A SPECIALIZATION OF OBJECT-ORIENTED DESIGN FOR THE FOURTH GENERATION LANGUAGE, BUSINESS ENVIRONMENT

ROGER LEE
Central Michigan University
Mount Pleasant, Michigan 48859

WILLIAM LEIGH
University of Central Florida
Orlando, Florida 32816

INTRODUCTION

This paper describes the use of selected object-oriented software design tools (including the object diagram, the state-transition diagram, and the use case) in the fourth-generation language environment of business information systems. A design tool, the Event/Transition/Object Table, which integrates state methods with the object diagram, is proposed. The employed use case format complements the Event/Transition/Object Table. The described method is presented as being adequately prescriptive in nature for teaching to undergraduate systems design students.

The structured design methods comprise the topic of systems design in many computer information systems textbooks (for example, 16). Structured design is most applicable for the design of systems to be implemented in procedural, third-generation programming languages, such as COBOL. In software engineering, the structured design methods have evolved into the object-oriented methods (2, 14) for the design of software systems, especially for implementation in object-oriented programming languages, such as C++. Without modification, neither the structured design methods nor the object-oriented methods are applicable to use with the fourth-generation, object-based, high productivity programming tools which are used with increasing frequency in business today (7). The object-oriented approaches are being integrated into the computer information systems textbooks, but more work is needed in the development of design methods which are directly applicable to the fourth-generation language (4GL) tools used in building business applications.

The application of object-oriented methods to information systems has been described in other places (for example, 6, 11, 13). This previous work was premised on bringing the whole mechanism of object-oriented programming to bear on the information systems problem. This paper concentrates on identifying only the aspects of the object-oriented methods that facilitate the implementation of information systems using 4GL, object-based tools, rather than full-blown object-oriented programming languages.

"The current state of the technology does not define object-oriented design as one definitive combination of models, languages, or methods" (9). In practice, object-oriented design may involve any of the tools described in the standard texts (2, 10, 14). These methods have been described as first-generation object-oriented design methods, and authors have tailored an object-oriented approach for a particular problem by selecting a semi-orthogonal set of tools and modifying them as appropriate (for example, 4). A similar course is followed in this paper to develop a set of design tools drawn mainly from the object-oriented family which supports business systems building with 4GL tools.

THE METHOD

The object-oriented design approach (8) begins by identifying the objects of the information system and then captures four types of relationships between the objects (which become dimensions of the software architecture): 1) logical static, 2) logical dynamic, 3) physical static, 4) physical dynamic. The objects in business information systems are the entities and events of business. Entities include vendors, invoices, and so forth. Events are the corresponding acts of doing business: approve vendor, issue invoice, and so forth.

In the methodology described herein the four dimensions of relationships between these objects are attended to:

1. Modeling of logical static relationships between objects: information system objects are data entities and processing events. An object structure diagram (also called a class diagram) performs this function in most object-oriented design methodologies. This diagram is a generalization of the Entity-Relationship Diagram (ERD) (originally proposed in 3, and related to business applications in 1) from an exclusive focus on data entities to all entities and events of the system, and the addition of a class inheritance hierarchy. For 4GL-implemented business systems, the ERD subset (data objects and their relationships) is considered to be the proper level of object structure to be concerned with (5, 16), as the discovery or programming of classes (as the 4GL is object-*based* -- implying that new objects and classes cannot be added -- rather than object-*oriented*) is not a possibility, and the relationships between the events in business information systems are highly structured. This paper will use the object-oriented terminology for the ERD subset of the object diagram. (See 16 for an explanation of the correspondence between the object-oriented and the ERD terminology.)

2. Modeling of logical dynamic interaction within objects: The state transition diagram (STD) is the standard object-oriented tool. The described methodology is based on a realization that business transaction processing events correspond to transitions between states of an object type.

3. Modeling of physical static relationships between

objects: This refers to the layout of code in modules, the use of standard functional components (such as the "Form" object in Microsoft ACCESS) makes this design dimension moot and given.

4. Modeling of the physical dynamic interaction between objects: This refers to the process and thread architecture for the software. This is completely determined by the 4GL environment and normally may not be modified by the applications designer at all, so this design dimension is omitted from this method.

The Event/Transition/Object Table (E/T/O) shown in the following example integrates the logical static, logical dynamic, and physical static dimensions by recognizing that processing events correspond to transitions between states of an object type and that each processing event must be implemented by an instantiation of an object of class "Form." Processing events may be implemented as interactive forms, requiring entry of the detailed data of the transaction, or they may be implemented as "batch" forms, the invocation of which executes a series of SQL (Structured Query Language -- the standardized relational database language for data definition, manipulation, and control). This E/T/O is a tabular representation which is quickly understood, is easy to synthesize from the object diagram, the STD, and an understanding of the system to be built. The preparation of the E/T/O causes the designer to analyze the object diagram and the STD so as to detect errors or omissions. The E/T/O is easy to inspect (that is, to "grade") in the instructional setting.

The use case is a tool from the object-oriented family which helps ground the design in the specifics of reality. The designer selects or contrives a sequence of example transactions from the application context to illustrate and test the design. A portfolio of use case examples is collected as the design process proceeds.

The format for presentation of the use case in the proposed method parallels the structure of the E/T/O diagram and can be used to directly validate the E/T/O. The use case format requires that the attribute make-up of the data objects be determined, which is the next step in system design following the E/T/O, so the use case serves as a design tool, as well as an illustration and test tool.

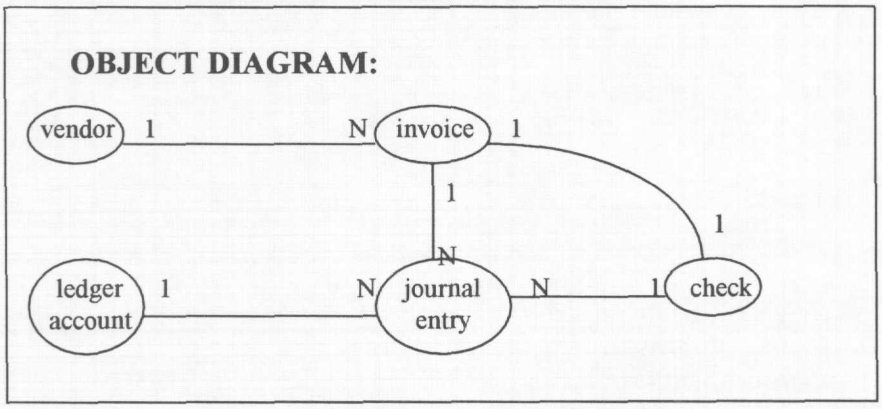
Implementation in a 4GL follows directly from the preparation of the object diagram, the E/T/O, and the use case. The multiplicities ("cardinalities" in ERD terminology) and relationships shown on the object diagram and the attribute information included in the use case translate directly into SQL Data Definition Language to establish the database (see 17 for specifics of this process). The interactive forms needed are generated by the 4GL. Additional business rules are implemented as procedural code attached to form actions. Other forms are built to directly execute the SQL Data Manipulation Language for reports and batch updates.

Reports for a system can be added as required. The reports are derived from the data objects and are easily implemented using the SQL and report generation tools of a 4GL. Reporting is well understood and may be addressed as a detail design activity after the general design is accomplished.

AN EXAMPLE

The object diagram in Figure 1 shows the data entities and the relationships between them in the example accounts payable information system. For example, a "ledger account" object occurrence is "one-to-many" with a "journal entry" object occurrence, that is, each ledger account may have up to many journal entries.

FIGURE 1
Object Diagram for Example Accounts Payable System



The STD in Figure 2 shows the dynamic behavior of the data entities in the object diagram. The "journal entry," "ledger account," and "check" entities each have only one state, resulting from their respective "create" transitions. The "invoice" progresses through "open," "approve," and "paid" states, resulting from the "create," "approve," and "pay" transition.

The relationships between external events, the state transitions, and the entry, update, or deletion of the data for the data objects is represented in the E/T/O Table. Each event in the E/T/O is implemented as a "form" in the AIS. The event "Friday Comes" is implemented as a batch processing execution form (a single button which invokes a processing stream of SQL, in this case), as the invoices are paid on Friday: the appropriate entries in the check table are made ("I" for insert), the invoices are marked paid ("U" for update), and the journal entry table insertions are made. Other events might require deletion entries ("D"). Each event on the E/T/O is implemented as a "logical transaction" for the database management system.

FIGURE 2
State-Transition Diagram for Example Accounts Payable System

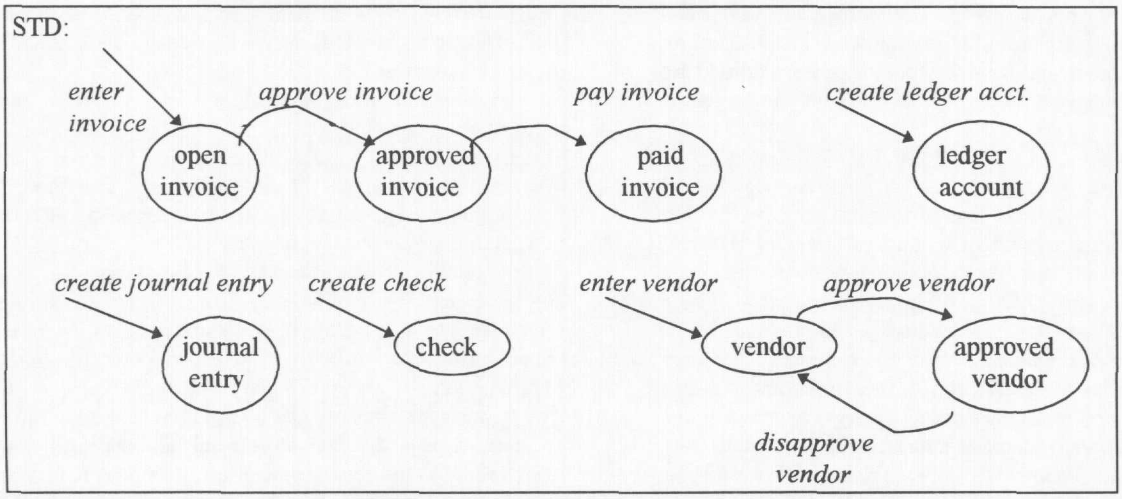
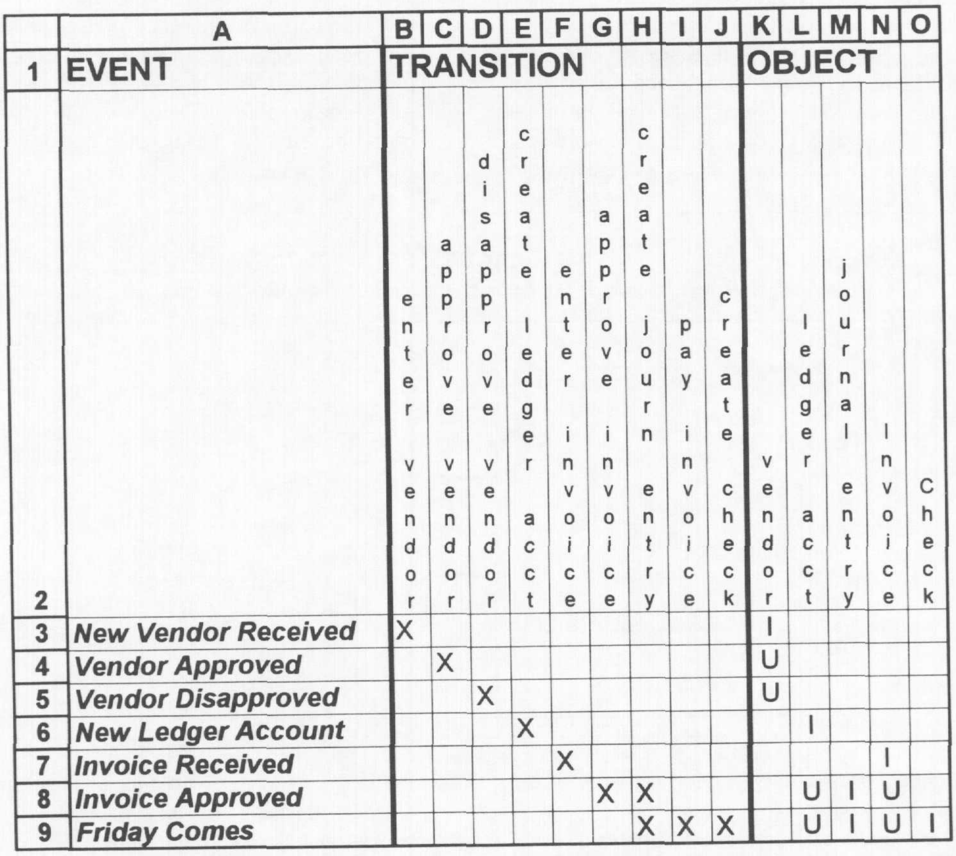


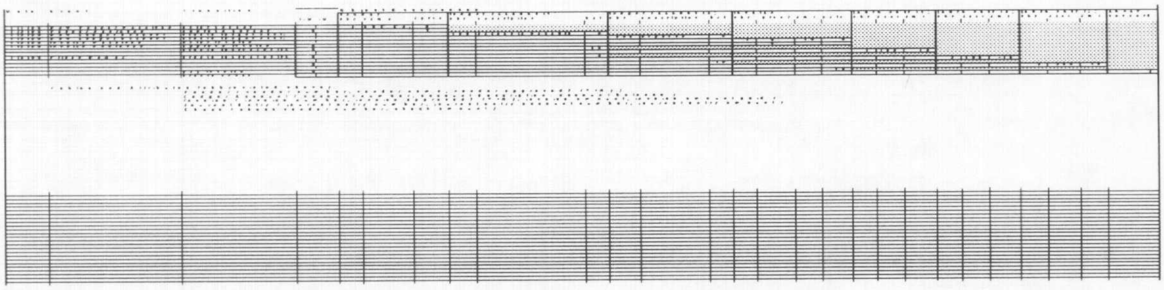
FIGURE 3
Event/Transition/Entity Table for the Example Accounts Payable System



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	EVENT	TRANSITION									OBJECT				
2		enter vendor	approve vendor	disapprove vendor	create ledger acct	enter invoice	approve invoice	create journal entry	pay invoice	create check	vendor	ledger acct	journal entry	Invoice	Check
3	New Vendor Received	X									I				
4	Vendor Approved		X								U				
5	Vendor Disapproved			X							U				
6	New Ledger Account				X							I			
7	Invoice Received					X								I	
8	Invoice Approved						X	X				U	I	U	
9	Friday Comes							X	X	X		U	I	U	I

The use case shown in Figure 4 displays the attribute make-up of some of the data objects and illustrates the effects of example transaction processing on some of the object occurrences. The format parallels the structure of the E/T/O Table, except that object occurrences, rather than object types, are included. Each object occurrence has separate columns for its attributes so that the history of changes to each data value may be seen in detail.

FIGURE 4
Use Case Example
(See Attached for Complete Version)



DESIGN AUDIT

The designer (or instructor) may audit the design and the implementation for internal validity and consistency through application of identities between representations:

1) There must be an object column in the E/T/O for each object on the object diagram.

2) There must be a transition column in the E/T/O for each transition in the STD.

3) There must be at least one "X" in the E/T/O under each transition.

4) There must be at least one "I" in the E/T/O under each object.

5) There must be at least one "X" and at least one "I/U/D" in each row of the E/T/O.

6) The implementation must have one table for each object on the object diagram.

7) The implementation must have one form for each row of the E/T/O (each event).

8) An "I" (insert) entry should be first in each object occurrence history column in the use case and a "D" (delete) should be last, with no Is or Ds in between, only "U"s (update).

This concept of design audit enhances the prescriptive and structured nature of the proposed method. Undergraduates tend to prefer methods based on explicit sets of rules.

CONCLUSION

The method is concise and minimalist, applicable to determining and representing requirements in evolutionary development, which is the software process model of choice for small and medium-sized business system development using 4GLs (15). The method is complete, in representing the four dimensions as required, but is structured and prescriptive, appropriate for introduction and use in an undergraduate information systems analysis and design course. The method is specialized to the specific needs of business information systems development in a fourth generation language context.

REFERENCES

1. Amer, T.S. "Entity-Relationship and Relational Database Modeling Representations for the Audit Review of Accounting Applications: An Experimental Examination of Effectiveness," **Journal of Information Systems,** 7, 1993, pp. 1-15.
2. Booch, G. **Object-Oriented Design With Applications,** Benjamin/Cummings, 1991.
3. Chen, P.O. "The Entity-Relationship Model -- Toward a Unified View of Data," **ACM Transactions on Database Systems,** 1, 1976, pp. 546-560.
4. Coleman, D. **Object-oriented Development -- The Fusion Method,** Englewood Cliffs, NJ: Prentice-Hall, 1994.
5. Davis, J.S. "Teaching a Client/Server Course to Business Graduate Students," **Journal of Computer Information Systems,** Fall 1996, pp. 42-47.
6. Deng, P. and C.L. Fuhr. "Using an Object-Oriented Approach to the Development of a Relational Database Application System," **Information & Management,** 1995, pp. 107-121.
7. Douglas, D.E. and P.D. Massey. "Is Industry Embracing Object-Oriented Technologies?" **Journal of Computer Information Systems,** Spring 1996, pp. 65-72.
8. Holland, I.M. and K.J. Lieberherr. "Object-Oriented Design," **ACM Computing Surveys,** 1996, pp. 273-275.
9. Hollander, A.S., E.L. Denna, and J.O. Cherrington. **Accounting, Information Technology, and Business Solutions,** Richard D. Irwin, 1996.
10. Jacobson, I., M. Christerson, P. Jonsson, and G. Overgaard. **Object-Oriented Software Engineering: A Use Case Driven Approach,** Reading, MA: Addison-Wesley, 1992.
11. Kandelin, N.A. and T.W. Lin. "A Computational Model of an Events-Based Object-Oriented Accounting Information System for Inventory Management," **Journal of Information Systems,** 1992, pp. 47-62.
12. McKie, S. "Accounting Objects: Financial Applications Go Object Oriented," **DBMS,** August 1994, pp. 69-74.
13. Murthy, U.S. and C.E. Wiggins. "Object-Oriented Modeling Approaches for Designing Accounting Information Systems," **Journal of Information Systems,** 1993, pp. 97-111.
14. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. **Object-Oriented Modeling and Design,** Englewood Cliffs, NJ: Prentice-Hall, 1991.

15. Sommerville, I. "Software Process Models," **ACM Computing Surveys,** 1996, pp. 269-271.

16. Teory, T.J. **Database Modeling and Design, The Fundamental Principles,** San Francisco: Morgan Kaufmann, 1994.

17. Yourdon, E. **Modern Structured Analysis,** Englewood Cliffs, NJ: Yourdon Press, 1989.

FIGURE 4
Use Case Example

Date	Event	Transition	I/D/U	Object Occurrences vendor			ledger account (1)			ledger account (2)			invoice				journ entr (1)			journ entr (2)			journ entr (3)			check	
				nbr	name	aprvd	nbr	description	bal	nbr	description	bal	nbr	aprvd	amt	paid	nbr	acct	amt	nbr	acct	amt	nbr	acct	amt	vendor#	amt
1/1/96	*New Vendor Received*	enter vendor	I	3	ABC Co	N																					
1/2/96	*Vendor Approved*	approve vendor	U			Y																					
1/2/96	*New Ledger Account*	create ledger acct.	I				100	accounts payable	0																		
1/2/96	*New Ledger Account*	create ledger acct.	I							200	expense	0															
1/2/96	*Invoice Received*	enter invoice	I										1	N	10	0											
1/3/96	*Invoice Approved*	approve invoice	U											Y													
		create journal entry	U						10																		
			I														1	100	10								
1/5/96	*Friday Comes*	create journal entry	U						0																		
			I																	2	100	-10					
			U									10															
			I																				3	200	10		
		pay invoice	U													10											
		create check	I																							3	10

Tables and Attributes (abbreviated above):

Vendor (vendor number, vendor name, vendor approved? (Y/N))
Ledger Account (account number, account description, account balance)
Invoice (invoice number, invoice approved? (Y/N), invoice amount, invoice amount paid)
Journal Entry (journal entry number, journal entry account number, journal entry amount)
Check (check vendor number, check amount)