A Tool to Reengineer Legacy Systems to Object-Oriented Systems

Hernán Cobo, Virginia Mauco, María Romero, and Carlota Rodríguez

Instituto de Sistemas de Tandil
Depto. Computación y Sistemas - Fac. Cs. Exactas
Universidad Nacional del Centro de la Pcia. de Bs. As. - Argentina
{hcobo, vmauco}@exa.unicen.edu.ar

Abstract. Software evolution is an inevitable process for software systems. Repeated changes alter the structure of a system, rapidly degrading it and making the system "legacy". Reengineering seems to be a promising approach to upgrade these systems according to the latest technologies. This paper describes a tool to reengineer procedural systems written in Cobol, Fortran, C or Pascal, into object-oriented ones written in Smalltalk. The prototype developed identifies potential classes automatically, but allows user intervention to work up conflicts.

1 Introduction

A meaningful number of the systems used nowadays are often many years old and have become reliable over the years. These systems are called legacy and may be defined as large software systems people do not know how to cope with, but that are vital to organisations since they may be the only place where organisations business rules exist [3]. Usually, during the maintenance process, the structure and the documentation of systems deteriorate. Something has to be done to keep these systems up to date and the decision on what to do is critical because they may represent years of accumulated experience and knowledge.

The object-oriented paradigm is the predominant software trend of the 1990's. According to 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 [21]. An object-oriented system is best developed starting with object-oriented analysis. Nevertheless, this may be difficult sometimes because of the existence of many legacy systems. Then, a need to reverse engineer and reengineer existing legacy systems in order to keeping them up with the latest technologies has arisen. There has been a lot of work done to improve legacy code quality because it has a great impact on legacy systems comprehension, maintenance and evolution. All these efforts may be referred to as software reengineering activities [1].

Among the proposals aiming at improving software quality and understanding code restructuring, program modularization and migration from imperative programs to object-oriented ones can be mentioned. Restructuring is one of the oldest and most refined reengineering techniques [1]. The modification of a program control structure to make it follow the rules imposed by structured programming, is one of its associated meanings. Many algorithms have been defined to restructure programs by introducing new variables in them but they always change program topology [5], [19]. In contrast, cliché-based methods can fail in unexpected situations [7], [16], [24].

Program modularization consists of decomposing a monolithic program or module and replacing it with a functionally equivalent collection of smaller modules. Modules should have high cohesion and low coupling. Several methods have been defined to elicit functions from programs and, according to each work goals, these functions are analysed as candidates for reuse or to rewrite the program in a modular way [6], [9], [18]. Many of these works employ program slicing, a program decomposition method well suited for isolating functionalities in a program [27].

The migration from imperative programs to object-oriented ones points to construct a hierarchy of classes that perform the same computations as the original procedures. Each class encapsulates data methods for processing it. Several techniques have been proposed to identify object-like features in imperative programs [13], [17], [20]. The one defined in [20] introduces two methods, one based on global and persistent data, and the other, based on the types of formal parameters and return values. Other approaches pointed to programs written in a specific programming language, like Fortran [23], [26], Cobol [4], [25] or RPG [14]. All these works agree in that transforming an imperative program into an object-oriented one is a difficult task, which cannot be completely automated.

This paper describes the last step of 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 object-oriented technology.

As part of this research, a prototype has been developed which implements the algorithms to restructure, modularise and extract objects automatically. Human intervention is allowed in order to improve the results.

2 The Project

This section presents a project that transforms legacy systems in order to enhance their architecture (Fig. 1). To achieve this, it is necessary to capture and recover all the knowledge extracted from procedural programs and store it in a higher level structure, which can be analysed and manipulated. From this structure, objects and classes are recognised and extracted to rewrite the program in an object-oriented 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 the procedural source code of the programs and its quality has a great influence on the quality of the recovered objects. To minimise this influence, programs are first syntactically restructured and modularised.