Object-Oriented System Design

John E. Gessford
California State University, Long Beach

The reason for a different approach to systems analysis and design is explained in cases where an object-oriented approach is to be taken to application programming. The paradigm of the object-oriented approach is shown to be a generalization of the paradigm used in the traditional approach to application software design.

A new object-oriented approach to systems analysis and design is defined and illustrated. The design of an accounts receivable system is used to illustrate the methodology. The object-oriented approach described divides the work into three phases. In the first phase, the application is defined in terms of the entity types and relationships to be included in the system. In the second phase, the business objects to be simulated are identified and categorized as permanent, event, or dependent business objects. In the third phase, the program modules for each business object are specified. A message flow diagramming technique is defined and illustrated that has proven useful in specifying the programs required for business event object classes. The need is pointed out for action diagrams to define processing logic for customized program modules if a CASE tool is to be used to generate the code. The fact that this approach does not use data flow diagrams in providing an overview of the system.

Traditional Approach

The traditional approach to system design is based on three main concepts. One is IPO (Input-Process-Output). The second is structured design. The third is application software automation or CASE (Computer-Aided Software Engineering). The sequence in which the three concepts are listed is the historical sequence in which they appeared.

The input-process-output paradigm of data processing has strongly influenced the traditional approach to system design. The system to be design is viewed as one that will accept some input data, process it in some logical way, and then output results as a report, a screen display, a record sent to a secondary storage device, or a message sent to a remote computer. According to Johnson (1990), first system flow diagrams, and later data flow diagrams, served to define the inputs, general processing logic, and outputs of information systems.

Structured design is a combination of two old military and engineering concepts that have proven useful in dealing with all kinds of systems. One is the technique of working from general plans to more specific designs. This technique is clearly described in Zachman (1987). The other is the objective of standardizing system parts whenever possible. These ideas were first applied to computer programming and then to system design in the 1970’s. Ken Orr (1977), Ed Yourdon (1979) and others led the way in the development of structured system design methodologies. The strategy of working from generalization to detailed designs and using standard constructs in various combinations proved to be wholly compatible with the IPO paradigm. The idea of working from the general to the specific is seen in the use of multiple levels of data flow diagrams. The level 0 data flow diagram depicts the system as a whole and its relationships...
with external systems; level 1, and higher level, diagrams specify the structure interior to the processing steps shown in the level 0 diagram. At the most detailed level of flow diagram, there are only standard programming constructs, if the system is well designed.

The way in which the CASE tools of today automate the traditional structured design and structured programming techniques is described by McClure (1987), Martin (1989) and others. Upper CASE systems allow data flow diagrams to be drawn on a computer screen. The data flow diagrams can be linked to Entity-Relationship diagrams that define the databases involved and to Action Diagrams that define the processing logic of specific processing steps. Screen designs and report definitions can also be specified by CASE tools and linked to the Action Diagrams. The net result is a structured design carried to a level of detail sufficient to allow automated code generation.

**Object-oriented Approach**

The object-oriented approach to system design adds a fourth main concept to the three already described. As defined by Alan Kay (1977), the additional concept is to view the computer as a simulator of objects. By viewing the computer as a simulator of objects, rather than as an IPO system, we more accurately describe its full capabilities. The computer can simulate multiple objects simultaneously. In the object-oriented paradigm, each object has its own set of inputs, processing programs, and outputs. In addition to the traditional forms of input and output, objects can receive and send “messages.” These messages trigger the execution of one of the programs of the receiving object.

This seemingly minor addition to the conceptual foundation leads to radical changes in the way software system design should be approached. It changes the first step that should be taken. Viewing the computer as a simulator of multiple objects (each an IPO system) should change the way we view systems. Rather than starting with a structured breakdown of the processing or a level 0 data flow diagram, the top-down design process should begin with an identification of the objects to be simulated. Then a specification of the simulation of each object can naturally follow, using the basic principles of structured design in an appropriate way.

This need for a change in orientation was recognized to some extent by Sidney Bailin in 1989. The method he proposed, however, still retained too much of the concept of systems as event sequences that can be explicitly described. Data flow diagramming makes this assumption. The method described in this paper does not use data flow diagrams.

**Methodology and Example**

The basic purpose of this paper is to merely describe an object-oriented system design methodology. The conclusion regarding the irrelevancy of data flow diagrams rests directly on this description. No empirical evidence concerning the merits of the methodology is presented, although it is based on the author’s experimentation during a series of system development projects.

The description of the methodology consists of a brief expository discussion of it combined with the presentation of an example of its application. The example used is the design of the software for an accounts receivable system suitable for a consumer product manufacturer.

The example receivables system supports the business functions of (1) issuing sales invoices, (2) receiving customer payments that may cover specific invoiced charges, (3) issuing credit memos, and (4) issuing statements to customers. It also supports the management of accounts receivable with reports on the age distribution of unpaid account balances. This receivables system may not meet all the requirements of a particular business concerning receivables management but it is intended to be realistic enough to show that this object-oriented methodology is capable of handling a complex set of system performance requirements.

**A New Starting Point**

The object-oriented approach to systems design should not begin by identifying or defining the system in terms of either procedures or data flows. It should not begin with data flow diagrams. Gessford (1992) provides evidence that system designs that do begin in this way lead to poor object-oriented designs.

System design should begin with a definition of the application in terms of the entity types and relationships involved. These will become the objects that need to be simulated in the business application. Ideally, these entity types and relationships are a subset of those defined for the business as a whole. This implies that the entity types and relationships of significance to the business already have been defined. Unfortunately, for most businesses, this is not the case. Defining the information requirements of the business in terms of the things (entities) the business needs to know about is quite different from defining them in terms of the accounting transaction documents, reports, messages, and querying capabilities needed to effectively run the business. The entity types and relationships can be derived from these forms of data with which business people are familiar but they are not the same.

Several methods of defining the entity types and relationships important to a business (a business-wide information structure) have been developed over the years. Martin (1989), Scheer (1989) and Gessford (1991) each define an approach to conceptual database design. It is an important topic in information engineering. It need not be treated as part of an object-oriented system design methodology, however. If a BwIS (Business-wide Information Structure) exists then the set of objects used in a particular application can be
Related Content

Mobile Agents Based Self-Adaptive Join for Wide-Area Distributed Query Processing
[www.igi-global.com/article/mobile-agents-based-self-adaptive/3319?camid=4v1a](www.igi-global.com/article/mobile-agents-based-self-adaptive/3319?camid=4v1a)

Concurrency Control for Replicated Data in Distributed Real-Time Systems
[www.igi-global.com/article/concurrency-control-replicated-data-distributed/51162?camid=4v1a](www.igi-global.com/article/concurrency-control-replicated-data-distributed/51162?camid=4v1a)

[www.igi-global.com/article/possibility-theory-framework-security-evaluation/3291?camid=4v1a](www.igi-global.com/article/possibility-theory-framework-security-evaluation/3291?camid=4v1a)

An Analysis of Semantic Overload in Database Access Systems Using Multi-table Query Formulation
[www.igi-global.com/article/analysis-semantic-overload-database-access/51176?camid=4v1a](www.igi-global.com/article/analysis-semantic-overload-database-access/51176?camid=4v1a)