ignored during the mapping. One solution to amend this is to map these concepts into constraints, comments or notes in ER model, so that the semantics of the UML model can be more or less preserved. Nevertheless, the essences of a UML class diagram can be sufficiently represented by an ER diagram. The translation of a UML class diagram to and from an ER diagram is very useful in database forward and reverse engineering.

More complete mappings will have to take advantage of dynamic features of relational DBMS, such as constraints, triggers, and stored procedures. These are, however, not easily represented in an ER context. This is another indication for the usefulness of a multi-model database design framework.

A CASE tool to automate the translation process is under construction. This CASE tool is intended to incorporate at least `Rational Rose` from the OO CASE tool area and the ER tools `Erwin` (Logic Works) and `DB-MAIN`. Our first goal is to provide automatic import and export facilities for UML diagrams to and from ER diagrams. Further work will have to deal with extensions of the ER-based representations to account for dynamic aspects (such as triggers and procedures). In parallel to that, we are also looking at target DBMS platforms different from SQL, namely ODMG's object databases (ODL, OQL).

References

BATINI, C., CERI, S., NAVATHE, S. (1992): Conceptual Database Design: An Entity-Relationship Approach. The Benjamin/Cummings Publishing Company, Inc., California.

BOOCH, G., RUMBAUGH, J., and JACOBSON, I. (1997): Unified Modeling Language, Version 1.0. Rational Software Corporation, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951 (USA), URL: `http://www.rational.com`.

BOOCH, G. (1994): Object-Oriented Analysis and Design with Applications, Second Edition. The Benjamin/Cummings Publishing Company, Inc., California.

CHEN, P. P. (1976): The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems, Vol.1, No.1, 9-36.*

ELMASRI, R. and NAVATHE, S.B. (1994): Fundamentals of Database Systems, Second Edition. The Benjamin/Cummings Publishing Company, Inc., California.

HAINAUT, J.-L., ENGLEBERT, V., HENRARD, J., HICK, J.-M., ROLAND, D. (1994): Database Evolution: the DB-MAIN Approach. In: P. Loucopoulos (Ed.):

Proceedings of the conference on the ER Approach – ER'94. LNCS, Springer-Verlag, Germany, 112-131.

JACOBSON, I., CHRISTERSON, M., JONSSON, P., and OVERGAARD, G. (1992): Object- Oriented Software Engineering. Workingham, England, Addison-Wesley Publishing Company.

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F., and LORENSEN, W. (1991): Object-Oriented Modeling and Design. Englewood Cliffs, New Jersey, Prentice-Hall.