Functional and Object–Oriented Methodology for Analysis and Design

Peretz Shoval
Ben-Gurion University, Israel

Judith Kabeli
Ben-Gurion University, Israel

INTRODUCTION

The chapter provides an overview of FOOM–Functional and Object-oriented Methodology–for analysis and design of information systems. FOOM integrates the functional and object-oriented approaches. In the analysis phase, two main models are created: a) a conceptual data model, in the form of an initial class diagram; and b) a functional model, in the form of OO-DFDs (object-oriented data-flow diagram). In the design phase, the above models are used to design the following products: a) a complete class diagram, including Data, Menus, Forms, Reports and Transactions classes, including their attributes, relationships and methods; b) the user interface–a menus tree; c) the input and output form and report; and d) detailed descriptions of the class methods, expressed in pseudo-code or message charts.

BACKGROUND

Background on Traditional Approach to Information System Development

Many paradigms for system analysis and design have been proposed over the years. Early approaches have advocated the functional approach. Common methodologies that support this approach are SSA and SSD (DeMarco, 1978; Yourdon & Constantine, 1979). SSA is based on the use of data flow diagrams (DFDs), which define the functions of the system, the data stores within the system, the external entities, and the data flows among the above components. Early SSA and similar methodologies emphasized the functional aspects of system analysis, neglecting somehow the structural aspects, namely the data model. This was remedied by enhancing those methodologies with a conceptual data model, usually the entity-relationship (ER) model (Chen, 1976), that is used to create a diagram of the data model, which is later mapped to a relational database schema.

SSD is based on the use of structure charts, which describe the division of the system to program modules as well as the hierarchy of the different modules and their interfaces. Certain techniques have been proposed to create structure charts from DFDs (Yourdon & Constantine, 1979). The main difficulty of an approach where functional analysis is followed by structured design lies in the transition from DFDs to structure charts. In spite of various guidelines and rules for conversion from one structure to the other, the problem has not been resolved by those methodologies (Coad & Yourdon, 1990).

Shoval (1988, 1991) developed the ADISSA methodology that solved this problem. It uses hierarchical DFDs during the analysis stage (similar to other functional analysis methodologies), but the design centers on ***transactions***. A transaction is a process that supports a user who performs a business function, and is triggered as a result of an event. Transactions will eventually become the application programs. Transactions are identified and derived from DFDs: A transaction consists of elementary functions (namely, functions which are not decomposed into subfunctions) that are chained through data flows, and of data-stores and external-entities that are connected to those functions. A transaction includes at least one external-entity, which serve as its trigger. The process logic of each transaction is defined by means of structured programming techniques, for example, pseudo-code.

Based on the hierarchical DFDs and the transactions, ADISSA provides structured techniques to design the user-system interface (a menus-tree), the inputs and outputs (forms and reports), the database schema, and detailed descriptions of the transactions, which will eventually become the application programs. The menus-tree is derived from the hierarchy of DFDs in a semi-algorithmic fashion, based on functions that are connected to user-entities. The design of the forms and reports is based on data flows from user-entities to elementary functions and from elementary functions to user-entities. The design of the relational database schema is based on the analysis of dependencies among the data elements within the data-stores. The data flows from elementary functions to data-stores and from data-stores to elementary functions serve as a basis for defining access-steps, namely update and retrieval operations on the relations. Access-steps are expressed as SQL statements that will be embedded in the program code of the respective transactions. The products of the design stages can be easily implemented using various programming environments.

Background on the Object-Oriented Approach to Information System Development

The development of object-oriented (OO) programming languages gave rise to the OO approach and its penetration into system analysis and design. Many OO methodologies have been developed in the early 1990s (e.g., Booch, 1991; Coad & Yourdon, 1990, 1991; Jacobson, 1992; Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1991; Shlaer & Mellor, 1992; Wirfs-Brock, Wilkerson, & Wiener, 1990). In the OO approach, the world is composed of objects with attributes (defining its state) and behavior (methods), which constitute the only way by which the data included in the object can be accessed. When using the OO approach, a model of the system is usually created in the form of a class diagram consisting of data classes with structural relationships between them (e.g., generalization-specialization), and each class having its attributes and methods.

The early OO methodologies tended to neglect the functionality aspect of system analysis, and did not show clearly how to integrate the application functions (transactions) with the class diagram. Another difficulty with those methodologies was that they involved many types of non-standard diagrams and notations. The multiplicity of diagram types in the OO approach has been a major motivation for developing the UML (Booch, Rumbaugh, Jacobson, 1999;

Figure 1. The initial class diagram of Music Program system