Quality Driven Software Migration of Procedural Code to Object-Oriented Design

Ying Zou
Dept. of Electrical & Computer Engineering
Queen's University
Kingston, ON, K7L 3N6, Canada
zouy@post.queensu.ca

Abstract

In the context of software maintenance, legacy software systems are continuously re-engineered in order to correct errors, provide new functionality, or port them into modern platforms. However, software re-engineering activities should not occur in a vacuum, and it is important to incorporate non-functional requirements as part of the re-engineering process. We present an incremental reengineering framework that allows for quality requirements to be modeled as soft-goals, and transformations to be applied selectively towards achieving specific quality requirements for the target system. To deal with large software systems, we decompose the system into a collection of smaller clusters. The reengineering framework can be applied incrementally to each of these clusters and results are assembled to produce the final system.

1. Introduction

Legacy systems refer to mission critical software systems that are still in operation, but their quality and expected operational life is constantly deteriorating due to prolonged maintenance and technology updates. Therefore, legacy systems become harder to understand and maintain after many years of evolution. To leverage business values entailed in such systems, a possible solution is to migrate selected parts of such systems to modern platforms and designs. One such possible solution is related to object oriented re-engineering whereby a procedural legacy system or component is migrated towards a modern object oriented design. With properties, such as information hiding, inheritance and polymorphism inherent in object oriented designs, essential parts of such a reengineered system can be reused or integrated with other applications using network centric Web technologies, enterprise integration solutions, or distributed systems.

In this context, the software reengineering community has already proposed a number of different methods to migrate procedural code into object-oriented platforms. In a nutshell, the existing migration methods aim to identify Abstract Data Types (ADT) and extract candidate classes and methods from the procedural code. These methods include concept analysis [1, 2], cluster analysis [3], slicing [6], data flow and control flow analysis [7], source code features [5], and informal information analysis [4]. However, existing migration techniques do not provide a comprehensive framework for ensuring that the migrant systems will posses certain quality characteristics. There are always questions posed and concerns raised regarding whether the newly migrant system will posses as good or better characteristics as the original system. For example, whether the new system can run as fast as the old system or whether the new system is as maintainable as the old one. Non-functional requirements, such as reusability, maintainability, performance, portability and, security define system properties, constraints and software qualities of the system being developed. These non-functional requirements play an important role in the software development and evolution process. Their achievement in the source code ensures the success of software products, and ease software maintenance process. Unfortunately, as pointed out by Lehman's laws of evolution, the quality of an evolving program tends to decline, and its structure becomes more complex [8]. Therefore, it is important to incorporate non-functional requirements in the migration process. Moreover, legacy systems are often large with million line of code. It is a challenging task to devise a tractable and incremental methodology where source code transformations can be associated with specific quality improvements and can be applied for the migration of large-scale procedural systems to the object-oriented platforms.

2. Proposed Approach

We are interested in transforming a system from its original procedural language implementation to an object-

oriented design without altering its external behaviors. Moreover, we consider target software quality goals as an integral part of the migration process. The migration process consists of two major phases.

The first phase focuses on representing the procedural code of the system being analyzed in terms of an annotated Abstract Syntax Tree (ASTs) or entity relationship model. The software entities represent source code features of interest, such as data types, formal and actual parameters, global variables, and functions. The relations denote interactions between software entities, such as function calls, global variable usage, or data type references. The central issue is to represent the procedural code of the system being analyzed in a generic representation. For different procedural language domains, we can provide a set of generic transformations that migrate procedural systems in different procedural languages into functionally similar object-oriented systems.

High-level procedural languages, such as C, COBOL, Fortran and Pascal, all comply with different domain specific models that denote the structures and to a certain extent the semantics of each language. However, a variety of procedural languages have semantic equivalent language constructs. We introduce the concept of a unified domain model for a variety of procedural languages such as C, Pascal, COBOL, and Fortran. This unified model is represented in XML and denotes common language features such as routines, subroutines, functions, procedures, types, statements, variables and declarations, just to name a few.

The second phase aims at the extraction of an object model by a series of transformations applied at the XMLized AST representation. We proposed a quality driven software migration framework that monitors and evaluates software qualities at each step of the migration process. Furthermore, we consider a migration process as a state transition system, denoted by a sequence of transformations that alter a system being migrated. A transformation detects a procedural code feature, and generates the desired migrant artifact. For example, the existence of global variables violates the concept of encapsulation in object-oriented design. In this case, global variables in procedural code are captured and removed in order to conform to target quality goals for the new system. Each transformation is selected to alter source code features in the original system, and is assessed according to its potential impact on the desired qualities for the target system. Consequently, the resulting system is evaluated against the desired goals. Finally, the process terminates either when the highest achievable measured level of quality for the target system has been reached or no further transformations can be applied.

To keep the complexity and the risk of the migration of large system into manageable levels, we provide an incremental approach that allows for the decomposition of legacy systems into smaller manageable units (clusters). The quality driven migration approach is applied to each such cluster to identify an object model with the highest quality. Then an incremental merging process allows for the amalgamation of the different partial object models into an aggregate composite model for the whole system. In this way, the proposed quality driven migration framework can tackle large systems in a reasonable hardware and software resource requirements.

3. Thesis Contributions

The proposed techniques and methodologies largely automate the migration of procedural systems into object-oriented platforms and incorporate quality attributes in the reengineering process. The contributions of this thesis are summarized in this section.

3.1 Modeling Quality Requirements

To drive the migration process to meet specific quality requirements (such as high maintainability), we provide a software quality modeling process that elicits quality goals, models these quality goals in a measurable level, and evaluates the achievement of these desired qualities in the final migrated system. Typically, quality requirements are modeled in a top down manner, which involves identifying a set of high-level quality goals, such as functionality, maintainability, as specified in ISO 9126, subdividing and refining these goals into more specific low level attributes (e.g., design decisions, or software code features). We adopt soft-goal interdependency graphs [9] to associate high-level quality goals with specific source code features. In such graphs, nodes represent design decisions, and edges denote positive or negative dependencies towards a specific requirement. The leaves of such graphs correspond to measurable source code features that impact other nodes to which they are connected. A set of metrics are selected to compute the corresponding source code features, appearing as leaves in the soft-goal interdependency graph. The metric results of the leaf nodes indirectly reflect the satisfaction of their direct or indirect parent nodes in the soft-goal interdependency graph.

3.2 Transformation Rules

In our research, we apply unified transformation rules on various procedural language domains, and generate object-oriented code. Within this context, we create a generic code representation for procedural languages. Therefore, transformations and source code features can be described abstractly in our generalized procedural code representation. A transformation rule is described with pre and post- conditions, using UML and OCL. A comprehensive, yet extensible list of possible

transformations are identified for the purposed of migration. We divide transformation rules into two categories, namely, object identification and object model refinement.

The rules in the object identification category define the criteria for the production of object models from the procedural code. In the search of an object model to extract from procedural source code, we aim at achieving high encapsulation, high cohesion within a class, and low coupling between classes. The object identification techniques focus on two areas: a) the analysis of global variables and their data types, and b) the analysis of complex data types in formal parameter lists.

The rules in the object model refinement category are used to identify the inheritance and polymorphism relations between abstract data types. To identify inheritance relations, we examine procedural source features, such as, type casting, a *struct* type defined inside other *struct* types and date field clones among *struct* types. To discover polymorphism relations, we examine procedural code features, such as function pointers, switch statements, conditional statements, and generic pointer parameters.

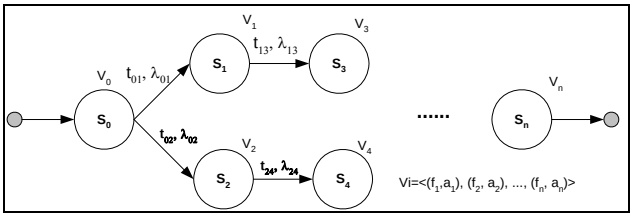

Figure 1: Quality Driven Software Migration Framework

3.3 Quality Driven Migration Approach

The migration process can be constructed as a set of non-empty states **S**, including, s_0, s_1, …, s_i, s_{i+1}, …, s_n, as shown in Figure 1. State, s_0, represents the original software system. Other than the initial state, the set of states, **S** can contain more than one final states that correspond to different resulting migrant systems (i.e. alternative object models), or to an empty state that denotes failure. The states between the initial state and final states represent snapshots of the migration process whereby a mix of procedural and object model exist as the system progress from its original procedural state to a fully object-oriented final state. Each transformation, t_{ij}, represents a transformation moving from s_i to s_j. Each transformation, t_{01}, t_{02}, …, t_{ij}, $t_{i,j+1}$, …, t_{kn}, alters a state and yields a consecutive state, and converts a software system in a stepwise fashion from its initial state (original procedural system) to a final state (new object-oriented system). Each transformation affects the source code features, appearing as leaves in the quality soft-goal graph. We consider transformations having a positive impact on the modeled quality. We associate each transformation with a likelihood, λ_{ij}, which indicates that a transformation contributes towards a desired quality goal.

The objective thus is to identify the optimal combination of transformations that yields the best quality requirements for the target migrant system. A likelihood of a transformation path can be calculated by multiplying the likelihood of each transformation along the path. An optimal transformation path has the highest likelihood towards achieving the desired qualities. Finally, the migrant system is chosen from the optimal path. We adapted the Viterbi algorithm [11] to generate all possible transformation paths and identify the optimal sequence of transformations. A draw back of using the Viterbi algorithm is the need to retain all states and paths before identifying an optimal path. In our research, we developed a number of heuristics to reduce the number of generated states and the time required to find the optimal path. These heuristics are essential to permit our approach to migrate large legacy systems with reasonable resource requirements. For example, when a segment of the search tree contains a sequence of states in a single path, these states can be collapsed into one composite state. The likelihood of the composite state is equal to the multiplication of the likelihoods of transformations along the single path inside the composite state.

3.4 Incremental Migration Process

Legacy software systems are usually large systems. Our proposed approach must scale well to handle the migration of these systems. We break a large system into a set of clusters, each of which contains the maximum number of source code entities that are related to a class candidate. These clusters have the least data dependencies on other clusters. This would enable us to migrate clusters independently. Moreover, the order of selecting which cluster to migrate would have no effect on the final object model. In a nutshell, the proposed quality driven migration approach is applied iteratively to every cluster. In each cluster, the migration process is divided into a sequence of states and transformations. During the migration of the first cluster, the initial state represents purely procedural code of the cluster. Once the optimal path in a cluster is identified, the final state from the optimal path in a cluster serves as the initial state for the next cluster. The object model is incrementally expanded by identifying additional classes from the new clusters.

3.5 Quality Evaluation Techniques

For this work, we consider reusability and maintainability as desired qualities of the target system. We define quality evaluation techniques to verify that the quality driven migration approach can generate object-oriented system

with the highest maintainability and reusability. Several case studies are conducted as follows:

- We select other possible migrants systems and compare their quality with the migrant system generated from the optimal transformation path.
- To compare the quality of systems, we collect a set of object-oriented metrics to quantify direct measurable features, for example, coupling between classes and cohesion inside a class. The high cohesion and low coupling further indicate high maintainability and reusability.
- We analyze and compare the metric results from different systems.

To select other possible migrations of the same systems, we use the transformation path presented in our incremental migration process. All transformation paths are generated and ordered in a sequence based on their attached likelihood scores. Several representative target systems can be generated from different paths.

To assess whether the desired goals in the target system have been achieved, a set of widely accepted object-oriented software metrics are adopted. We aim to achieve high cohesion and low coupling in the target object-oriented systems. In this respect, coupling between classes can be measured using metrics, such as, CBO (Coupling Between Objects), DCC (Direct Class Coupling) and IFBC (Information Flow Between Classes)[10]. Moreover, the cohesion inside a class can be measured using metrics, such as, TCC (Tight Class Cohesion), Coh (Cohesion Measurement) and IFIC (Information Flow Inside Class) [10].

To reflect the overall value of the entire system in terms of one specific metric, we calculate the average value of each metric for each class. These set of metrics are different from the ones used to compute the transformation likelihoods. By combining results from all the metrics, we see that the final object-oriented system generated from the optimal path has the highest qualities in comparison to the object-oriented systems generated from the alternative paths.

4. Conclusion

In the context of software maintenance, legacy software systems are continuously re-engineered in order to correct errors, provide new functionality, or port them into modern platforms. However, software re-engineering activities should not occur in a vacuum, and it is important to incorporate specific non-functional requirements for the migrant system as part of the re-engineering process. The proposed techniques and methodologies largely automate the migration of procedural systems into object-oriented platforms and incorporate quality attributes in the reengineering process. The obtained results demonstrate the effectiveness, usefulness, and scalability of our framework. The proposed approach is applied in migrate several legacy systems, such as UNIX Bash Shell, Apache Web Server, and Clips expert system, written in C and Fortran to C++. On-going work focuses on applying the proposed framework in more generic software transformation context, such as refactoring, restructuring and development process.

5. Papers Published in the Context of this Thesis

1. Ying Zou, Kostas Kontogiannis, “Incremental Transformation of Procedural Systems to Object Oriented Platforms”, to appear in the IEEE 27th Annual International Computer Software and Applications Conference (COMPSAC), Dallas, Texas, USA, November 3-6, 2003.
2. Ying Zou, Kostas Kontogiannis, “Quality Driven Transformation Framework for Object Oriented Migration”, the 2nd ASERC Workshop on Software Architecture, Banff, Alberta, Canada, February 18-19, 2003.
3. Ying Zou, Kostas Kontogiannis, “Quality Driven Transformation Compositions for Object Oriented Migration”, the 9th IEEE Asia Pacific Software Engineering Conference (APSEC), Gold Coast, Queensland, Australia, December 2002 , pp. 346-355.
4. Ying Zou, Kostas Kontogiannis, “Migration to Object Oriented Platforms: A State Transformation Approach”, the 19th IEEE International Conference on Software Maintenance (ICSM), Montreal, Quebec, Canada, October 2002, pp.530-539.
5. Ying Zou, Kostas Kontogiannis, “A Framework for Migrating Procedural Code to Object-Oriented Platforms”, the 8th IEEE Asia-Pacific Software Engineering Conference (APSEC), Macau SAR , China, December 2001, pp. 408-418.
6. Ying Zou, Kostas Kontogiannis, “Towards A Portable XML-based Source Code Representation”, International Conference on Software Engineering (ICSE) 2001Workshops of XML Technologies and Software Engineering (XSE), Toronto, Ontario, Canada, May 2001.
7. Ying Zou, Kostas Kontogiannis, “Towards A Web-Centric Legacy System Migration”, International Conference on Software Engineering (ICSE) 2001, 3rd International Workshop on Net-Centric Computing (NCC): Migrating to the Web, Toronto, Ontario, Canada, May 2001.
8. Ying Zou, Kostas Kontogiannis, “Migrating and Specifying Services for Web Integration”, In Lecture Notes in Computer Science LNCS vol. 1999, Springer-Verlag, 2001, pp. 244-260.
9. Ying Zou, Kostas Kontogiannis, “Migration and Web-Based Integration of Legacy Services”, the 10th Centre for Advanced Studies Conference (CASCON), Toronto, Canada, November 2000, pp. 262-277.
10. Ying Zou, Kostas Kontogiannis, “Migrating and Specifying Services for Web Integration”, 2nd International Workshop on Engineering Distributed Objects (EDO), University of California at Davis, California, USA, November 2000.

11. Ying Zou, Kostas Kontogiannis, “Web-based Legacy System Migration and Integration” the 4th International Conference in Systematic, Cybernetics and Informatics (SCI), Orlando, USA, July 2000, pp. 254-260.
12. Ying Zou, Kostas Kontogiannis, “Localizing and Using Services in Web-Enabled Environments”, Workshop on International Web Site Evolution (WSE), Zurich, Switzerland, March 2000.
13. Prashant Patil, Ying Zou, Kostas Kontogiannis and John Mylopoulos, “Migration of Procedural Systems to Network-Centric Platforms”, the 9th Centre for Advanced Studies Conference (CASCON), Toronto, Canada, November 1999, pp. 68-82.
14. Ying Zou, Kostas Kontogiannis, “Enabling Technologies for Web-Based Legacy System Integration”, Workshop on International Web Site Evolution (WSE), Atlanta, USA, October 1999.

Acknowledgment

I would like to thank my Ph.D. thesis supervisor, Dr. Kostas Kontogiannis for his valuable contributions and support.

References

[1] C. Lindig and G. Snelting, “Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis”, In Proc. Of International Conference on Software Engineering, 1997.

[2] H. A. Sahraoui, W. Melo, H. Lounis, F. Dumont, “Applying Concept Formation Methods To Object Identification In Procedural Code”, In Proc. Of 12th Conference on Auotmated Software Engineering, 1997.

[3] S. Mancoridis, B.S. Mitchell, Y. Chen, and E. R. Gansner, Bunch: a clustering tool for the recovery and maintenance of software system structures, In Proc. Of International Conference on Software Engineering, 1999.

[4] Letha H. Etzkorn, Carl G. Davis, “Automatically Identifying Reusable OO Legacy Code”, Computer, IEEE, October, 1997.

[5] K. Kontogiannis, P. Patil, “Evidence Driven Object Identification in Procedural Systems’’. STEP’99, September 1999, pp. 12-21.

[6] Filippo Lanubile, and Giuseppe Visaggio, “Extracting Reusable Functions by Flow Graph-Based Program Slicing”, IEEE Transactions on Software Engineering, Vol. 23, No. 4, April, 1997.

[7] De Lucia, G.A. Di Lucca, A.R. Fasolino, P. Guerra, S. Petruzzelli, “Migrating Legacy Systems toward Object Oriented Platforms”, 1997, IEEE.

[8] M.M. Lehman, "Programs, life cycles, and laws of software evolution", Proc. IEEE, v. 68, 1980.

[9] John Mylopoulos *et. al.*, “Representing and Using Nonfunctional Requirement: A Process-Oriented Approach”, IEEE Transactions on Software Engineering, v. 18, n. 6, June 1992.

[10] Lionel C. Briand, J¨urgen Wst, and Hakim Lounis. Replicated case studies for investigating quality factors in object-oriented designs. Empirical Software Engineering: An International Journal, 6, 2001.

[11] Paul Alphen and Dick Bergem. Markov models and their application in speech recognition. Technical report, Reader Colloquium Signaalanalyse en Spraak, IPO report 765, 1992.