A Workbench for Quality Based Software Re-engineering to Object Oriented Platforms

Ladan Tahvildari and Kostas Kontogiannis
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1
+1 519 885 1211
{ltahvild,kostas}@swen.uwaterloo.ca

ABSTRACT

This research presents a re-engineering workbench and architecture that allows for legacy systems written in procedural languages to be migrated to new object oriented platforms. This methodology allows for specific design and quality requirements of the target system to be considered during the re-engineering process through an iterative and incremental process. The migration process recovers an object model from the procedural code and incrementally refines it by taking into consideration the design requirements for the target system.

Keywords
Software Migration, Software Quality, Software Architecture

1. PROBLEM DESCRIPTION

Most of legacy applications have been designed and implemented as monolithic systems using procedural languages and obsolete programming paradigms. An approach towards the solution of the software maintenance crisis is to migrate such legacy systems by re-engineering the code [1]. Over the past few years significant research work has been conducted in the area of reverse engineering and software migration, especially for business applications [4, 5, 7]. However, there is no silver bullet for addressing all the issues involved in understanding unfamiliar source code, especially with the new challenges imposed by the target architecture of the migrated system. This research aims to fill this gap. It means that extending and adapting existing software re-engineering techniques for migrating the legacy systems to the target architecture is one of the main contributions of this research.

Specifically, this research aims to develop a methodology and a workbench where migration of procedural systems to new object oriented platforms can be facilitated. Specific problems that need to be addressed include the recovering of an efficient object model, incorporation of appropriate design patterns [3] for the target system, and conformance with specific re-engineering requirements such as enhancement in performance and maintainability.

2. GOAL STATEMENT

The fundamental changes required in the re-engineering of the legacy systems suggest the need for a comprehensive framework to guide the re-engineering and migration process, both at higher levels (requirements and architecture) and at lower levels (source code). In this research, a legacy system is analyzed at the “architecture level” as well as the “code level”. At the architectural level, we focus on re-engineering the original architecture to a new architecture that conforms with an “intended design” that is provided by the user as a UML (Unified Modeling Language) diagram [10]. At source code level, we focus on re-engineering from a procedural language to an object-oriented one (i.e., C++) and deal efficiently with the issues of encoding polymorphism, overloading, and inheritance in the new target source code. The proposed migration process is incremental in the sense that parts of the system can be migrated at a time, and iterative in the sense that the generated new system can be refined in several steps. The key to the proposed migration approach is exploiting the synergy between requirements analysis, architecture, and reverse engineering. Understanding the architecture of an existing system aids in predicting the impact of changes mandated by the new requirements.

3. PROPOSED APPROACH

The migration of procedural legacy systems to object-oriented systems is divided into two main categories. The first category includes techniques which focus on the identification of objects and abstract data types [2, 8, 12]. The second category focuses on re-engineering and restructuring an existing application in an object-oriented way. This includes source code transformations, clustering, wrapping, and integration with other systems [5, 6, 7].
Our approach builds on top of the techniques presented in [9] as it adopts an incremental and iterative object discovery and migration process. The software analysis tool Refine, which is a Trademark of Reasoning Systems Inc., has been used to develop the prototype tools. First, the source code is parsed and annotated ASTs are created. Annotations on the ASTs include link information, usage information (fetches, stores), and metrics. Second, the global data types and formal parameters are analyzed and collected together as a preliminary candidate class pool. A “re-targetable and dynamic software repository” maintains multiple views of the system under analysis. This repository models and stores a number of high-level descriptions of a software system (e.g., architecture). A set of “generic migration routines” on top of the repository is used for identifying programming patterns, major data types, and related operations on these data types.

Once software artifacts have been understood, classified, and stored during the reverse engineering phase, their behavior can be readily available to the developers during the forward engineering phase in order to generate a new migrant architecture and a new implementation as shown in Figure 1. Population of classes with methods and implementation of the new design and architecture is the next step. We refer to these steps as “target architecture”. At any point of this process, the user can access and evaluate the target object model proposed by the developed tool and edit the evolving model using a UML compliant editor. The proposed process is iterative and incremental. At each iteration cycle, an evaluation procedure is applied to ensure conformance with the re-engineering requirement set. Identifying and classifying types of requirements that are relevant to the target architecture (e.g., maintainability, performance, etc.) for use as generic goals in a systematic goal-oriented framework is the main concern of this work. Potential techniques for addressing these goals will also be collected and organized in a catalogue. Another aspect is to identify, classify, and characterize recurring patterns for a re-engineering context in order to produce a working system that allows for the identification of an object model from the procedural system and the semi-automatic generation of the new migrant object oriented source code.

Once a final target design has been extracted, new C++ code is generated for the new migrant system. The code generation process traverses the ASTs for each class and generates syntactically correct C++ code for .h and .cpp files. In some cases, human intervention is required for obtaining a fully compilable code. However, this intervention involves only minor parts of the system and does not constitute a bottleneck in the migration process. We have obtained some preliminary results by performing experiments on migrating two medium size systems implemented in the C language to a new object-oriented architecture and implementation using C++. The migration process considered enhancement in the maintainability and performance as requirements for doing the migration [11].

4. SUMMARY

We have proposed an architecture for establishing requirements-driven workbenches re-engineering. As part of the proposed workbench, we have concentrated on different levels of abstraction, from code level to the architectural level. At lowest levels, migration takes the form of transforming (or transliterating) the code from one language into another. At higher levels, the structure of the system may change as well to make it more object-oriented. At this higher levels, this global architecture of the system may change as part of the migration process. This comes from the fundamental changes required in the re-engineering of the legacy systems and the need for a comprehensive framework to guide the re-engineering and migration process.

5. REFERENCES