Chapter XI

A Taxonomic Class Modeling Methodology for Object-Oriented Analysis

Il-Yeol Song, Drexel University, USA

Kurt Yano, Drexel University, USA

Juan Trujillo, University of Alicante, Spain

Sergio Luján-Mora, University of Alicante, Spain

ABSTRACT

Discovering a set of domain classes during object-oriented analysis is intellectually challenging and time consuming for novice analyzers. This chapter presents a taxonomic class modeling (TCM) methodology that can be used for object-oriented analysis in business applications. Our methodology helps us discover the three types of classes: (1) classes represented by nouns in the requirement specification, (2) classes whose concepts were represented by verb phrases, and (3) hidden classes that were not explicitly stated in the requirement specification. Our approach synthesizes several different class modeling techniques under one framework. Our framework integrates the noun analysis method, class categories, English sentence structures, checklists, and other heuristic rules for modeling. We illustrate our approach using a detailed case study and summarize the results of several other case studies. Our teaching experience shows that our method is effective in identifying classes for many business applications.

INTRODUCTION

An object-oriented system decomposes its structure into classes. In object-oriented systems, the notion of the class is carried over from analysis to design, implementation, and testing. Thus, finding a set of domain classes is the most important skill in developing an object-oriented system. However, finding classes is a discovery process (Booch, 1993). Discovering a set of domain classes in a problem domain is intellectually challenging and time consuming for novice analyzers. We need systematic methods and guidelines to discover classes.

A class is an abstraction of meaningful real-world objects. A class is a description of objects that share the same attributes, exhibit the same behaviors, and are constrained by the same rules (Starr, 2001). Classes are organized into a class diagram. A class diagram in the Unified Modeling Language (UML) shows classes used in the system and the various static relationships that exist among them. Classes in the class diagram serve as the vocabulary of the object-oriented system, model simple collaborations, and become a basis for the logical database design (Booch, Rumbaugh, & Jacobson, 1999).

A class diagram can be developed at different levels of abstraction. Classes can be *domain (or analysis) classes, design classes*, or *implementation classes*. Domain classes represent important business activities at the analysis level such as Customer or Account. Domain classes are enduring classes regardless of the functionality required today (Stevens & Pooley, 1999). Design classes are those that are added during the design stage to develop an architecture (such as control and boundary classes) or to accommodate design patterns (such as Strategy objects that encapsulate algorithms). Implementation classes are added during the implementation stage and are used to facilitate programming. Examples of implementation classes are String, Tree, Date, or Money. In this chapter, we focus on identifying domain classes that capture fundamental business activities at the analysis level.

In order to discover classes for a problem domain, we have to examine various sources and documentation, and apply various techniques to those specifications. We frequently begin to identify classes from the problem statement or use case descriptions. Rosenberg (1999) states that the best sources of classes are:

- The high-level problem statement
- Lower-level requirements
- Expert knowledge of the problem space

Blaha and Premerlani (1998) recommend that we always begin analysis with a written problem statement. Thus, in this chapter, we assume the modeler has a specification in the form of a problem statement or a use case description in a written form. The statement, written in English, usually defines goals, scope, important functional requirements, and some nonfunctional requirements of the