Chapter VII

How to Transform Legacy Systems into Object Oriented Systems

Hernán Cobo
Universidad Nacional del Centro De la Pcia. de Bs. Argentina, and
Virginia Mauco
Universidad Nacional del Centro De la Pcia. de Bs. Argentina

Introduction

The OO paradigm is the predominant software trend of the 1990s. According to the literature, it provides a unifying model for various phases of development, facilitates system integration, allows prototyping, encourages software reuse, eases system maintenance and provides support for extensibility (Meyer, 1997). An OO system is best developed starting with OO analysis. However, some times this may be difficult because of the existence of so many systems developed 20 or more years ago, which are still used. These systems are called legacy and may be defined as large software systems people do not know how to cope with, but they are vital to organizations (Bennett, 1995). Hence, the decision on how to manage them is crucial because they may represent years of accumulated experience and knowledge. Besides, the software may be the only place where organizations business rules exist.

Maintenance costs are a major issue with software (Pressman, 1992). Legacy systems maintenance is a difficult task because it is typical that during the maintenance process the structure and the documentation of the system deteriorate, making the maintenance
progressively harder. It is essential to record the understanding of the system before it is forgotten and to structure it in such a way that it can be easily accumulated and retrieved. The development of new architectures and the improvements in programming methods and languages, have caused a need to reverse engineer and reengineer existing program code in order to get as much value as possible from legacy systems while exploiting the latest technology.

This chapter describes a project whose aim is to develop a tool to transform legacy systems in order to simplify and improve their maintenance and understanding, taking benefit from OO technology. To achieve this, it is necessary to capture and recover all the knowledge extracted from imperative programs and store it in a higher level structure, which can be analyzed and manipulated. From this structure objects and classes are recognized and extracted to rewrite the program in an OO language. Besides preserving the original functionality, the new code generated should be structured, legible, modular, reusable, and more easily maintainable. The only source of information is programs imperative source code, and its quality has a great influence on the quality of the recovered objects. To minimize this influence, programs are first syntactically restructured and modularized. As part of this research, a prototype has been developed which implements the algorithms to restructure, modularize and extract objects automatically. Human intervention is allowed in order to improve the results.

Background

There has been a lot of work done to improve the quality of legacy code because it has a great impact on legacy systems comprehension, maintenance and evolution. All these efforts may be referred to as software reengineering activities (Arnold, 1994). Reengineering generally includes some form of reverse engineering to achieve a more abstract description, followed by some form of forward engineering or restructuring (Chikofsky & Cross, 1994).

Two different techniques, which are the basis to be used in many reengineering tools, have been defined for translating source code. Program translation via transliteration and refinement is the standard approach. The alternative approach is called translation via abstraction and re-implementation. In this process, the source program is first analyzed in order to obtain a programming-language-independent
Object-Oriented Modeling in UML2
www.igi-global.com/chapter/object-oriented-modeling.uml2/25630?camid=4v1a