An Overview of the State-of-The-Art Reverse Engineering Techniques

László Angyal, László Lengyel, Hassan Charaf

Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Goldmann György tér 3, H-1111 Budapest, Hungary
{angyal, lengyel, hassan}@aut.bme.hu

Abstract: Nowadays the development without model-based approaches are hardly imaginable, because models are not only closer to human thinking but also help the communication between developers. During a long development process the initial model becames inconsistent with the code that can be synchronised manually or automatically by tools. Reverse engineering tools have been created to help developers achieving that the design and the implementation harmonize again. This work examines the importance of the model-based development and gives an overview of the state-of-the-art reverse engineering methods and tools. Round-trip engineering is a more advanced approach of software development than reverse engineering, because the changes that affect the design are made not in the code but in the model, hereby better software quality can be achieved.

Keywords: model-based development, reverse engineering, round-trip engineering, tools, design recovery

1 Introduction

Model-based development plays an important role in managing complexity of software systems. Visual modeling of software systems represents the desirable behaviour of the system in a higher abstraction level, which can be an effective way to make the software design process more efficient. Well-constructed models are unambiguous, easy-to-understand, and enable communication between developer team members. By modeling software, developers understand the design and the associated risk, managers are able to make resource planning.

General purpose modeling languages are insufficient for modeling a domain specific activities like business process modeling. Domain-Specific Modeling [1] (DSM) is a way of designing and developing systems in a higher abstraction level, closely focusing on the problem domain and usually applied together with generative programming. Using very expressive DSM languages can dramatically

improve code quality due to automated source generation, and increase productivity.

The model represents the design state of a system, but the source code represents the implementation state of the same system. Automation of synchronizing these states is important to avoid error prone and tedious human tasks, and to reduce costs. Certain models such as the most UML diagrams lets the developers gain better understanding of the functionality and system behaviour but only a small amount of these models can be mapped to source code.

Model-Driven Architecture (MDA) [2] is the OMG's software development approach, which increases the role of models in development process. It uses models with different level of abstraction to design, to modify, and to maintain software systems. The models with the highest level of abstraction called Platform-Independent Models (PIMs) that describe the business rules and independent of any implementation technology. Next step is that PIMs are transformed into Platform-Specific Models (PSMs) that describe how the system uses a technology platform. PSMs are closer to the implementation of the system. The final step is to transform PSM models to executable code. Since the entire MDA process is driven by PIMs, there will be far less manual coding, the quality of the software will be higher and the development process focuses the high-level, technology independent modeling and parts of that can be reused in other projects. MDA concept is useless without tools automating the model transformations.

Model-Integrated Computing (MIC) [3] [4] is a technique that converts domain-specific models into executable code. MIC provides a framework for software production using both metamodeling environments and model interpreters. MIC supports the flexible creation of modeling environments, and helps tracking the changes of the models.

Model-based development (MBD) is an increasingly applied method in producing software artifacts. Model-driven development approaches emphasize the use of models at all stages of system development. In model-based development, models are used to describe all artifacts of the system, i.e., interfaces, interactions, and properties of all the components that comprise the system. These models can be manipulated in a number of different ways to analyze the system, and in certain cases to generate the complete implementation of the system. In order to capture the semantics that is as close as possible to the domain of the developed system in an effective manner, building a domain-specific modeling language is a suitable choice. Using domain concepts to modeling systems helps increase productivity, makes systems easier to maintain and evolves and shortens the development cycle.

The current work focuses on reverse engineering approaches and tools. The rest of the paper is organised as follows: Section 2 presents the importance of round-trip engineering. In Section 3, we review existing reverse engineering tools and approaches. Finally, conclusions and future work are elaborated.

2 Supporting Round-Trip Engineering with Model-based Development

The position of round-trip engineering [5] is rising in software development projects, many visual modeling tool is not limited to depicting the design, these tools support source code generation from visual models and existing code transformation into these kinds of models to make changes on the design than regenerate the code.

The structure and behaviour of the system defined by the model can be translated by tools into source code. This process is referred to as forward engineering. The idea behind forward engineering is that the model as a special graph can be traversed, and source code can be generated from the incorporated information. Generally, forward engineering starting with a model generates one or more software artifacts that are closer in detail and level to the final implementation of the software. MDA has been proposed for forward engineering, where abstract models are created by the developer.

Reverse engineering of a source code regenerates the model of a system graphically on a higher abstraction level, therefore, the structure is easy to conceive, and the relations between components can be discovered. More generally, reverse engineering is applied to generate a more abstract model from a software artifact.

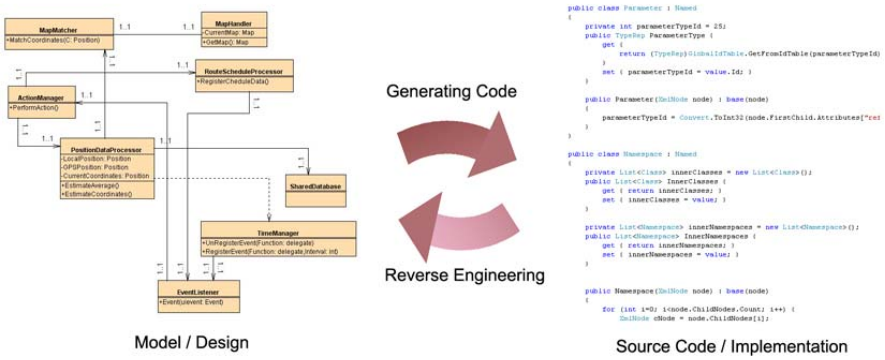

Figure 1
Forward and reverse engineering

In efficient and optimal cases forward and reverse engineering performs only incremental transformations, the changed parts of the model is transformed instead of the whole model.

Round-trip engineering applies together forward and reverse engineering, in order to synchronize source code and model: any changes of the source code are

synchronized back into the model, the source code stay consistent with any changes of the design model. The goal of this approach is to minimize the distance between different representations of the system. Round-trip engineering provides a technique for enabling the developers to move freely between source code and model, this supports iterative development approach. After synchronizing the model with the revised code, developers may choose the best way to work: modify the code further or change the model.

A software development project can be split into iterations instead of sequential phases. An iteration involves all the phases, and a portion of the project is implemented as an executable in an iteration. After each iteration, developers have a working software to use and learn from it and feedback into the next iteration of the project. Iterative and incremental development (IID), due to the low development risks, applied in most development process. Advantages of IID are lower risks, errors come into sight sooner, easier to test new functions. Round-trip engineering automates the source code and design synchronization between iterations.

Since data can be lost during model transformations, forward engineering and also model transformation executions may produce a trace model of the transformation to keep the correspondance between the source and the target elements. Trace information is used in code synchronization, allows incremental transformation. Furthermore, when the relationship between source and target model is not bijective (not one-to-one), trace information is necessary to be able to trace back the original state or perform round-trip engineering. OMG's Query/View/Transformation (QVT) specification [6], which is supposed to become the MDA's standard language for model transformation, contains a proposal how to deal with the trace information. An instance of the trace class stores the relationship between models established by a transformation execution.

3 Existing Tools and Approaches

This section introduces the most relevant tools and approaches that supports round-trip or reverse engineering.

3.1 CASE Tools

Most CASE tools can reverse engineer class diagrams, but there is lack of tool support for extracting other diagrams. Popular CASE tools like Rational XDE [7], Borland Together [8], Eclipse GMT [9], and Fujaba [10] support round-trip engineering. Borland's Together is one of the leader in commercial round-trip enigineering tools. LiveSource [8] technology automatically synchronizes models

and code, thereby the model is based directly on the source code itself, the UML class diagram will be a view of the implementation. By model changes the source code is automatically updated, the design is always up-to-date.

Eclipse is an open source framework, which can be extented to be a complete integrated developing environment by external plugins. Several model transformation tools are implemented as Eclipse plugin, for example UML modeling tools including simple round-trip engineering support. Borland Together is available as an Eclipse plugin. Eclipse Generative Modeling Tools (GMT) is a set of tools in the area of model driven development. ATLAS transformation language (ATL) [11] plugin provides a set of transformation tools for GMT. An ATL transformation program specifies the rules how to match the elements of source model and how to create elements of target model. ATL supports traceability using a developer predefined trace metamodel, this approach differ from QVT. The traceability approach of ATL elaborated in [12] is based on the fact that an ATL program is just a model and can transformed by an other ATL program, which automatically inserts transformation rules that creates trace model elements. The traceability links are described by trace models.

MetaEdit+ [13] is az integrated modeling and metamodeling environment for DSLs. In MetaEdit+, developer can define a domain-specific modeling language with roles, constrains and specify the mapping these elements to code fragments in domain-specific generator. The model execution can be emulated, the generated code and the model elements are linked to each other, MetaEdit+ shows which model elements is under execute and also shows the corresponding source code.

Fujaba System elaborated in [10][14] introduces Story-Driven Modeling as a new method for software development, which is based on a high-level visual programming language called story diagrams. Fujaba is a round-trip engineering environment that create class diagrams and story diagrams from extracting the abstract syntax graph of a Java source code. Story diagrams can visualize the dynamic aspects of a system as a control flow, adapt UML class, activity, and collaboration diagrams and can be translated to Java. Story diagrams support two kinds of activities statement activities and story patterns. Story pattern is a graph rewriting rule [15] that represents a complex Boolean condition. The Fujaba generator translates story diagrams into method bodies of the classes depicted in class diagrams.

3.2 Source Code Parsers and Tools

The input of many reverse engineering tools is files or abstract syntax trees (AST) that have been created by other tools such as parsers and fact extractors. These reverse engineering tools do not perform the extraction of the source code but focuses only on making abstraction and visualizing of these outputs.

Columbus [16] that is a reverse engineering framework contains a C/C++ source code analyzer plugin that can extract even large C/C++ projects into UML diagrams, AST, and call graph. Columbus also contains design pattern identification and code auditing tool.

The developers of Columbus have been presented a standard exchange schema (C/C++ AST scheme) that is able to completely describe the AST of any C/C++ source code. This scheme makes the interoperability of other applications easier. An XML document format CPPML (C++ Markup Language) has been also presented that has a structure based on the schema, that can be processed by third party tools easily. The original source can be regenerated using this file.

CPP2XMI [17] is a reverse engineering tool which allows extracting UML class, sequence, and activity diagrams in XMI format from C++ source. The tool processes the CPPML output of Columbus/CAN fact extractor and visualize it by its Layout Creator module in 2D or 3D views. The idea behind this CPP2XMI is that CPPML output file contains all information from the AST and it can be used to generate UML sequence and activity diagrams. From object allocations and function calls, sequence diagram can be created. The conditional and iterative statements are important for activity diagram generation, the XMI output of Columbus contains all information about class diagrams.

The Rigi reverse engineering system [18] is an interactive visualization tool of software structures, it analyzes the dependencies between software artifacts from the source code. The relations of procedures, procedure calls, data accesses, variables and among others data and control-flows are discovered and stored in Rigi graph model (Rigi Standard Format - RSF) than visualized in scaleable hierarchical graph diagrams. RSF is an intermediate data format and processed by many reverse engineering tools.

SHriMP (Simple Hierarchical Multi Perspective) [19] is an application and a visualization technique of exploring hierarchical stuctures like software code. Bookshelf [20] is a set of tools that aims visualization and navigation on the information of large software systems.

GUPRO [21] uses own graph exchange language (GXL [22]) which is widely supported XML schema for fact exchange between reverse engineering tools. GUPRO is an integrated workbench strongly based on graph modeling and algorithms. Because source code is stored in internal repository graph, the developer can choose a code representation or an analyzis technique. The objects in the repository graph can be queried and browsed using GReQL language. Query results can be displayed in tables or pointed in the source code directly.

One of the most popular lexical analyzer and parser generators is JavaCC (Java Compiler Compiler) [23] that can handle any programing or non programing languages. JavaCC needs a grammar specification to provide Java classes that can parse any source that matches to that grammar.

3.3 Design Pattern Recognition

Another research trend of reverse engineering is design pattern recognition from source code. A design pattern [24] is a reusable object oriented software design artifacts that solves a problem in particular context. Pointing out the presence of well known design patterns in an architecture making faster the understanding the design considerations of a software system. There are several different approaches to identify patterns in source code. Design patterns can be identified by among others inter-class relationship in method call, data-flow analyzis, by fuzzy logic, graph matching or formal semantic.

Pattern recognition is also suitable for measuring software quality [25], because not only desing patterns, but also anti-patterns [26] can be detected in the implementation, thus, bad design considerations or weakness of the code can be discovered. Similarly to design patterns, anti-patterns are piece of reusable code, but applying these kinds of patterns shuld be avoided. CrocoPat [27] tool does graph search, it processes RSF (Rigi Standard Format) files that contains the graph of a system than uses own imperative language to find the predefined patterns between class inheritance relations and method calls. Columbus uses graph matching algorithms. Other methods are also available, such as PtideJ [28], which uses constraint solving or SPOOL [29] , which uses database query. PINOT [30] pattern inference and recovery tool reclassifies the GoF patterns and implements a lightweight static inter-class and data-flow analysis.

The evolution of these pattern recognition tool is going towards faster, minimized misrecognition and ability to find more patterns.

Conclusions and Future Work

Current paper has introduced the importance of the model-based development, we have pointed out that round-trip engineering improves software quality and effectiveness of development. Reverse engineering approaches are aiming greater understandability of software systems, which involves up-to-date documentation, consistent models. We have examined the most efficient tools and approaches that support either the whole or a part of the round-trip engineering. We have shown that the model-based development approaches without comprehensive tools are hardly usable in practice. Table 1 compares the above-mentioned CASE tools and Table 2 compares the enumerated reverse engineering tools in certain aspects.

As future work we plan to improve the presented approaches to create a more efficient both reverse and round-trip engineering tool that can be used as a designing environment which applies two-directional validated source to model transformations.

Case Tools	Languages	Transformation	Modeling
Borland Together	Java, C++, VB, C#	source code - UML Class, database model-physical DB synchronization, source code to UML sequence, UML Class to any model with QVT	UML Class, database
Rational XDE	many, including Java, Delphi, VB, C#	source code - UML Class, database model-physical DB synchronization, source code to UML sequence	UML Class, database
Eclipse GMT	language independent	model-based transformations	models, metamodels
MetaEdit+	language independent	general model to source code, API for user written reverse engineering program	any domain-specific model
FUJABA	Java	Story-Diagrams + UML Class - executable source code round-trip	Story-Diagram, UML Class

Table 1
Comparison of CASE tools

Tools	Input data	Technique	Diagram Construction	Pattern Recognition
Columbus	C++	graph matching	UML Class	+
CPP2XMI	CPPML, XMI	dynamic analysis	UML Sequence, UML Activity, UML Class	-
Rigi	C++	dependency-, relational analysis	hierarchical graphs	-
SHriMP	Rigi RSF	visualization	hierarchical view of graphs	-
Bookshelf	Rigi RSF	function call, variable, cross file analysis	software landscape	-
GUPRO	GXL	graph query	-	-
CrocoPat	Rigi RSF	graph search	visualize graphs in 2D, 3D	+
FUJABA	Java source	dynamic analysis, fuzzy logic	Story-Diagram + UML Class	+
SPOOL	UML-based CDIF [31]	database query	UML Class, HTML	+
PINOT	Java	data-flow-, inter-class analysis, consider the usage of java utility classes	-	+
PtideJ	Java byte-code	constraint solver	+	+

Table 2
Comparison of reverse engineering tools

Acknowledgement

The fund of ‘Mobile Innovation Centre’ has supported in part, the activities described in this paper.

References

[1] Jonathan Sprinkle and G´abor Karsai. A Domain-Specific Visual Language For Domain Model Evolution. Journal of Visual Languages and Computing, 15(2), April 2004

[2] MDA Guide Version 1.0.1, OMG, document number: omg/2003-06-01, June 12, 2003, www.omg.org/docs/omg/03-06-01.pdf

[3] J. Sprinkle, Model-Integrated Computing, IEEE Potentials, 23(1):28-30, 2004

[4] J. Sztipanovits, G. Karsai, Model-Integrated Computing, IEEE Computer, 30(4):110-111, 1997

[5] S. Sendall, J. M. Küster: Taming Model Round-Trip Engineering, Proceedings of Workshop on Best Practices for Model-Driven Software Development (part of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications), Vancouver, Canada, October 25, 2004

[6] Transformation Specification Object Management Group Query, View. http://www.omg.org/cgi-bin/apps/doc?ad/05-03-02.pdf

[7] Rational XDE home page: http://www-128.ibm.com/developerworks/rational/products/xde

[8] Borland Together home page: http://www.borland.com/us/products/together

[9] Eclipse GMT home page: http://www.eclipse.org/gmt/

[10] U. Nickel, J. Niere, J. Wadsack, A. Zündorf: Roundtrip Engineering with FUJABA. Proceedings of 2nd Workshop on Software-Reengineering (WSR), Bad Honnef, Germany(J. Ebert, B. Kullbach, and F. Lehner, eds.), Fachberichte Informatik, Universität Koblenz-Landau, August 2000

[11] Eclipse ATL home page: http://www.eclipse.org/gmt/atl/

[12] Jouault F. Loosely Coupled Traceability for ATL. In: Proceedings of the European Conference on Model Driven Architecture (ECMDA 2005) workshop on traceability, Nuremberg, Germany

[13] MetaEdit+ home page: www.metacase.com

[14] Thorsten Fischer, Jorg Niere, Lars Turunski, and Albert Zündorf: Story Diagrams: A New Graph Grammar Language Based on the Unified Modelling Language and Java. Lecture Notes in Computer Science, Springer, 2000

[15] G. Rozenberg (ed.), Handbook on Graph Grammars and Computing by GraphTransformation: Foundations, Vol. 1 World Scientific, Singapore, 1997

[16] R. Ferenc, A. Beszedes, M. Tarkiainen, T. Gyimothy: Columbus – Reverse Engineering Tool and Schema for C++, IEEE International Conference on Software Maintenance, 172-181, Montreal, Canada, 2002

[17] E. Korshunova, M. Petkovic, M. G. J. van den Brand, M. R. Mousavi: CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity

Diagrams from C++ Source Code (Tool Paper), Working Conference on Reverse Engineering (WCRE'06), Benevento, Italy, 2006

[18] Rigi home page: http://rigi.uvic.ca

[19] SHriMP home page: http://www.thechiselgroup.org/shrimp

[20] Bookshelf home page: http://www.swag.uwaterloo.ca/pbs/intro.html

[21] J. Ebert, B. Kullbach, V. Riediger, A. Winter: GUPRO – Generic Understanding of Programs An Overview, Electronic Notes in Theoretical Computer Science 72 No. 2 (2002)

[22] GXL Graph Exchange Library home page: http://www.gupro.de/GXL/

[23] JavaCC home page: https://javacc.dev.java.net

[24] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[25] Paakki, J., Karhinen, A., Gustafsson, J., Nenonen, L., Verkamo, A. I.: Software Metrics by Architectural Pattern Mining, In Proc. International Conference on Software: Theory and Practice (16th IFIP World ComputerCongress). Beijing, China, 2000, 325-332

[26] W. J. Brown, R. C. Malveau, H. W. McCormick III, T. J. Mowbray: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, New York, John Wiley and Sons, Inc., 1998

[27] D. Beyer, C. Lewerentz: CrocoPat: Efficient pattern analysis in object-oriented programs, In Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC 2003), pp. 294-295, IEEE Computer Society, 2003

[28] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, Narendra Jussien: Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together, 16th IEEE conference on Automated Software Engineering (ASE'01), 2001

[29] R. K. Keller, R. Schauer, S. Robitaille, P. Page: Pattern-based Reverse-Engineering of Design Components, In Proc. ICSE, pp. 226-235, ACM, 1999

[30] Nija Shi, Ronald A. Olsson: Reverse Engineering of Design Patterns from Java Source Code, ase, pp. 123-134, 21st IEEE International Conference on Automated Software Engineering (ASE'06), 2006

[31] CDIF Technical Committee, 1994, CDIF Framework for modeling and extensibility, Electronic Industries Association, EIA/IS-107, January