Comparison and Evaluation of Methodologies for Transforming UML Models to Object-Relational Databases

Rajani Chennamaneni

Emanuel S. Grant, PhD.

Department of Computer Science
University of North Dakota
rajani@cs.und.edu

Abstract

Application and system modeling have become an important part of many software development projects. The Unified Modeling Language (UML) has been accepted as the general object-oriented modeling language for software systems. In relational database development entity-relationship models have traditionally been used for modeling such systems. It has been acknowledged that relational database management systems need a better representation of the real world than that obtained with the current tabular representation that are derived from the entity-relationship models. Object-oriented modeling is an effective mechanism for representing real world structures. There are many techniques for transforming UML models into object-oriented relational database systems. In this work we will compare, and evaluate methodologies for transforming UML class diagram (CD) models into object-oriented relational databases. This work will focus on the transformations of CDs into a set of statements in SQL:1999. This work will seek to identify the commonality and variability in the methodologies and advantages of the approaches.

Introduction

Application modeling has become important to many software development projects such as E-commerce and real-time systems. The Object Management Group [1] (OMG)'s Unified Modeling Language [2] (UML) has been accepted as the general object-oriented modeling language for software systems, and is supported by major corporations such as IBM®, and Oracle®. In relational database development entity-relationship (E-R) models have traditionally been used for modeling such systems. It has been acknowledged that relational database management systems need a better representation of the real world than that obtained with the current tabular representation that are derived from the E-R models. E-R diagrams are used specifically for data modeling and do not specify how to process the data. Object-oriented modeling is an effective mechanism for representing real world structures as it provides diagrams for modeling different aspects of the system. The UML supports object-oriented technologies and thus could be used for object-relational database modeling. The UML models both data and process, which allows the use of one notation for entire system modeling.

There are many proposals and techniques for extending and transforming UML models to express object-oriented relational database systems [10]. In this work we will demonstrate, compare, and evaluate two methodologies for transforming UML class diagram (CD) models into object-oriented relational databases. This work will focus on the transformations of UML CDs into a set of Data Definition Language statements in SQL:1999 [8]. The comparative analysis and evaluation of two methodologies will be accomplished by applying both methodologies to a common example CD.

One of the methodologies (Methodology 1) [3] being researched extends the UML in terms of stereotypes, tagged values, and constraints to enable the modeling of applications with object-relational database constructs.

The second methodology (Methodology 2) [4] uses standard UML CDs to model the applications. These CDs are converted into semantically equivalent ones by transforming all the roles into sub-classes and removing any cycles in the class diagram. The models are then converted to Nested Normal Form [7] (NNF) tables. In object-relational databases tables may not be in first normal form [9], because they contain nested relations. Thus several NNF for nested relations have been defined to eliminate redundant data.

Our work will seek to identify the commonality and variability in the two methodologies and determine the advantages of the two approaches in using UML to model object-relational databases.

This paper is organized as follows: The next section summarizes the background concepts needed for the research. The next two sections summarize the two methodologies using a common example class diagram. The final section gives the expected results and conclusions.

Background

The Unified Modeling Language

The Unified Modeling Language (UML) is a graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [1]. The UML's architecture is based on a four-layer meta-model structure, which consists of meta-meta-model, meta-model, user-defined model, and objects. The functions of these layers are summarized in Table 1 [1].

Table 1

Layer	Description	Example
meta-meta-model	The infrastrure for a meta-modeling architecture. Defines the language for specifying meta-models.	MetaClass, MetaAttribute, MetaOperation
meta-model	An instance of a meta-meta-model. Defines the language for specifying a model.	Class, Attribute, Operation, Component
user-defined model	An instance of a meta-model. Defines a language to describe an information domain.	Flight, Airline, Customer, TicketPrice
user objects	An instance of a model. Defines a specific information domain.	Bowing 747, Delta, John, $200

The UML is at the meta-model layer, which is an instance of the meta-meta-model layer. The UML meta-model is described using Abstract Syntax, Well-formedness rules, and Semantics. The abstract syntax is given as models described in UML class diagrams and natural language. The well-formedness rules are given in a formal language and natural language. The semantics are mainly given in natural language, but may include some additional formal notation [1].

The UML provides extension mechanisms that enable new kinds of modeling elements to be defined and also enable the attachment of information to new modeling elements. This is accomplished by using stereotypes, constraints and tagged values. Stereotypes are used to extend the vocabulary of the UML, allowing the creation of new kinds of modeling elements that are derived from existing ones, which are specific to a particular class of problem and has its own properties, semantics, and notation. A stereotype is represented as a name enclosed by guillmets (<<stereotype>>). A tagged value is used to extend the properties of a UML modeling element, allowing for the creation of new information in the element's specification. A constraint is used to restrict the semantics of a UML modeling element allowing for new rules or modification of existing ones.

The UML defines several graphical diagrams in terms of the views of a system. These diagrams are class diagram, use-case diagram, state-chart diagram, activity diagram, sequence diagram, collaboration diagram, component diagram, and deployment diagram.

Details of the diagrams except class diagram are outside the scope of this paper. Class diagrams are described in the next section.

Class Diagrams

A class diagram illustrates the graphical view of the static structure of a system. It is a collection of classes, interfaces, and their relationships. A class is represented graphically as in Figure 1:

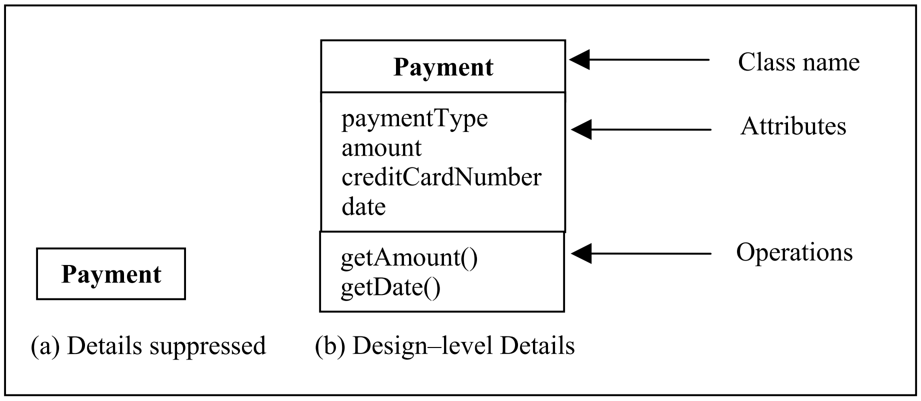

Figure 1: Representation of a Class

Relationships are mainly classified into association, aggregation, composition, and generalization. An association is a generic relationship between two classes and is shown as a thin line connecting two classes. Association has cardinality and role (optional) on the each end and a label (optional) for the relationship. Role is the named end of an association to indicate its purpose. Cardinality indicates the number of objects from each class that may participate in the relationship. Aggregation indicates whole-part relationship between two classes, shown as a line with a hollow diamond on the whole class end. Composition is a strong form of aggregation where the "whole" is completely responsible for its parts and each "part" class is only member of one "whole" class, shown as a line with a filled (black) diamond on the whole class end. Generalization is a taxonomic relationship between a more generalized class and specialized classes. It represents super-class (generalized) and subclass (specialized) relationship between the classes, shown as a line with a hollow triangle on the generalized class end. The notation to model these relationships is shown in the Figure 2. Table 2 shows the types of cardinalities, which indicate the multiplicities.

Table 2: Types of Cardinalities

0..*	Zero or more
0..1	Zero or one
1..*	One or more
1	One only
n	n only (n > 1)
0..n	Zero to n (n > 1)
1..n	One to n (n > 1)

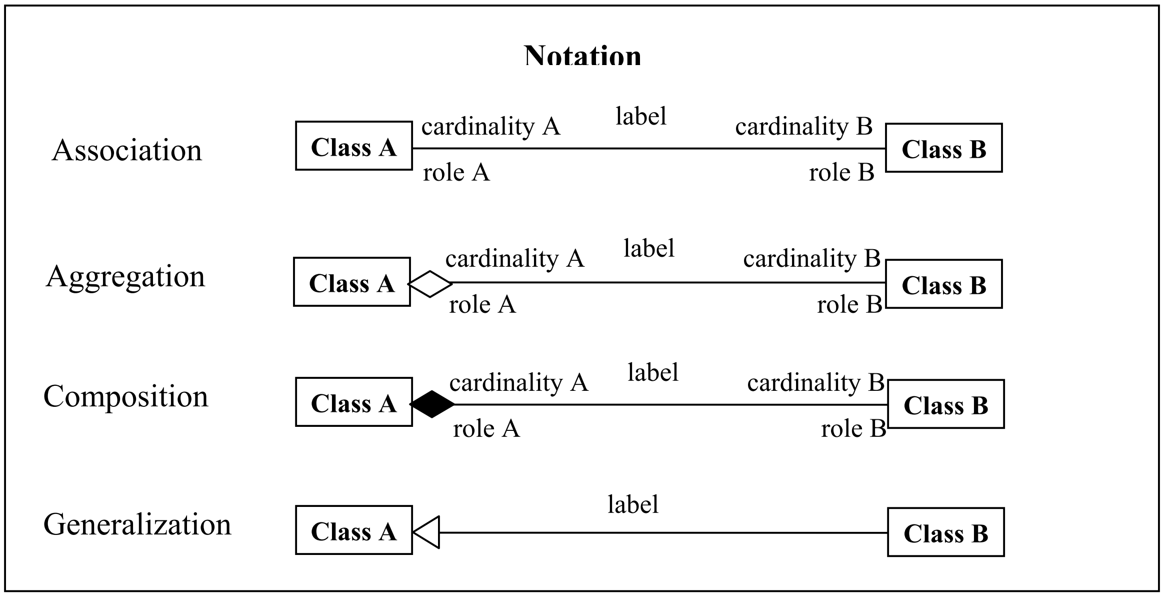

Figure 2: Notations of relationships

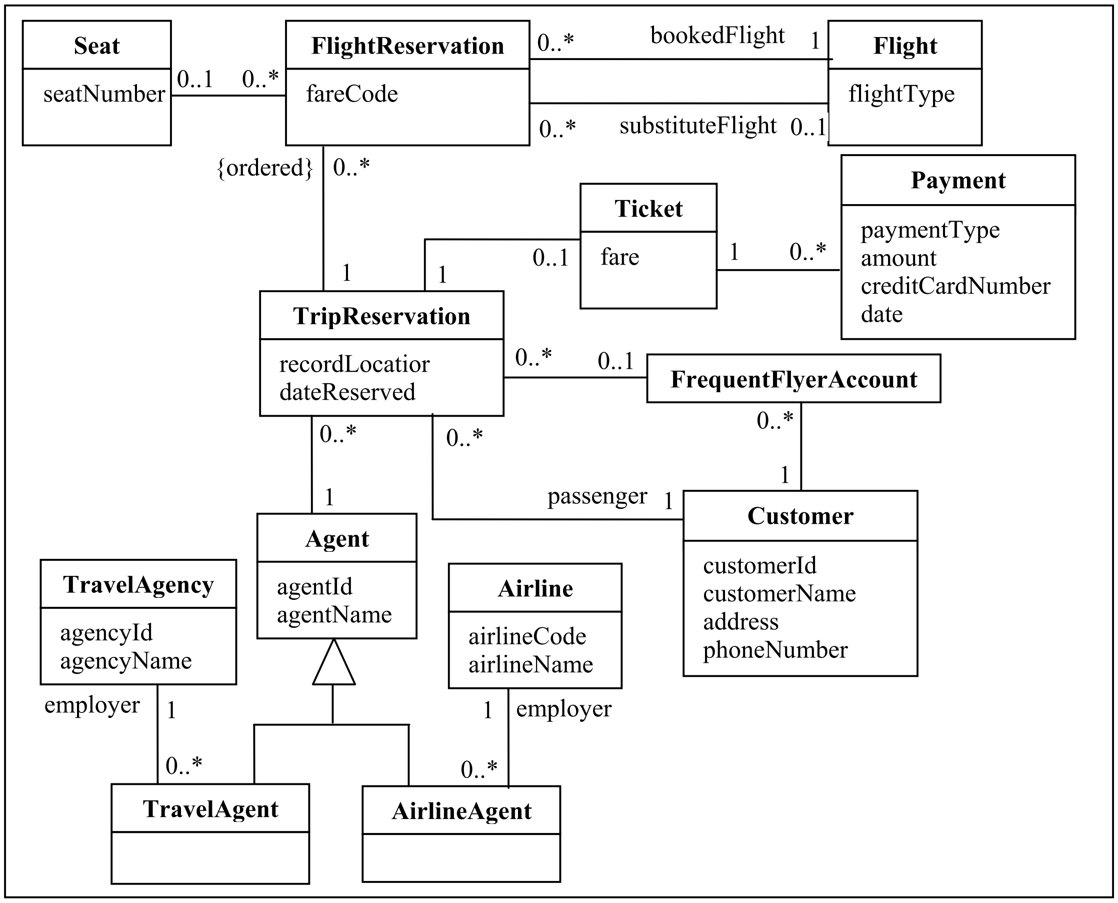

Figure 3: Class diagram of Airline Reservations

Figure 3 is an analysis-level class diagram (methods suppressed) of the Airline Reservation system obtained from [6] and adapted for UML, which will be used as the sample for analyzing and evaluating the methodologies. This class diagram illustrates classes, attributes, associations, multiplicities, generalization and specializations, and roles. A trip reservation consists of a sequence of flight reservations and each flight reservation refers to a specific flight, sometimes it may also have a substitute flight and also may refer to a seat. Each trip reservation is reserved on particular date and is distinguished by record locator for further tracking, as one may not purchase the ticket on the same day of reservation. A customer makes a trip reservation and may use a frequent flyer account. The customer will receive the ticket only after the purchase has been made. Multiple payments can be made with one purchase. An agent, who works for a travel agency or an airline, may reserve a trip.

Normal Forms

Normalization is the process of analyzing the relationships between various elements of the relational database and arranging the data efficiently in order to increase the consistency of the data. This is accomplished by applying various normal forms [9] to the tables. *First normal form* states that each row-column combination in a table must contain a single value rather than a set of values. *Second normal form* states that a table should be in first normal form and all attributes of the table must depend on the entire primary key. *Third normal form* states that a table should be in second normal form and no attribute should transitively depend on the primary key. As tables satisfy higher normal forms, they are more consistent and store less redundant data.

Tables of object-relational databases may not be in the first normal form as they contain nested relations and abstract data types with set of values as opposed to single value for the row-column combination. For object-relational database design several normal forms for nested relations have been defined, which are called nested normal forms, which helps reducing redundant data values in object-relational databases. In [7] nested normal forms are discussed in detail.

Methodology 1 [3]

Methodology 1 implements the transformation by using the UML extension mechanism (stereotypes, tagged values and constraints) to define a new set of UML model elements that represent object-relational database concepts. These new model elements are then used to design the object-relational databases. Table 3 illustrates some of the extensions that are proposed for relational databases. Table 4 lists the extensions proposed for object-relational databases which can support collection types, methods etc. in methodology 1. This methodology proposes some rules as guidelines for object-relational database design given in Table 5.

Application of the extension mechanisms on the object-relational database design model for the section of the CD in Figure 3 that contains TripReservation, Agent, TravelAgent, AirlineAgent, TravelAgency and Airline classes of the Airline Reservation system results in the stereotyped equivalent CD of Figure 4, which will then be transformed into SQL:99 statements.

Table 3: Stereotypes for database design

SQL:1999	**UML element**	**Stereotypes**
Database	Component	<<Database>>
Schema	Package	<<Schema>>
Tablespace	Component	<<Tablespace>>
Table	Class	<<Table>>
View	Class	<<View>>
Index	Class	<<Index>>
Column	Attributes	<<Column>>
Primary Key	Attributes	<<PK>>
Foreign Key	Attributes	<<FK>>
Multivalued Attribute	Attribute	<<AM>>
Calculated Attribute	Attribute	<<AD>>
Composed Attribute	Attribute	<<AC>>
NOT NULL Restriction	Attributes	<<NOT NULL>>
Unique Restriction	Attributes	<<Unique>>
Trigger	Restriction	<<Trigger>>
Restriction	Restriction	<<Check>>
Stored Procedure	Class	<<Stored Procedure>>

Table 4: Stereotypes for object-relational databases

SQL:1999	**UML element**	**Stereotypes**
Structured Type	Class	<<udt>>
Typed Table	Class	<<persistent>>
Composes	Association	<<composes>>
REF Type	Attribute	<<ref>>
Array	Attribute	<<array>>
Row Type	Attribute	<<row>>
Redefined Method	Method	<<redef>>
Deferred Method	Method	<<def>>

Table 5: Guidelines for object-relational databases

UML	**SQL:1999**
Class Class Extension	Structured Table Typed Table
Attribute Multivalued Composed Calculated	Attribute Array ROW/Structured Type in column Trigger/Method
Association One-to-One One-to-Many Many-to-Many	 REF/REF REF/ARRAY ARRAY/ARRAY
Aggregation	Array
Generalization	Types/Typed Tables

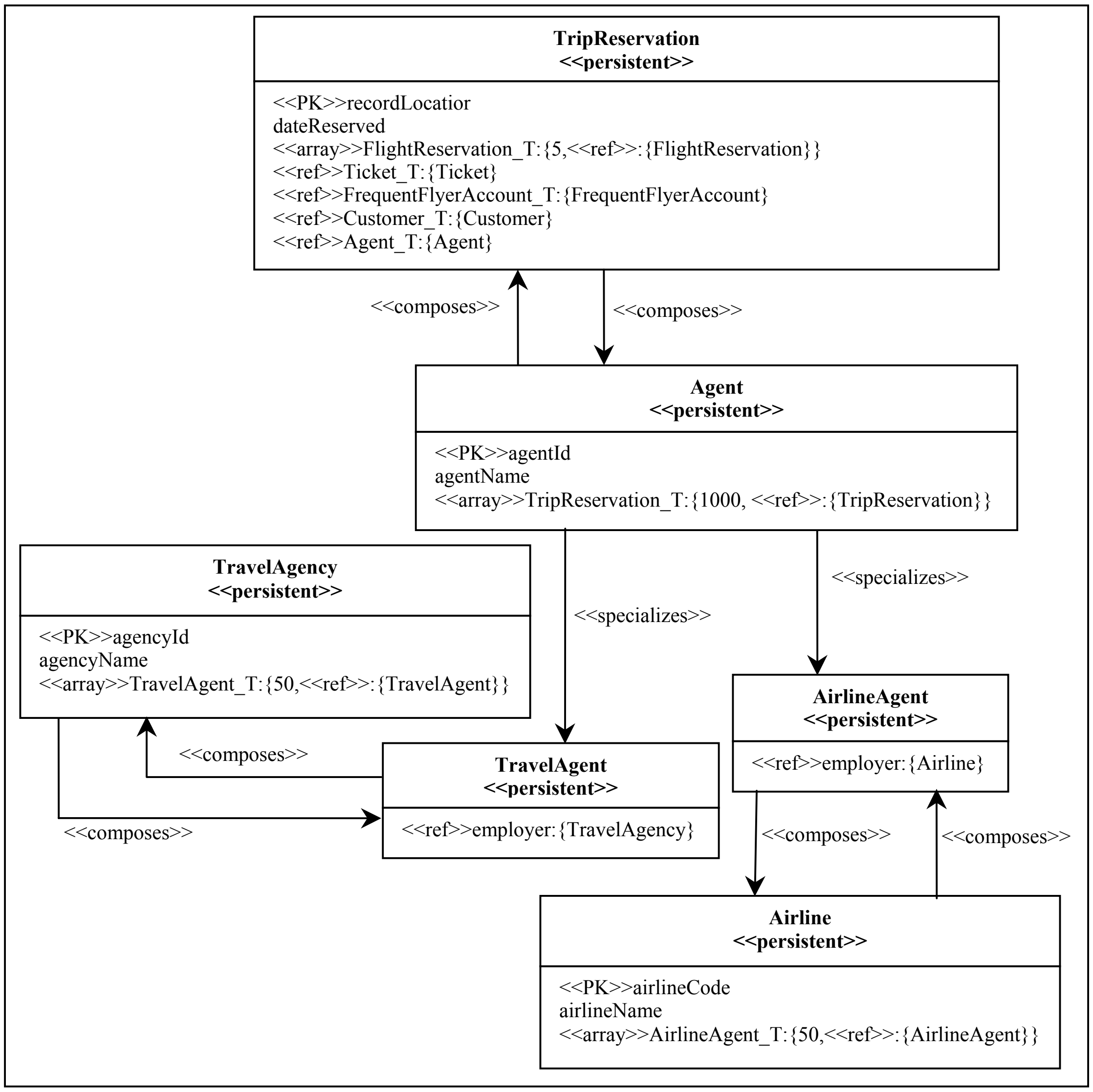

Figure 4: UML design for Airline Reservations using Methodology 1

Figure 4 presents the UML class diagram that uses the extensions proposed for object-relational databases. The one-to-many association between TravelAgency and TravelAgent classes is modeled using composes stereotype and REF/ARRAY. Table 6 illustrates the implementation of the model in SQL:1999 as types and tables.

Table 6: SQL:1999 statements

```
CREATE OR REPLACE TYPE TripReservation AS OBJECT
(recordLocator VARCHAR(20),
 dateReserved DATE,
 FlightReservation_T REF(FlightReservation) ARRAY[5],
 Ticket_T REF(Ticket),
 FrequentFlyerAccount_T REF(FrequentFlyerAccount),
 Customer_T REF(Customer),
 Agent_T REF(Agent) );
```

```
CREATE OR REPLACE TYPE Agent AS OBJECT
(agentId VARCHAR(10),
 agentName VARCHAR(20),
 TripReservation_T REF(TripReservation) ARRAY[1000] );

CREATE OR REPLACE TYPE TravelAgency AS OBJECT
(agencyId VARCHAR(10),
 agencyName VARCHAR(20),
 TravelAgent_T REF(TravelAgent) ARRAY[50] );

CREATE OR REPLACE TYPE TravelAgent UNDER Agent AS
(employer REF(TravelAgency) );
    ….

CREATE TABLE T_TripReservation OF TripReservation;

CREATE TABLE T_Agent OF Agent;

CREATE TABLE T_TravelAgency OF TravelAgency;

    ….
```

Methodology 2 [4]

Methodology 2 uses UML design to generate NNF nested tables from a class diagram by applying *algorithm 0* of [4]. *Algorithm 0* converts the class diagrams to semantically equivalent ones and then *algorithm 1* of [7] is used get the NNF nested tables. These nested relation schemes are used for the implementation of SQL:1999 statements for object-relational databases.

Figure 5 presents the semantically equivalent class diagram of the section of the Airline Reservation System containing TripReservation, Agent, TravelAgent, AirlineAgent, TravelAgency and Airline classes. The nested tables for the model of Figure 5 are given as follows:

*TripReservation:[recordLocator, dateReserved] (FlightReservation:[fareCode] ...)**
    Ticket (Payment) FrequentFlyerAccount Customer(FrequentFlyerAccount)* Agent)*
*Agent (TripReservation)**
*employer_TA (TravelAgent)**
*employer_AL (AirlineAgent)**
...
stand alone classes:
TravelAgent Employer_TA
AirlineAgent Employer_AL
TravelAgency:[agencyId, agencyName]
Airline:[airlineCode, airlineName]
...

Table 7 presents the SQL:1999 implementation of the model in Figure 5.

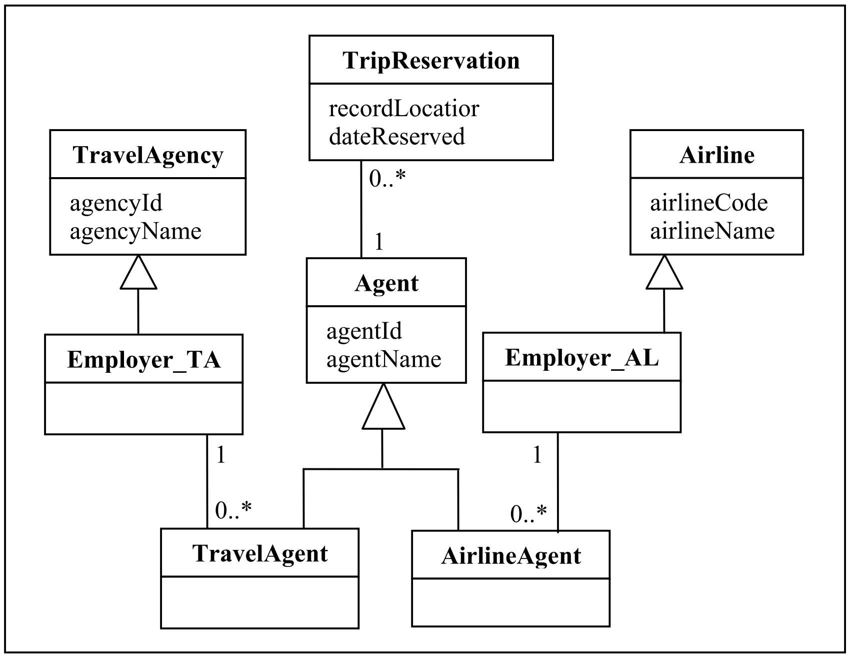

Figure 5: UML design for Airline Reservations using Methodology 2

Table 7: SQL:1999 Statements

```
CREATE OR REPLACE TYPE TravelAgency AS
(agencyCode VARCHAR(10),
 agencyName VARCHAR(20) )

CREATE OR REPLACE TYPE Employer_TA AS
(TravelAgent_T REF(TravelAgent) array[n] )

CREATE OR REPLACE TYPE TravelAgent AS
(Employer_TA_T REF(Employer_TA) )

CREATE OR REPLACE TYPE Agent AS
(agentId VARCHAR(10),
 agentName VARCHAR(20),
 TripReservation_T REF(TripReservation) array[n] )

CREATE OR REPLACE TYPE TripReservation AS
(recordLocator VARCHAR(10),
 dateReserved Date,
 FlightReservation_T REF(FlightReservation) array[n],
 Ticket_T REF(Ticket),
 FrequentFlyerAccount_T REF(FrequentFlyerAccount) array[n],
Customer_T REF(Customer),
Agent_T REF(Agent) )
```

Conclusion

This research work is at an early stage and presently we are reviewing a number of published works on the two transformation methodologies being evaluated. Once the review has been completed we will proceed with the application of the methodologies on the example CD of Figure 3.

Not all aspects of each methodology will be applicable to the Figure 3 example CD. In Methodology 1 there was no approach of normalization for the database, because UML modeling takes care of most of the design issues. In [3] there is no stereotype defined for the *generalization* association, which we think is necessary. Thus, we will introduce a stereotype for generalization (<<specializes>>) in our application of Methodology 1 to the Figure 4 CD.

In Methodology 2 the first activity is the elimination of cycles in the CD. A cycle occurs when it is possible to navigate from one class (the *source*) to another (the *destination*) along two or more association paths and select the same set of objects in the *destination*. Such occurrences usually result from poor design of the model. In our example, by design, there are no cycles.

As an additional output of our evaluation of the methodologies we hope to be able develop a *hybrid* UML CD to SQL:1999 transformation that amalgamates the best of the two methodologies, and thus define a more efficient transformation technique. The ultimate goal of this work is to broaden the use of the UML in software development, by making it more usable in domains outside of the more popular ones in which it is currently being used.

References

1. *http://www.omg.org*, Object Management Group.
2. *http://www.uml.org*, Unified Modeling Language.
3. E. Marcos, B. Vela, and J.M. Cavero (2001). *Extending UML for Object-Relational Database Design*.
4. W. Y. Mok, and David P. Paper (2001), *On Transformations from UML Models to Object-Relational Databases*.
5. *OMG Unified Modeling Language Specification*, March 2003, version 1.5.
6. M. Blaha, and W. Premerlani (1998). *Object-oriented Modeling and Design for Database Applications*. Prentice Hall.
7. W.Y. Mok (2002). *A Comparative Study of Various Nested Normal Forms*.
8. A. Eisenberg, and J. Melton (1999). *SQL:1999, formerly known as SQL3*.
9. D. Maier (1983).*The Theory of Relational Databases*. Computer Science Press.
10. M. Stonebraker, and P. Brown (1999). *Object-Relational DBMSs: Tracking the Next Great Wave*. Morgan Kauffman.