ADVANCING THE SYSTEMS ANALYSIS AND DESIGN CURRICULUM

Stevan Mrdalj, Eastern Michigan University, stevan.mrdalj@emich.edu
Vladan Jovanovic, Georgia Southern University, vladan@gasou.edu

ABSTRACT

Computer Information System and related programs are expected to produce students who possess a broad and contemporary understanding of analysis and design for information systems. An empirical analysis of the state of practice in systems analysis and design education revealed an emphasis on structured design in the majority of schools. The opportunity to transition to object-oriented analysis and design, in the CIS curriculum, is based on the use of the Unified Modeling Language (UML) standard. A roadmap for such advancement is presented by a comparison of concepts and tasks in modeling web applications using various diagrams offered by conventional structured methods and by UML.

Keywords: systems analysis and design, curriculum, object-oriented, UML, structured design.

INTRODUCTION

UML has clearly become the specification language of choice for systems analysis and design in the industry (1). According to a review of eighteen empirical studies (8), the majority found the object-oriented design (OOD) approach superior to the classical structured design (CSD). The supporters of OOD claim many advantages, including greater user satisfaction, improved modularity, maintainability and adaptability. Nearly all studies, where negative results were obtained, come from the use of inexperienced students as subjects. Many claim to be teaching OOD. According to a study by Hardgrave and Douglas (6), 51% of the schools are teaching at least one OOD method (n=106). In reality, very few programs seem to use UML and teach OOD. In our study we examined the web based syllabi for systems analysis and design courses at 125 programs. For this purpose we started with the AASCB directory of accredited CIS programs. We discovered that only 16% are teaching using UML and only 4.8% teach both CSD and OOD. By further examination of the books used in those courses that teach OOD, we discovered that some (4) have replaced data flow diagrams with use cases and entity-relationship diagrams with class diagrams ignoring or lightly covering many other very important representation diagrams or they use professional books (10). Other most frequently used books (7,9,14) have one chapter describing all UML diagrams. There is only one book (12) that we are aware of that successfully covers in parallel both SD and OOD with UML. At the same time, the study by Hardgrave and Douglas (6) reveals that 84% of the programs teach at least one OO language. Such commitment to OO programming (5) also reveals that the larger picture of system design is being neglected, indicative of the educational content that is no longer appropriate.

In order to advance the systems analysis and design curriculum content, we need to examine CSD and UML and to analyze the conceptual content that provides the basis for producing better systems. As most of the problems with the CSD are evident in developing web applications, we decided to perform a comparison in the framework of web applications. This decision was also influenced by Wells who pointed out (13) that web applications and eBusiness will fundamentally alter the methods, techniques, and tools within the system development life cycle. Thus, an overview of diagrams used in both CSD and OOD is presented in respect to system complexity, structure, behavior, user interface navigation and architecture.

COMPLEXITY CONTROL

A standard technique of mastering a complex system is to decompose the system into smaller and smaller parts until each part becomes small enough to be understood. When the system is decomposed into modules, each carrying out a major step in the overall process, we are using the top-down structured design. When the system is decomposed into usage profiles, each carrying out a major interaction between actors and the system, we are using use case modeling.

The most frequently used diagramming technique for top-down system decomposition is the data flow diagram (DFD). System models represented by DFDs capture the flow of control among processes and demonstrate how data get modified as they flow throughout the system. Another decomposition method is to decompose the system into use cases, Figure 1. Booch (1) describes a use case as a description of a set of sequences of actions, including variants, which a system performs to yield an observable result to an end-user. A use case diagram (UCD) captures the intended behavior of the system without having to specify how that behavior is implemented.

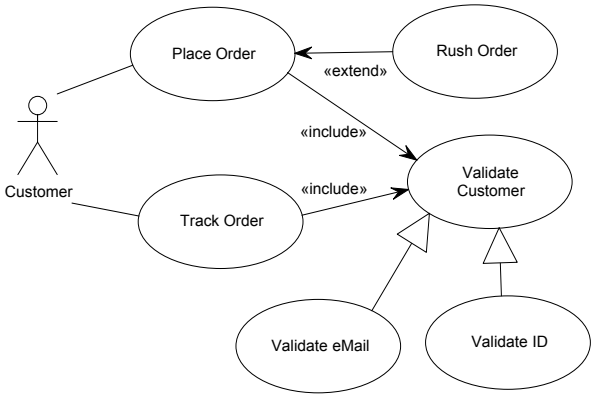


Figure 1: Use Case Diagram

Perhaps the most serious weakness of DFD is that it uses a single function at the top, a dubious requirement for many modern event-driven systems. Defining web applications in terms of a single top-level function is artificial and yields overly complex and non-adaptive architectures. Top-down function-based design also has scalability limitations that become apparent when developing and maintaining large systems. The customer-centered, event driven and interaction based approaches in deriving requirements for web applications make DFDs difficult to use in the analysis phase (13).

Contrary to the hierarchical structure of functions, use cases represent independent interactions that can be initiated at any time, in any order, and they are well suited for all interactive GUI and web based applications. This approach yields system decomposition into a set of use cases with the ability to incrementally grow from a reliable small system into a more complex one (1).

Use case diagrams offer behavior sharing and variants by specifying <<include>> and <<extend>> as shown in Figure 1. DFD's support for reuse, while possible, is not formalized. The third method of dealing with complexity is through abstraction specified by generalization relationships in Figure 1. Abstraction is easier and is a more natural part of system decomposition than functional decomposition methods.

REPRESENTING SYSTEM STRUCTURE

The system structure is described using entity-relationship diagrams (ERD) within CSD and using class diagrams (CD) within UML. Since ERD is a subset of CD, whatever is possible to represent using ERD, is also possible to represent with CD. In respect to the physical data base design, it can be performed the same way using CD as it would be done by using ERD. Table 1

summarizes some features of CD that are not available in ERD, like realization, dependency and qualification which may significantly help in the process of physical system design.

Table 1: CD extensions to ERD

Operation	Class Name / Operation List
Composition	◆
Aggregation	◇
Realization	▷
Dependency	>
Qualification	Class Name [Qualifier]

The essential difference is that a class consists of the data and all the necessary operations that manipulate it. Classes are founded on the basic concepts of encapsulation, messaging, inheritance and polymorphism. Encapsulation organizes data and the corresponding processes which manipulate that data into a single object. The data and operations contained in an object are conceptually related to each other and distinct from all other objects in a system. Inheritance allows objects to share attributes and behaviors without separately duplicating the program code that implements them. Polymorphism, perhaps the most powerful feature together with inheritance, allows the shared code that objects acquire through inheritance to be tailored to fit the specific requirements of an object. This feature of an object allows for a higher level of abstraction in the design of software and the construction of the frameworks.

Representing Document Structure

The great distinction of web applications is the usage of enabling technologies. Client-side enabling technologies are object-oriented and can be quite sophisticated. Regardless of the underlying philosophy for enabling the client, the technology relies on the Document Object Model (DOM) framework (2). The DOM is a platform-neutral interface to the browser and the HTML document. DOM contains user interface classes like window, document, buttons, etc. The trick to designing a web application is in the understanding of the objects and the interfaces you have to work with. For example, Figure 3 shows the design for the web page in Figure 2. Since quite often web applications require the development of business objects at the application server side, it is necessary to use CD to represent the application server structure. Therefore, the ERD is an inadequate tool to be used to design both the client and server application structure.

REPRESENTING BEHAVIOR

The main tool for representing behavior in CSD beside DFD is a structure chart (SC). In developing SCs we employ a top-down decomposition or stepwise refinement by moving from a general statement about the process involved in solving a problem down towards more and more detailed statements about each specific task in the process. Since the decomposition only highlights the functional aspects of the problem, the influence of the data structures on the problem is lost or is very minimal.

For the purpose of comparison with SC we will use UML's sequence diagrams (SD) shown in Figure 4 and collaboration diagrams (CoD) shown in Figure 5. Both diagrams show an interaction consisting of a set of objects and messages that may be dispatched among them. Booch (1) describes a SD as a diagram that emphasizes the time ordering of messages and a CoD as a diagram that emphasizes the structural organization of the objects.

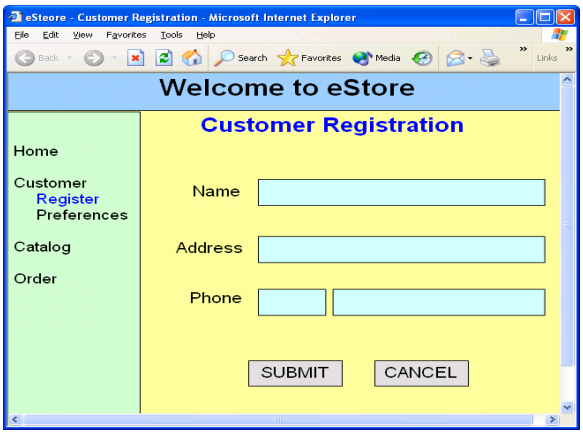


Figure 2: Customer Registration Web Page

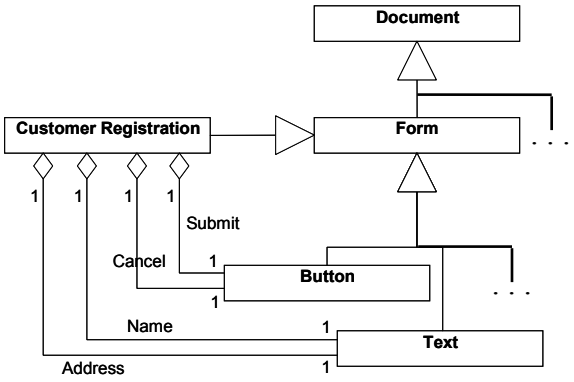
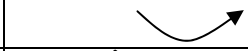
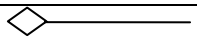


Figure 3: Simple web page design

Table 2 describes differences between SC, SD and CoD. While all three diagrams have similar control structures, SC can only represent hierarchical structure whereas both SD and CoD can represent a network structure. The SC may create a good software model for the initial requirements of a system. But as that system changes and new requirements are added to a relatively fixed tree structure, changes usually require extensive pruning and graphing. Both SD and CoD have much richer semantics than SC. For example, CoD has links that specify a path along which one object can dispatch a message to another. Such paths can be stereotyped as association, self, global and parameter. SD and CoD have message arrow-lines that can represent call, return, signal, creation and destruction. CoD may represent synchronization and treads.

Table 2: Comparison of SC, SD and CoD

Feature	Structure Chart	Sequence Diagram	Collaboration Diagram
Process	Function	Message	Message
Data	Data couple	Object	Object
Connection	Line		Link
Flow of Control	Top-down, Left-to-right, Control flag	Top-down, Arrow-line	Link, Arrow-line, Sequence numbers
Iteration		*	* or *[condition]
Selection		[condition]	[condition]

With the emergence of the web applications, there are web interfaces issues that pertain to all web sites and require special consideration during application design (13). The principal strategy for deploying web applications is to use multi-tier architecture like the one shown in Figure 8. Each tier contains objects that communicate with each other. For example, the client tier contains browser related objects, the web server tier contains controller related objects like sessions, and the application server contains business objects. The UML interaction diagram is an especially useful tool to represent interaction among those objects. SC and ERD were amenable with the traditional two tiered client/server architecture, separating the process and data, but they are insufficient to model modern multi-tiered architectures.

Document Navigation Flow

Most web applications get their work done by navigating from one screen to the next. It is crucial to design and document expected navigational paths since they represent business logic. Because of the lack of messaging and objects, SC cannot be used to represent the navigation flow

between screens. Since CSD does not offer a tool for representing navigation flow, some authors use hierarchical dialog diagrams (HDD) for this purpose (7). Although such representation shows all the screens and their hierarchy, it does not adequately represent the dynamic flow between the screens and related events. Other authors (3,4) use Windows Navigation Diagrams (WND) whose main addition to HDD is a network structure. We advocate using SD for representing object interaction among tiers like it is shown in Figure 4.

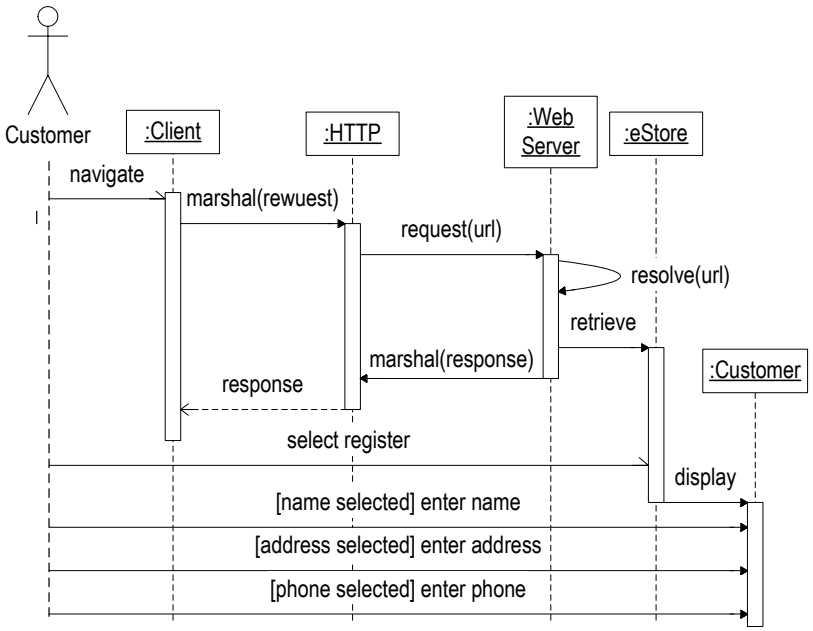


Figure 4: Sequence diagram

Representing Business Logic

As we mentioned before, an application server may contain business objects. The business logic is accomplished by messaging between objects. Messages and objects are independent. That is, the same message may be sent to more than one object, each performing its specific tasks or an object may respond to more than one type of message. A message to one object may require that object to send a message to a third object, and it to a fourth object, and so on. Changing the business logic of a system can therefore be accomplished by changing objects or by changing the messaging structure. Business logic through the messaging among objects is best represented with CoD like the one shown in Figure 5 in conjunction with the corresponding CD.

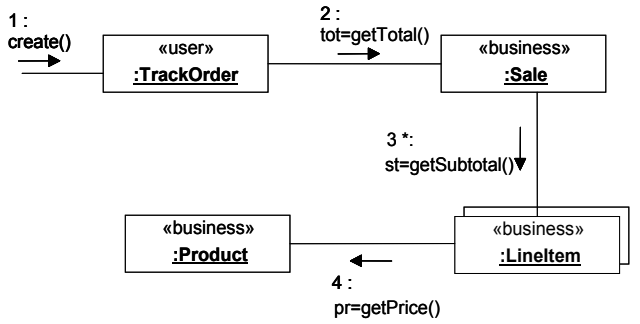


Figure 5: Collaboration Diagram

Representing Business Rules

The statechart diagrams are one of the UML diagrams for modeling the dynamic aspects of systems. They can be attached to a class, a use case or even an entire system. For the most part, they are used for modeling the behavior of reactive objects. In this paper, we will use the statecharts shown in Figure 6 to formally represent the business rules in user interaction with the web page, shown in Figure 3, where each independent event like data entry or button click need to be appropriately modeled as a transitions. This can be modeled using composite states. Such a level of abstraction is not available in WND or in any other diagram within SCD.

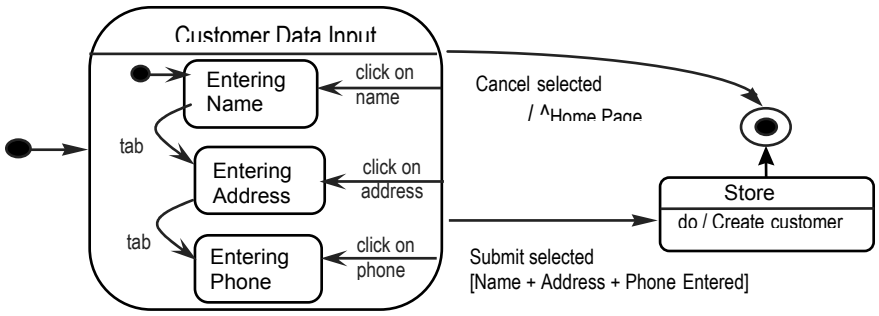


Figure 6: Statechart Diagram

REPRESENTING PHYSICAL ARCHITECTURE

When you architect a web application, you have to consider both its logical and physical dimensions. On the logical side we design classes, interfaces, interactions and states. On the physical side we design components and their deployment on the set of hardware.

Representing Components and Deployment

In UML all physical things are modeled as components. A component is a physical thing that confirms to and realizes a set of interfaces. A graphical representation of a component is shown in Figure 7.

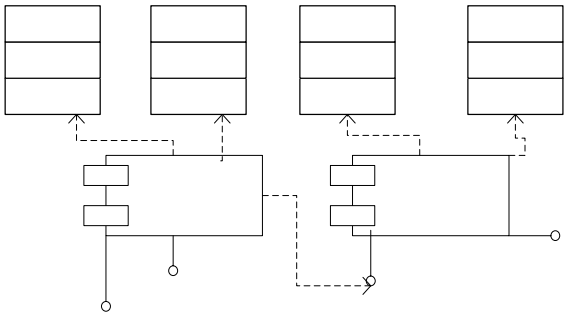


Figure 7: Component Diagram

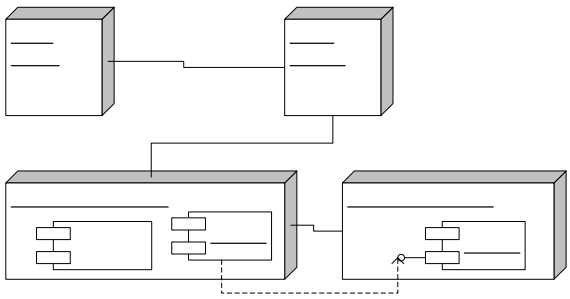


Figure 8: Deployment Diagram

The components we develop or reuse, as part of a web application, must be deployed on some set of hardware. The UML provides a graphical representation of the node, as Figure 8 shows. A node should be visualized apart from any specific hardware. Using stereotypes would accommodate the representation of the specific kind of hardware such as a processor, device etc.. The most common type of a relationship among nodes is an association. Associations may be stereotyped to indicate a physical type of connection like <<10-T Ethernet>> or <<RS-232>>. Nodes may be grouped in packages.

CONCLUSION

By raising the level of abstraction from the function-level to the object-level and by focusing on the real-world aspects of the system, OOD tends to promote cleaner designs that are easier to implement and provides for better overall communication. The UML diagrams in OOD work together, as views of the same system, synergistically to produce design solutions that better

Order lineItem Product Catalog

model problem-domain aspects than similar systems produced by CSD. The systems designed using UML are easier to adapt to changing requirements, easier to maintain, more robust and promote greater design and code re-use. Using an object-oriented language (e.g. C++, VB, Java or C#) adds support for OOD and makes it easier to produce more modular and reusable code. Of course, you can develop a web application using DFD, ERD and SC. By not being able to design the document structure, document navigation flow, document business rules, system components and system deployment using SD you trade short-term convenience for long-term inflexibility and you lose sight of the architecture and sacrifice maintainability.

As the industry continues to adopt web applications, it will actively recruit students who have received training in OOD with the UML standard. For this reason, CIS and related programs should make plans for transitioning to object-oriented analysis and design by introducing UML into their curricula. While some faculty resistance to change is anticipated with this approach, this paper is also aimed at providing a reasonable roadmap to such change.

REFERENCES

1. Booch, G., Rumbaugh, J. and Jacobson, I. (1998). The Unified Modeling Language User Guide, Addison-Wesley.
2. Conallen, J. (2003). Building Web Applications with UML, 2nd ed. Addison Wesley.
3. Constantine , L. and Lockwood, L. (1999). Software for Use, Addison-Wesley.
4. Dennis, A., Wixom, B.H. and Tegarden, D. (2002). Systems Analysis and Design: An Object-Oriented Approach with UML, John Wiley and Sons.
5. Gotwals, J.K., Smith, M.W. (1993). "Bringing Object-Oriented Programming into the Undergraduate Computer Information Systems Curriculum," Journal of Information Systems Education, 9/93, Volume 5, Number 3.
6. Hardgrave, B.C. and Douglas, D.E. (1998). "Trends in Information Systems Curricula: Object-Oriented Topics, Proceedings of the Fourth Americas Conference on Information Systems," August 14-16, Baltimore, Maryland, 686-688.
7. Hoffer, J.A., George, J.F., and Valacich, J.S. (2002). Modern Systems Analysis and Design, 3rd ed. Prentice Hall.
8. Johnson, R.A. (2002). "Object-Oriented Systems Development: A Review of Empirical Research," Communications of the Association for Information Systems, Volume 8 Article 4, 65-81.
9. Kendall, K.E. and Kendall, J.E. (2002). Systems Analysis and Design, 5th ed., Prentice Hall.
10. Larman, C. (1988). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice Hall.
11. Rosenberg, D. and Scot, .K. (2001). Applying Use-Case Driven Object Modeling with UML-An annotated e-commerce example, Addison Wesley.
12. Satzinger, J.W., Jackson, R.B., and Burd, S.D. (2002). Systems Analysis and Design in Changing World, Course Technology.
13. Wells, J.D. (2000). "Systems Analysis and Design for E-Business: Implications for Research, Proceedings of the Sixth Americas Conference on Information Systems**,** August 10-13, Long Beach, California, 253-256.
14. Whitten, J.L., Bentley, L.D. and Dittman, K.C. (2000). Systems Analysis and Design Methods, McGraw-Hill.