Integrating ERD and UML Concepts When Teaching Data Modeling

Traci A. Carte
Michael F. Price College of Business
University of Oklahoma
Norman, OK 73019, USA
tcarte@ou.edu

'Jon (Sean) Jasperson
Mays School of Business
Texas A&M University
College Station, TX 77843, USA
jon.jasperson@tamu.edu

Mark E. Cornelius
IT Services, Anadarko Petroleum Corporation
The Woodlands, TX 77380, USA
mark_cornelius@msn.com

ABSTRACT

In this paper, we describe a teaching approach that evolved from our experience teaching in both the traditional database and systems analysis classes as well as a number of semesters spent team-teaching an object-oriented systems development course. Fundamentally, we argue that existing knowledge of structured systems development can and should inform our teaching processes when teaching object-oriented systems development techniques. We draw from an anecdotal industry example provided by one of our former students to illustrate the value of this approach given our perception that there is a need in practice today to easily shift from structured to object-oriented thinking.

Keywords: Unified Modeling Language, Data Modeling, Database Course

1. INTRODUCTION

There seems to be growing interest in adopting the Unified Modeling Language (UML) within information systems (IS) curricula, and some authors of database texts have expressed interest in changing their widely adopted books to include the UML notation when representing data models. One might argue that notation is only syntax; therefore, a change in notation should not require a change in the content or approach used in teaching data modeling techniques. However, the interest in UML suggests consideration of a more fundamental question: *Should we rethink the processes taught in our database courses to more closely align the way we think about data with the way applications are developed?*

Currently, most database courses use entity-relationship diagram (ERD) techniques for data modeling. The traditional ERD has a rich theoretical basis and is specifically intended for modeling relational database structures (Chen 1976, 1977; Date 1986; Martin 1982). Clear guidance exists in many academic and practitioner books about how to use this method to develop conceptual models and transition them to logical forms (including normalization practices) and physical forms that are focused on tuning for performance (Chen 1977; Hoffer, Prescott, and McFadden 2005; Martin 1982). Further, some empirical studies suggest ERDs are often more correct and easier to develop than corresponding object-oriented (OO) schemas (Shoval and Shiran 1997).

Advocates of the UML suggest that the class diagram should replace the ERD notation and approach to data modeling. Class diagrams provide the same opportunity to document data and their relationship as ERDs do. In addition, class

diagrams provide for the capture of operations. This allows for the modeling of relational data but also provides rich support for object-oriented implementations in the form of OO program languages (i.e., JAVA) as well as a more component-based approach (i.e., J2EE). Moreover, the UML includes mechanisms for modeling behavior, and the acceptance of the UML as an Object Management Group (OMG) standard provides wide support in industry for using the UML, especially for the design of object-oriented software (Halpin and Bloesch 1999). While some might advocate ERD versus UML as contrasting methodological perspectives, we see advantages of teaching both methods to information systems students in an integrated fashion.

Understanding the implications of the adoption of UML notation in the database class cannot be undertaken without consideration of the other courses in the IS curriculum as well as indications of the future of practice. In a recent panel discussion focused on database course content (Vician et al. 2004), the interrelationships among courses in the IS curriculum was discussed. Specifically, the panelists advised that when the core systems analysis course uses an object-oriented analysis and design approach, using a more traditional approach in the database class may lead to inconsistencies that hinder student learning.

A review of the trade literature suggests that transitions to object-oriented databases may be on the horizon; with Oracle and IBM indicating willingness to "go beyond classical relational dogma" (Monash 2005; p 26). Some organizations have adopted object-oriented databases for storing important corporate data (Lai 2005). Further, as more applications are developed in OO languages such as Java or C++ while utilizing data stored in a relational database, IT professionals are left to write custom code to access the data or rely on object-relational mapping (Walsh 2005). Growing interest among firms such as Oracle as well as the ubiquity of relational databases suggests that mapping OO applications to relational data storage is the likely immediate experience current MIS students can expect in practice (Krill 2005).

Should we rethink the processes taught in our database courses to more closely align the way we think about data with the way applications are developed? This is our fundamental question. In this paper, we draw from our experience in the classroom and in practice to make recommendations for how the UML notation can be integrated with ERD when teaching data modeling.

2. BACKGROUND

In Fall 2000, the two academic authors began team-teaching an object-oriented systems development (OOSD) course. At that time, we had a number of years experience teaching in the traditional database and structured systems analysis and design courses.

As we began designing and teaching this OOSD course we wanted to allow the students to form their own opinions regarding the efficacy of the OO approach to systems development and using the UML. Therefore, in our first semester teaching the course, when discussing class diagrams, we asked our students to read the technical specifications regarding class diagrams directly from the UML specifications (at that time, UML version 1.2 was the standard accepted by the OMG). In class, we spent time discussing and defining classes, objects, state, behavior, associations, association classes, multiplicity, and etc. as recommended by Fowler and Scott (2000, chapter 4). We worked a simple example class diagram in the classroom and assigned the students to develop their own class diagrams for a more complex case.

All students in the class had previously completed a traditional, database course (where they learned the ERD approach to data modeling) and a structured systems development course (where they were exposed to extensive ERD and data flow diagramming exercises). Thus, we expected the students to complete class diagrams for the assignment with relative ease.

We were surprised to find that the students struggled to complete this assignment. The assignments submitted for grading by the students contained much variation across students. Most students struggled with identifying classes to include in the diagram. Some students included user interface classes, while others did not. Many students experienced difficulty in capturing the basic data needs for the case (i.e., applying the data modeling skills learned in previous classes).

As we discussed the performance on this assignment with the students enrolled in the course, we realized that in our attempt to faithfully apply the UML to a modeling situation and to avoid introducing the biases from our structured development backgrounds into the course, we failed to help the students understand how to leverage the skills and knowledge they already possessed. The students observed that they had not understood that a class diagram was related to data modeling. Another difficulty experienced by the students was identifying the behavior that should be included in the class diagram as operations. The students indicated they were unsure about which operations to include in the diagram and even if they were sure an operation should be included, they were unsure which class should have the operation assigned to it.

After this initial attempt at teaching and applying the UML, we reevaluated our position and approach. We concluded that many, if not most, systems developed in a business setting would need to have some interaction with and use of data (likely stored in relational databases). Thus, when students use the UML to model business systems, they should realize that the class diagram should represent the data model for the new system. Furthermore, we observed that asking the students to focus on both the state and behavior properties of classes at the same time added complexity, which resulted in confusion among the students. Therefore, we redesigned the course content and our teaching approach to enable us to emphasize the following ideas when using the UML for systems development projects: 1) logical class diagrams need to be created that represent the data model for the business system to be

developed and 2) the state and behavior properties of classes can be considered separately when creating initial logical class diagrams for a business system.

In the next section, we present a short teaching case, our recommended development process, and the resulting UML diagrams which show the result of our learning.

3. CLASSROOM EXAMPLE

To illustrate our recommended approach, we use the case scenario presented in Figure 1. In developing a system to support Jason's Video, we recommend the following development sequence to model this problem using UML diagrams:

1. Create Use Case Diagram to set system scope
2. Create Class Diagram that is identical to an ERD
3. Create Sequence Diagram for each use case
4. Use the messages from the Sequence Diagrams to add operations to classes in Class Diagram

Jason's Video
Jason's Video is a small neighborhood video rental store. The only products available at Jason's Video are movies to rent. Jason requires all customers to pay cash for movie rentals. When a customer rents a movie from the store for the first time, Jason issues the customer a membership card. For future rentals, the customer presents his/her membership card with the movies to rent.

Jason keeps track of basic contact information for each customer. In addition, he tracks each movie by title, MPAA rating, media type (VHS or DVD), genre, and acquisition date. Jason would like to create a simple system to track movie rentals. Movies that Jason has owned for more than 6 weeks can be rented for 1 week. All other movies are considered "recent arrivals" and can only be rented for 2 days

Figure 1. Example Case Scenario

For simplicity, we only discuss two major use cases of the new system for Jason's Video: 1) a customer rents a movie and 2) a customer returns a movie. Figure 2 contains a use case diagram for these two scenarios. The development of the use case details is beyond the scope of this paper; as such we will not elaborate on this process.

After a student develops the use case diagram, we encourage him/her to create a class diagram that contains state information only. A class diagram with state information only is very similar to a logical data model or entity-relationship diagram. At this point in modeling the new system, for the class diagram, we focus on the data that will be physically stored and its interrelationships. Thus, classes that would represent aspects of the user interface are not represented. Further, we discourage students from adding behaviors to the classes at this point in time. Instead, we encourage the student to concentrate on modeling the data for the case. Given these guidelines, we are essentially encouraging students to draw an ERD using the UML notation.

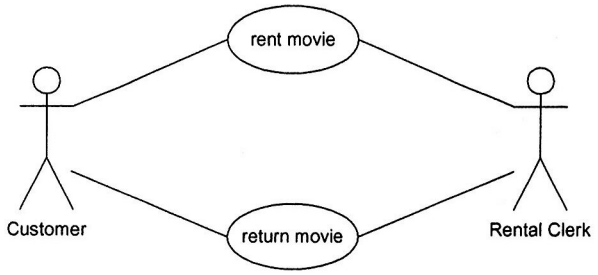


Figure 2. Jason's Video -- Use Case Diagram

For the Jason's Video case, we would model three classes, (e.g., movie, rental, and customer) and two associations (see Figure 3). As can be seen in this model, implementation in a relational environment would not be a problem. Further, the student could easily apply the process knowledge he/she has about constructing ERDs to developing a class diagram that well represents the data and relationships.

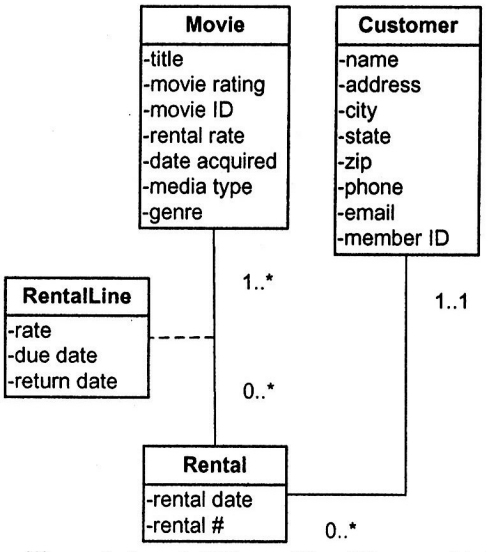


Figure 3. Jason's Video -- Class Diagram (state only)

As mentioned previously, we did a non-scientific experiment in the classroom with adopting a new paradigm. In our first semester, we made no conceptual link between data models and class diagrams, and we encouraged the students to think of this as a paradigm shift. In later semesters, we explicitly linked the thought process of data modeling with the construction of class diagrams. We believe learning was facilitated by the linkage. Similar observations have been made when project teams in practice work to make the transition to OO analysis and design (Shaft and Jasperson 2003).

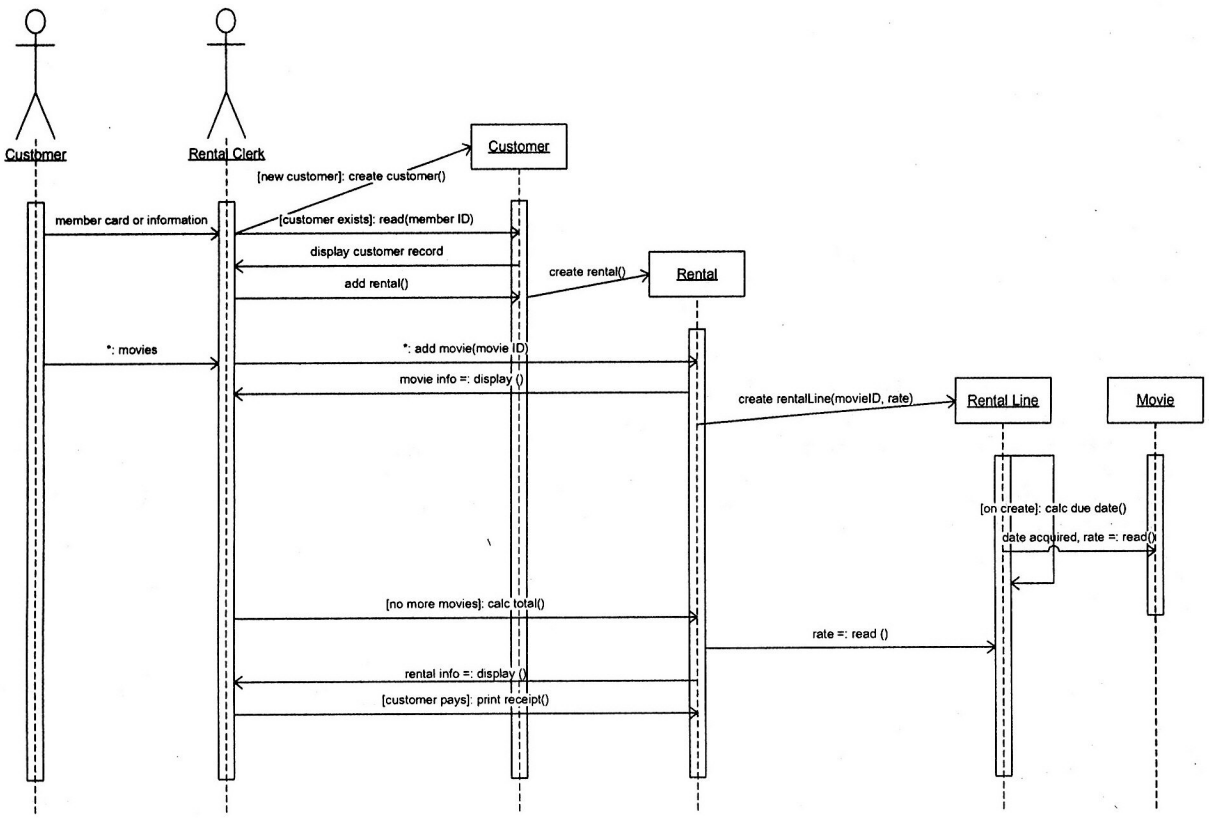
The next step in the development sequence is to create a sequence diagram for a generic scenario for each use case in the use case diagram. As with the class diagram, when creating the sequence diagrams, we recommend ignoring the user interface classes. These classes and messages passed among them can be added when the system gets closer to implementation.

One approach we found useful in helping students understand sequence diagrams is the use of a CRC card exercise (Beck and Cunningham 1989; Bellin and Simone 1997). In this exercise, index cards are created -- one for each class in the class diagram. State information is listed. Students are then asked to role play, pretending to be one of the classes and/or a single object. This exercise is particularly useful in helping the students understand which class can/should be responsible for various behaviors based on state information and associations. After role playing the use case in the classroom, we draw from the in-class role play to create the sequence diagram. Figure 4 contains the sequence diagram for the rent movie use case. Figure 5 contains the sequence diagram for the return movie use case.

After completing the sequence diagrams, we use the messages from the diagram to update the class diagram to reflect behavior (i.e., add the operations). To add operations to the class diagram, we consider each class represented in the sequence diagram and examine the messages that point to that class (i.e., messages that are received by that class). To receive a message, the class must have an operation that corresponds with the message it receives.

By default, all classes have create, read, update, and delete (CRUD) operations. Therefore, in creating the sequence diagram, we require that CRUD messages be included in the sequence diagram using these keywords as names for the messages. For example, see the "[new customer]: create customer()" message in Figure 4. Because all classes are able to perform all CRUD operations, we do not add CRUD operations to the class diagram.

For other messages passed to a class in the sequence diagram, the class diagram must be updated to reflect an operation that enables the class to perform the requested behavior. For example, consider the "add rental()" message to the customer class in Figure 4. For the rental clerk agent to send a message to the customer class entitled "add rental()" the customer class must have an operation "add rental()." Thus, we add the "add rental()" operation to the Rental class in the class diagram. To complete this iteration of the class diagram, all non-CRUD messages in each sequence diagram must be added as operations to the appropriate classes in the class diagram. Figure 6 contains the updated class diagram.



Legend
* indicates messages that iterate
[ ] contain conditions that must be true for message to be activated
= data items to the left of the = represent items returned by the message
( ) data items contained in the message ( ) represent items that are passed as inputs with the message
: separates output parameters and conditions from message name
display () message used to suggest future user interface classes

Figure 4. Jason's video -- Sequence Diagram for Rent Movie Use Case

This approach was successfully used and refined over the course of four subsequent semesters. In these semesters, the students had fewer issues and problems developing "proper" class diagrams for assignments and in testing situations. The students were able to apply their previously learned data modeling skills. Furthermore, they responded well to treating state and behavior separately during the logical design of a business system. Lastly, the students were better able to understand the behavior property of classes and were more confident in their ability to assign operations to classes in a class diagram.

4. DISCUSSION

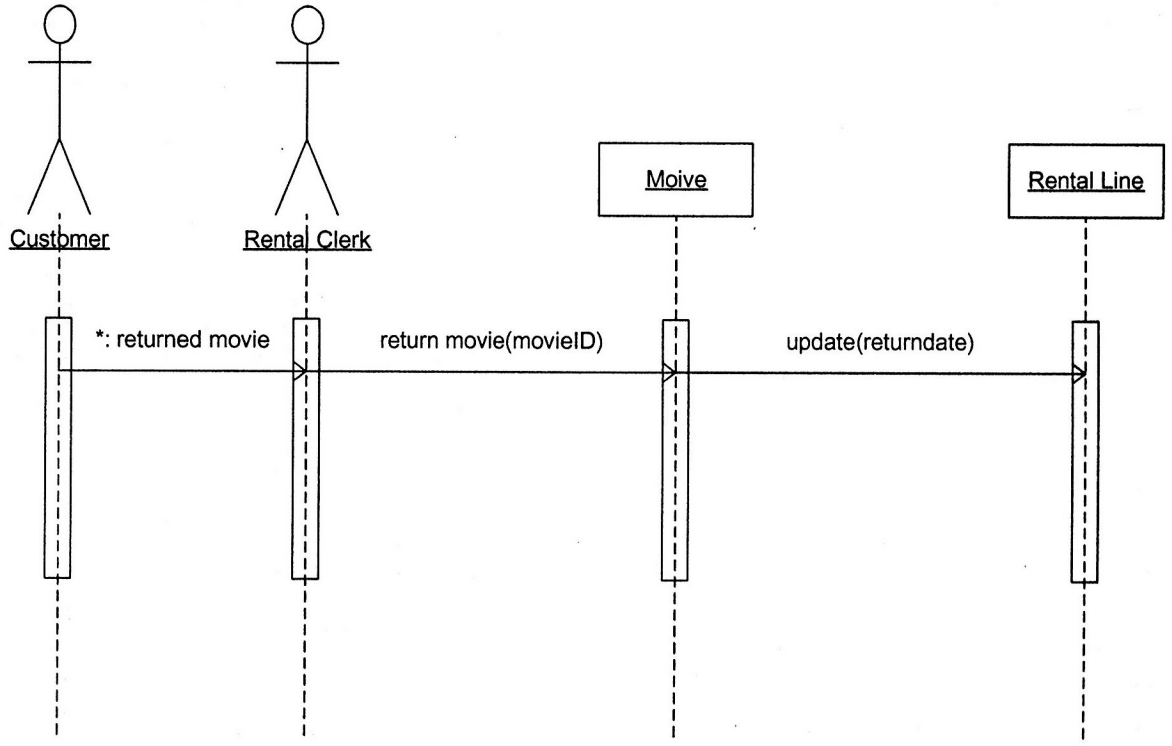
Should we rethink the processes taught in our database courses to more closely align the way we think about data with the way applications are developed? Our answer to this question is yes. As will always be the case in our discipline, the process and techniques we teach are always shifting to keep up with changes in practice. Where we may disagree with others is in the nature of the change. We articulate a teaching process that incorporates the UML syntax as well as the encapsulation of state and behavior, but we suggest this can be done without a revolution.

While we recognize that our experience is non-scientific, we believe our approach has merit and is supported by prior research and current industry practice. In this section, we review existing research and provide an anecdotal example from industry that lends credence to our recommendations.

4.1 Previous Research

A review of the literature suggests the use of OO design and development practices are moving to the main stream. For instance, one can find claims such as:

- the UML is becoming widely used for software and database modeling (Halpin and Bloesch 1999), and
- Object-Oriented (OO) systems development is a "hot topic" among computing professionals and academics (Johnson and Hardgrave 1999).



Legend

* indicates messages that iterate
[ ] contain conditions that must be true for message to be activated
= data items to the left of the = represent items returned by the message
( ) data items contained in the message ( ) represent items that are passed as inputs with the message
: separates output parameters and conditions from message name
display () message used to suggest future user interface classes

Figure 5. Jason's Video -- Sequence Diagram for Return Movie Use Case

However, the shift to OO techniques receives a more mixed review. While advocates of OO design proclaim advantages such as conceptual clarity and better alignment with the conceptual space (Pancake 1995; Rosson and Alpert 1990), critics claim it is conceptually complex, difficult to use, and difficult to learn (Sheetz et al. 1997). Further, shifts to OO analysis and design (OOAD) are likely more difficult than shifts to OO programming (OOP) (Sircar, Nerur, and Mahapatra 2001). Empirical studies indicate that ERDs are often more correct and easier to develop than corresponding OO diagrams (Shoval and Shiran 1997).

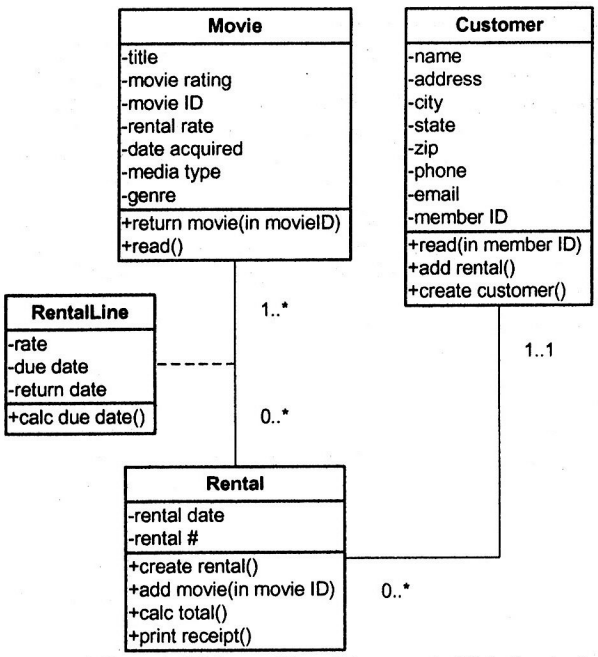


Figure 6. Jason's Video -- Class Diagram (with behavior)

While considerable work has been done to understand the transition from traditional methods (i.e., ERD diagrams) to OO thinking, previous researchers often overlook or give little attention to the question of whether the two approaches are mutually exclusive. This is an especially relevant question when attempts to utilize OO concepts in a data management context are considered.

There is some debate regarding whether a shift to OO methods requires evolutionary or revolutionary change in the cognitive processes of developers (Morris, Speier, and Hoffer 1999; Sircar et al. 2001). Those who argue that OO methods require revolutionary change (or paradigm shifts) suggest a need to abandon previous process knowledge due to a cognitive mismatch between the OO concepts and structured design concepts that interferes with the production of a high-quality product (Nelson, Armstrong, and Ghods 2002).

In one of few studies we found that decomposed the development lifecycle and considered the level of change taking place in different stages, the authors suggest that the level of change during the analysis and design phase is architectural (somewhere between incremental and radical) (Sircar et al. 2001). These authors conclude based on a co-citation analysis that a developer's knowledge about data and processes -- based on training or experience using a structured approach -- does not need to change in order to adopt an OO approach. However, the developer's knowledge about the relationship among these components must change because data and process are encapsulated.

Advocates of evolution suggest that knowledge of a structured development process can (and potentially should) transfer into the OO development environment (Morris et al. 1999; Shaft, Albert, and Jasperson 2004). Morris, et al, (1999) asked novice and experienced developers to develop two systems -- one using a structured approach the other an OO approach. They found developers experienced in using structured methods produced higher quality solutions than novice developers regardless of method employed suggesting that "...some prior systems analysis problem-solving knowledge transferred when moving from process to object modeling (p. 124)."

We argue for the latter approach. The advantages of this approach are fundamental: the reduction of uncertainty for the student and easier learning of OO concepts. Based on our experience in the classroom, our approach attempts to encourage students to transfer their existing knowledge about data modeling into their attempts to develop a class diagram. We believe this approach reduces uncertainty for the student making him/her more confident in his/her skills and more capable of evaluating his/her own solutions prior to submitting assignments.

We also believe our approach improves the efficacy of our coverage of modeling behaviors. The modeling of behavior has proven a difficult concept for students to learn. One recent study offered the following: "Placing the right operations in the appropriate objects...is a consistent problem for students (Sim and Wright 2001/2002: p 99)." Our approach facilitates the coverage of behaviors by not asking the students to engage in a joint modeling of state and behavior. While these elements are encapsulated within objects in an OO system, we find no evidence (either in the literature or our own experiences) that considering them serially results in lower quality solutions.

In contrast, our experience using CRC cards suggests that the difficulty in modeling behaviors is related to a difficulty in understanding the relationship between state and behavior that may be made more clear by considering them separately. In effect, we argue that by considering the state and behavior characteristics of a class separately we reduce the complexity of understanding the concept of a class. This in turn leads to an enhanced ability to move from the traditional, relational development environment to the more modern, OO development environment.

We are not the first to suggest that OO and traditional modeling processes might be combined. Others suggest a similar combined approach and recommend that OO schema be based on an ERD (Shoval 1988; Shoval and Kabeli 2001). Our method is not conceptually different from theirs. We do, however, provide a more detailed process for classroom

instruction using this approach. Further, we have attempted to articulate why this approach should be undertaken. In a related fashion, Halpin and Bloesch (1999) recommend using UML purely for analysis, thus the UML class diagram would provide an extended Entity-Relationship Diagram (ERD) notation.

It is our premise that the norm our students can expect in practice will very likely include both a traditional (i.e., relational) database environment and an object-oriented application environment. As such, the data modeling elements of many projects experienced in today's IT environment require those in practice to straddle both paradigms.

4.2 A Perspective from Practice

We asked an IT professional (our former student and third author) to comment on this premise. The following section contains his response. (We have made a few minor edits to the following for style and flow; however, for the most part the response is un-edited.)

> In the systems development projects I have completed the use of entity relationship diagrams and object model diagrams have both been critical to solving the business problem presented. In some cases, both models have driven the overall development effort. I find that several factors have led to the choice of which model was used. Those factors are as follows:
>
> 1. New system versus existing system enhancements
> 2. Tightness of coupling to other existing systems
> 3. Transactional nature of the system being designed
> 4. The level and amount of reporting needed for business users
>
> Factor one requires little explanation. In enhancement projects, we rely heavily upon the initial system design, and therefore, whatever drove this process to begin with would typically continue to do so for each enhancement effort.
>
> When a system is very tightly coupled to other systems, it becomes important to consider the designs of those systems in the design of the system we wish to build. For one project, the new system was tightly coupled to two existing systems, even sharing the same Oracle database. In this situation, although the application was a Java based application, and did have a detailed object model, it was the design of the database, and the entity relationship models that drove the overall design of the system.
>
> When a system is highly transaction based, the design of the database also tends to be a more critical factor in the overall success of the system. A system that processes large volumes of transactions will typically require the data model to be optimized for that purpose, whereas a system that is not transaction based doesn't have as strong a dependency.
>
> On another project, the system was not transaction based. It simply integrated information from two other transaction-based systems, which were built around a data model. Because the new system was designed with summation in mind and didn't share a data store with either of the two existing systems, the object model drove the design of the system.
>
> In a current project, I am developing a document classification rules management application. This system will be used to manage the classification rules, attributes, and information (metadata) for our enterprise document repository. The system integration is done by the use of Enterprise Java Beans. In this particular situation, the coupling of the systems is not very tight, and more importantly, neither system is at all transaction based. In fact, design patterns such as Session Facades, Business and Value Objects, and Business Delegates were implemented for the sole purpose of decoupling the logic of the two systems, and encapsulating the logic of the rules management system to make change management easier. This was clearly a case where the object model drove the design of the overall system. We began with hashing out an object model to facilitate our system and then built a database structure which could support that model.
>
> The last factor I have found to play a key role in the decision of which model drives the design is the need for reporting. In another project I was assigned, called Well Process Management (WPM), the system was designed to be the integration point between all other systems in the scope of the Well Life Cycle. Although it was to be coupled with several transaction based systems, the coupling was to be very loose, and the business logic of the WPM system was to be completely encapsulated from the other systems it touched. This was a situation where the object model played a much greater role in system design. However, the purpose of the system was to provide management with a very high level view of the well life cycle across the entire company. That need for real time reporting made the design of the database an equally important consideration. In this particular situation, one could argue that both models played an equal role in leading the design process.

Assuming these experiences are not uncommon, current IT practitioners often shift from ERDs to object models and back again. Further, they need to understand the mapping from one to the other. Finally, as the new systems often must be integrated with existing systems, this need to straddle relational and OO thinking is unlikely to go away any time soon.

If in practice shifting from one modeling technique to the other is the norm, perhaps the focus of our teaching should be less about which model to teach (i.e., ERD or class diagram), but instead on how to conceptually integrate them in such a way as to facilitate easy transition from one to the other and arm our students with the ability to use either depending on the best fit with the project characteristics.

We believe these examples from prior research and current practice support our recommended approach. Furthermore, we believe the student gains ability to leverage the robust traditional processes while simultaneously tapping into the rich and standardized notation available in the UML class diagram.

5. CONCLUSION

The purpose of this special issue was to address a growing question in our community about whether we should be teaching ERDs versus modeling using the UML notation. Ours is a view from the fence. We believe both a relational data modeling view as well as a class diagram view of data is necessary. The method chosen depends on where the core database course appears in the MIS curriculum. If it precedes systems analysis, then we would encourage instructors to cover data modeling using traditional ERD notation – keeping data and process logic separate. After which, the instructor in the systems analysis course can introduce process logic. Now the combining of process and data can be more fully understood, and UML notation can be useful. If database comes later in the curriculum, after systems analysis, then UML notation should be used in the database course. This presupposes that data and process have been decoupled in the systems analysis course and students are now ready to absorb the complexities of the more integrated OO approach.

The use of OO for new application development is fairly well-established as fact—especially the use of OO programming languages. Paradoxically, relational databases are characterized in the trade press as ubiquitous (Walsh, 2004). As such, IT professionals charged with developing processes that capture, process, or report data will continue to need relational database skills as well as developing an understanding of classes and objects.

Based on this premise, we articulate a process to facilitate teaching data modeling and object thinking in an integrated (albeit serial) fashion. While we have used our approach for a number of semesters, we recognize that the conclusions we draw are non-scientific and encourage rigorous research to determine the efficacy of our advocated approach.

6. REFERENCES

Beck, K. and Cunningham, W. "A Laboratory for Teaching Object-Oriented Thinking," *SIGPLAN Notices* (24:10), October 1989, pp 1-6.

Bellin, D. and Simone, S. S. *The CRC Card Book*, Addison-Wesley, Boston, MA, 1997.

Chen, P. "The Entity Relationship Model - Towards a Unified View of Data," *ACM Transactions on Database Systems* (1:1), March 1976, pp 9-36.

Chen, P. *The Entity-Relationship Approach to Logical Database Design*, Q.E.D. Information Sciences, Wellesley, MA, 1977.

Date, C. J. *An Introduction to Database Systems*, (4th ed.) Addison-Wesley, Reading, MA, 1986.

Fowler, M. and Scott, K. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, Reading, MA, 2000.

Halpin, T. and Bloesch, A. "Data Modeling in UML and ORM: A Comparison," *Journal of Database Management* (14:4), Oct-Dec 1999, pp 4-14.

Hoffer, J. A., Prescott, M. B., and McFadden, F. R. *Modern Database Management*, (7th ed.) Addison-Wesley, Reading, MA, 2005.

Johnson, R. and Hardgrave, B. C. "Object-Oriented Systems Development: Current Practices and Attitudes in Industry," *Journal of Systems & Software* (48:1) 1999, pp 5-12.

Krill, P. "Two Database Projects Afoot in Eclipse," *InfoWorld* (27:17), April 25, 2005, p 23.

Lai, E. "Starwood Checks in with Object Database for Reservations," *Computerworld* (40:3), January 16, 2006, p. 19.

Martin, J. *Computer Database Organization*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

Monash, C. A. "Looking Beyond the Big Three," *Computerworld* (39:19), May 9, 2005, p 26.

Morris, M. G., Speier, C., and Hoffer, J. A. "An Examination of Procedural and Object-Oriented Systems Analysis Methods: Does Prior Experience Help or Hinder Performance," *Decision Sciences* (30:1), Winter 1999, pp 108-136.

Nelson, H. J., Armstrong, D. J., and Ghods, M. "Old Dogs and New Tricks," *Communications of the ACM* (45:10), October 2002, pp 132-137.

Pancake, C. M. "The Promise and the Cost of Object Technology: A Five-Year Forecast," *Communications of the ACM* (38:10), October 1995, pp 33-49.

Rosson, M. B. and Alpert, S. R. "The Cognitive Consequences of Object-Oriented Design," *Human-Computer Interaction* (5:4) 1990, pp 345-379.

Shaft, T. M., Albert, L. J., and Jasperson, J. "A Longitudinal Study of Information Systems Developers' Understanding of Software Development Concepts During a Transition from Structured to Object-Oriented Development," presented at *Americas Conference on Information Systems*, C. Bullen and E. Stohr (eds.), New York, NY, 2004, pp. 1620-1630.

Shaft, T. M. and Jasperson, J. "Transition to an Object-Oriented Development Environment: A Summary of Learning from CITGO," presented at *University of Oklahoma Center for MIS Studies Spring Meeting*, Norman, OK, April 2003.

Sheetz, S. D., Irwin, G., Tegarden, D. P., Nelson, H. J., and Monarchi, D. E. "Exploring the Difficulties of Learning Object-Oriented Technology," *Journal of Management Information Systems* (14:2), Fall 1997, pp 103-131.

Shoval, P. "ADISSA: Architectural Design of Information Systems Based on Structured Analysis," *Information Systems* (13:2) 1988, pp 193-210.

Shoval, P. and Kabeli, J. "FOOM: Functional- and Object-Oriented Analysis & Design of Information Systems: An Integrated Methodology," *Journal of Database Management* (12:1), Jan-Mar 2001, pp 15-25.

Shoval, P. and Shiran, S. "Entity-Relationships and Object-Oriented Data Modeling -- An Experimental Comparison of Design Quality," *Data & Knowledge Engineering* (21) 1997, pp 297-315.

Sim, E. R. and Wright, G. "The Difficulties of Learning Object-Oriented Analysis and Design: An Exploratory Study," *Journal of Computer Information Systems* (42:2), Winter 2001/2002, pp 95-100.

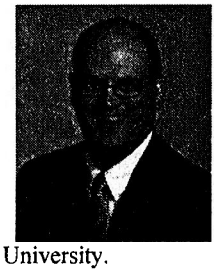
Sircar, S., Nerur, S. P., and Mahapatra, R. "Revolution or Evolution? A Comparison of Object-Oriented and Structured Systems Development Methods," *MIS Quarterly* (25:4), December 2001, pp 457-471.

Vician, C., Garfield, M., Hoffer, J. A., Prescott, M. B., Rollier, B., Strong, D. M., and Elder, K. L. "The AMCIS 2003 Panels of IS Education-II: The Chicken and the Egg Debate: Positioning Database Content in the Information Systems Curriculum," *Communications of the AIS* (14:7) 2004, pp 147-231.

Walsh, G. "The Art of Object-Relational Mapping," *Software Development Times* (113) 2005, p 25.

AUTHORS BIOGRAPHIES

Traci A. Carte is an Associate Professor of MIS in the Michael F. Price College of Business at the University of Oklahoma. She received her Ph.D. in MIS from the University of Georgia. Currently, her research interests include IT support for diverse teams, politics and IT, and research methods. Her research has been published in such journals as *MIS Quarterly*, *Information Systems Research*, *Decision Support Systems*, and *Journal of the AIS*. She serves on the editorial board of *MIS Quarterly*.



'Jon (Sean) Jasperson is a Clinical Assistant Professor for the Information and Operations Management Department in the Mays Business School at Texas A&M University. His research interests include the adoption, use, management, and implementation of information technology in organizational settings. He received his Ph.D. from Florida State University.

Mark E. Cornelius is a senior ITS Developer in IT Services for Anadarko Petroleum Corporation. He received his MBA from the Michael F. Price College of Business at The University of Oklahoma. His professional interests currently focus on J2EE development, systems integration, and messaging technologies.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096