MODEL TRANSFORMATION USING ADAPTIVE SYSTEMS

G. Howells
Department of Electronics, University of Kent, Canterbury, U.K.

B. Bordbar
School of Computer Science, University of Birmingham, Birmingham, U.K.

D. H. Akehurst
Department of Electronics, University of Kent, Canterbury, U.K.

Keywords: Reverse engineering, Design recovery, Model transformation, Adaptive systems.

Abstract: The key research focus of this paper is the combination of advantages from rule based and adaptive systems to produce a hybrid technique that is better able to handle transformations than either technique in its own right. The target problem for the techniques we are developing of reverse engineering is a significant problem when dealing with legacy systems but has great advantages over the significant costs of maintaining or reengineering the old code. The significant novelty of the system is the application of adaptive systems to the problem, these serve to reduce the complexity and quantities inherent in defining transformations rules for each individual case. Current reverse engineering approaches fail due to the difficulties of writing rules to recognize every possible pattern of code that maps to the higher level model.

1 INTRODUCTION

Since the publication of the seminal paper of Chikosfsky and Cross (Chikosfsky and Cross 1990), which sets the taxonomy of reverse engineering, there has been very little change in the objectives of design recovery. These objectives, which can be classified into the following three main categories, are still amongst the most formidable challenges of software engineering:

- analysis of the system,
- synthesizing higher level abstraction to allow better understanding, and
- capability to reuse the legacy system.

Currently, software engineering is going through fundamental changes. Unlike more established engineering disciplines, software engineers tend to overlook the role of models. In recent years, following the birth of model based approaches (MDA 2005, Stahl and Volter 2006), models are being promoted to first class citizens of the software world. Modern software models are expressed in well-established and highly accepted standard languages such as the UML. Such models are not only machine-readable, but also can be handled by different tools (XMI 2005). Moreover, prevailing model based approaches such as the MDA are supported by a large number of available commercial and academic model transformation frameworks, for a list see (Planet MDE 2005).

However, there is a false sense of security that advances of model based approaches will effortlessly result in solving the problems of reverse engineering. It is very naïve to assume that reverse engineering is yet another model transformation which transforms the code back into a UML model. Akehurst et al (Akehurst 2007) highlight the challenges of implementation of UML models in Java. The correspondence between high level abstraction and legacy code is often *one-to-many*, i.e. the same abstract design may be created from different snippets of legacy code. To revisit the above example, in reverse engineering from Java to a UML class diagram, multiple different snippets of Java code implemented using various objects from the *Collection* API, are, by-and-large, reverse-engineered to the same sub-model comprising UML classes connected via associations. Hence, a diverse set of code snippets is reverse-engineered to a single

Howells G., Bordbar B. and Akehurst D. (2009).
MODEL TRANSFORMATION USING ADAPTIVE SYSTEMS.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 133-138
DOI: 10.5220/0002242601330138

pattern of design in the UML. Considering the complexity and high number of variants of such snippets of code, and their combination which may add even further complexity, it is not possible to recognise every possible pattern of code and define suitable transformation rules. As a result, current UML reverse engineering tools are often too simplistic and provide poor high-level representation, which is neither adequate nor precise. It is widely accepted that this is the single most important reason for the lack of en-masse adoption of such techniques by the industry.

Nevertheless, the need for practical reverse engineering systems is significant. By early 1990, the need for reverse engineering and design recovery was already acute. In recent years, the application of Computer Aided Software Engineering (CASE) techniques to reverse engineering has received considerable attention. In particular, mass adoption of standards, such as the UML, and advances in CASE tool technology, such as model driven approaches, have provided new opportunities to address challenges of design recovery in legacy systems. The mainstream idea is to recover a design captured in a legacy system using UML based languages. This will allow an understanding of the core idea of the design by recreating a design abstraction of the legacy system using a UML CASE tool with minimal human intervention. However, in practice, existing tools and methods are not capable of conducting this process accurately. Conceptually, the problem is that the process of reverse engineering transcends a legacy piece of code to a higher level of abstraction. In recent years, model transformation techniques have evolved to such an advance stage that it is both timely and advantageous to address this significant problem.

Our proposal to address the Reverse Engineering problem, is to consider analogies with situations which are successfully addressed by adaptive techniques. In order to allow such systems to react to previously unencountered situations, it is typically necessary to present the systems with pre-defined typical examples characterising the nature of generalised categories or classes which they are required to recognise. The system may subsequently be employed to allow it to deduce abstract properties which may efficiently categorise previously unseen examples. This research thus aims to integrate the advantages of adaptive techniques with existing rule-based technology in order to achieve major productivity enhancements to the current Reverse Engineering environments. Significantly, adaptive systems offer the considerable advantage of being able to generalise from a reduced set of examples and hence potentially may be employed to alleviate the requirement for an infeasibly large set of rules to be defined which would otherwise be needed in non-trivial Reverse Engineering problems. The possibility of employing such systems to address reverse engineering issues is explored in this paper.

The paper is organized as follow. Section 2 highlights the princliples of Model Driven Development while Section 3 introduces the challenges of reverse engineering of legacy systems into the UML languages. Section 4 discusses adaptive approaches and presents a sketch of our method which applies adaptive techniques to model transformation, the details of which are introduced in Section 5. Finally, Section 6 concludes the paper by considering the potential offered by the techniques presented.

2 UML AND MODEL DRIVEN DEVELOPMENT

Model Driven Development (Stahl and Volter 2006) aims to promote the role of *modeling* in software engineering. Model Driven Architecture (MDA) is a flavor of MDD which is initiated by the Object Management Group (MDA 05). MDA relies on standards such as Meta Object Facility (MOF) (www.omg.com) for describing *metamodels*. Metamodels are high-level models from which models of the system are instantiated. MOF can be compared to EBNF, which is used for defining programming languages grammar. As a result, MOF is a blueprint from which *MOF Compliant metamodels* are created.

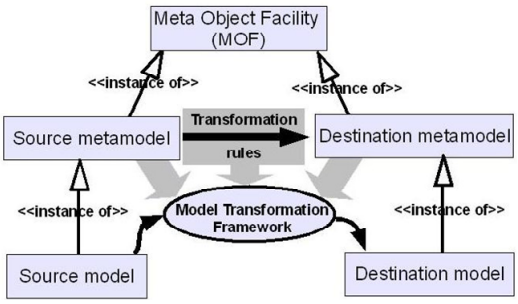

Figure 1: An overview of MDA.

Figure. 1 depicts an outline of MDA and the process of *Model Transformation.* A number of *Transformation Rules* are used to define how various elements of one metamodel (*Source metamodel*) are mapped into the elements of another metamodel (*Destination metamodel*). The process of Model Transformation is carried out automatically via

software tools which are commonly referred to as *Model Transformation Frameworks*. A typical Model Transformation Framework requires three inputs: Source metamodel, Destination metamodel and Transformation Rules. For any instance of the Source metamodel, the Transformation Rules are executed to create an instance of the Destination metamodel.

3 REVERSE ENGINEERING TO UML LANGUAGES

Reverse engineering and design recovery in essence involves bridging the gap between two modeling paradigms. In particular, to reverse-engineer a piece of legacy code to UML languages, the concepts of the language must be described as corresponding UML representations. However, it is not always possible to identify the corresponding elements and present the correct rule for transformation. For example, (Akehurst 2006) describes the challenges of describing UML association in Java, as Java directly does not support the idea of associations. Reverse-engineering of associations has been addressed by various approaches. Barowski and Cross (Barowski and Cross 2002) propose a method investing Java classes to identify the associations and dependencies between the classes. However, the approach fails to identify detailed information such as multiplicities. Other approaches for identifying association multiplicity, aggregation and composition is presented in (Gueheneuc and Albin-Amiot, 2004), (Gogolla and Kollman 2000). Sutton and Malenic (Sutton and Malenic 2005) describe shortcomings of existing UML tools for reverse engineering and present a method of design recovery from C++. The situation is even more complex in case of the languages that are not Object Oriented, such as COBOL, which also requires dealing with issue related to style of coding, such as the use of GOTO (Zhang et al 2005).

4 AN ADAPTIVE APPROACH TO MODEL TRANSFORMATION

Adaptive systems, often typified by Artificial Neural Architectures or Genetic Algorithm based systems, offer the ability to learn and generalize from a set of known examples allowing them to recognize previously unseen inputs based on their similarity of characteristics with previously seen examples. Although there are numerous variations of adaptive systems (Haykin 1999, Michalewicz 1996), in essence they operate by searching a large, potentially multi-dimensional search space looking for optimal solutions for a problem. Subsequent examples presented to the system are then classified according to their similarity with previous examples. This novel proposal combines existing techniques in rule based and adaptive systems to produce a hybrid capable of addressing problems to which neither is individually suited.

As has been stated, there is a false sense of security that advances of model based approaches will inevitably result in solving the problems of reverse engineering. Moreover, large programs are created from scattered code in different parts (*delocalised plans* (Letovsky and Soloway 1986)), exiting methods of program plan recognition based using run-time information (Bojic and Velasevic 2000), slicing techniques (Walkinshaw 2005) and static approaches (Tonella 2003) fail to cope with sheer number of involving variations. To do an accurate and correct transformation, it is crucial to identify all such variations of code.

A fundamental concept in order to identify such variants is the concept of the *closeness* between two snippets of code or more generally between two abstract models. Essentially, in order to evaluate a candidate solution to a problem, adaptive systems require an objective measure as to how close a candidate solution lies to an ideal solution supplied as part of a training example. It may be worth noting at this point that all adaptive systems in our method employ supervised learning strategies where a number of training examples with known optimal solutions is present. Further, a mechanism for adapting such candidate solutions such that they form an even closer approximation is required. In conventional weighted artificial neural architectures such as the Multi-Layer perceptron, closeness is represented as a real number. Further, the weights themselves are represented as real numbers and they are adapted either upwards or downwards via a learning rule such that the closeness (or error) value is reduced. By repeated exposure to training examples, the network adapts so as to allow a good approximation to the desired results for the given training patterns. The expectation is that, having been exposed to the training patterns, subsequent unseen patterns will be correctly classified (or transformed) due to their implicit similarity to the training patterns. A fundamental goal of our system is thus to identify a suitable measure of closeness between independent code segments as described below.

Artificial neural networks have been researched extensively and offer a technology capable of learning from its environment using relatively simple distributed processing elements (neurons). The technology depends on the employment of *learning* or *training* algorithms capable of modifying the weights associated with neural connections to take into account properties associated with a known sample presented to the system (other neural models are available but this description represents the most common approach.) It is critically these weight values, which represent the knowledge present within the system. This concept is fundamental in capturing the inherently concealed dependencies, such as subtle multiple distinct instances of a particular programming concept which are physically disjointly located. Alternative approaches such as (Bojic and Velasevic 2000), (Walkinshaw 2005) or (Tonella 2003) do not possess such capabilities, which is thus a significant novelty of this approach.

For our purposes, the structure of such a network will be generalised from that described above to allow the weights and learning functions within the network to model efficiently the component concepts of the models under investigation. They will be represented as multi-dimensional components where the constituent model components represent fundamental model concepts from the source model of the transformation. The system employed for our network is based on Constructive Type Theory (CTT) (Howells and Sirlantzis 2008, Sirlantzis et al 1999, Thompson 1991). This generalisation of the structure of the neural architecture is a fundamental novel component of the proposal and underlies its ability to address the previously intractable problem of reverse engineering.

Constructive Type Theory is a formal logic based on the application of Constructive mathematics. Constructive mathematics differs from classical mathematics in that all mathematical proofs produced must be based on a demonstration of how to construct an example of the theorem or proposition being asserted. In other words, proof by contradiction is not allowed. As a result of this, many paradoxes of Classical mathematics such as Russell's paradox are eliminated. A further consequence of this approach is that a proof in Constructive Type Theory is itself an algorithm indicating how to construct an example of the proposition being asserted. That proposition itself may be considered a *datatype definition* or at a higher level a *formal specification* of the algorithm forming the proof. Constructive Type Theory thus represents a merging of the worlds of formal mathematics and software engineering and its potential for the production of guaranteed bug-free, provably correct software is enormous.

5 HYBRID TRANSFORMER

A key advantage allowing the proposed system to be practically applicable is the combination of advantages from rule based and adaptive systems to produce a hybrid technique that is better able to handle transformations than either technique in its own right. The target problem for the techniques we are developing of reverse engineering is a significant problem when dealing with legacy systems but has great advantages over the significant costs of maintaining or reengineering the old code. The significant novelty of the proposal is the application of generalised adaptive systems to the problem, which serves to reduce the complexity and quantities inherent in defining transformations rules for each individual case. Current reverse engineering approaches fail due to the difficulties of writing rules to recognise every possible pattern of code that maps to the higher level model whereas our method significantly reduces the complexity by employing adaptive systems to model small variations in source model components.

The two strands of the proposal, *MDD rule based transformations* and *adaptive systems*, complement each other as the rules constrain the adaptive system allowing it to converge in a computationally feasible manner and the adaptive system serves to reduce the quantity of rules required to identify all possible code patterns

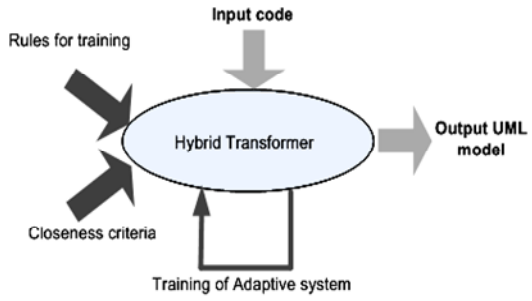

Figure 2: The underlying Paradigm.

The underlying paradigm is exemplified by Figure 2. A Hybrid Transformer is constructed which has at its base a set of rules; on top of which the adaptive system is trained to recognise patterns of code that are 'close' to the patterns described in the rules. After sufficient training the Hybrid Transformer will

receive the Input Code to produce the Output UML model which reverse engineers the Input code.

The notion of *closeness* of two models is fundamental to the operation of an adaptive system in this context; however we cannot assign a simple total ordering on all models due their diversity. We rather incorporate a multi-dimensional notion of closeness allowing freedoms of movement under the governance of transformation constraints. The two components of the hybrid system are now introduced. In the next two subsections, the main two strand of the approach rule based training and adaptive system will be explained.

5.1 The Rule based Training

Reverse engineering and design recovery in essence involves bridging the gap between two modelling paradigms. For example, to reverse-engineer C++ to the UML, the concepts of the C++ language must be described as corresponding UML representations. Currently, UML tools, as described in (Sutton et al 2005) for C++, often deal with the problem superficially, resulting in inaccurate design recovery. Such poor design recovery is due to absence of a "bridge between C++ and the UML". We wish to emphasise that the problem is not exclusive to C++. In case of Java, (Akehurst et al 2007), highlight shortcomings of existing tools in code generation, which are also not considered in reverse engineering. In our opinion one major cause of such shortcomings is the diverse style of coding adopted by programmers. For example consider the Class diagram shown in Figure 3.

UML tools and developers can implement this diagram in a number of different ways in Java. For example for the association which relates the two class can be implemented as an `ArrayList`, `LinkedList`, conventional `Array`, a `Set`, … among others. This by-and-large depends on the style of coding adopted by the developer of the tool vendor. Using Rules, we capture such styles. For example, one rule can be about reverse engineering of a associations which are modelled as `ArrayLists` and the other as `LinkedLists`. The adaptive systems will pick up most suitable rules after sufficient learning.

5.2 The Adaptive Sub-system

The rule based system is complemented by an adaptive system which initially maps the lowest level program constructs into more general, and hence smaller number of, categories where a feasible set of rules may be defined.

Figure 3: A simple class diagram.

The system utilises the multi-dimensional Constructive Type Theory (CTT) logic based adaptive artificial neural network which has been developed by the authors. Each processing element within the network is associated with a Judgement playing the role of an activation function within conventional networks. The Judgement is a higher order implication type where the component domain types mirror the range types of the weights associated with each input to the processing element. For example, a processing element with two inputs would have a Judgement of type $\mathbf{A \rightarrow B \rightarrow C}$ where the weights associated with the two inputs would be of type $\mathbf{X \rightarrow A}$ and $\mathbf{Y \rightarrow B}$ respectively. The output of the processing element will be of type **C**. Note that no restriction is placed on the types **A,B,C,X and Y** so as to allow them to model the components of the languages under investigation**.**.

The connections between the processing elements have weights associated with them. It is proposed that the weight values themselves will be *Judgements* where a Judgement is defined to be a logical proposition together with its proof. We will identify the fundamental concepts within a model and define sub-orderings for each concept. For example, a simple product type such as an n-tuple maybe sub-divided into its component types and a closeness function defined as the sum of the closeness values of the component types.

Our primary approach to the closeness problem is a structural one in that the closeness value of a complex structure is defined to be a function (a simplistic example is the sum) of the closeness values for the components of the structure. Significantly, the closeness value itself may be a multi-dimensional value, typically an array of real valued numbers. For a number of primitive types, an absolute value of closeness will be defined. Care must be taken here to normalise the closeness values for the primitive structures to ensure than any given closeness value (e.g. the number 4) always carries the same semantic weight and contributes equally to each component of the closeness evaluation.

Two independent components of the system require measures of closeness:-

1. Primarily, a closeness value is required for the candidate outputs of the adaptive system so that the output may be compared to the training outputs for a given training pattern. For our examples, the outputs consist of UML diagrams and hence the closeness values will

be a multi-dimensional vector based on the structural components of UML

2. A closeness value will also be required for the CTT component values which reside within the adaptive system so that the network is able to adapt in a logical fashion. Each Judgement will be assigned a closeness value and during adaptation mapped to a similar Judgement as required by the error value of the network represented as the closeness values between the actual and required UML diagrams resulting from the initial network.

This methodology is feasible due to the restriction on the adaptive system inherent in the multidimensional closeness measurement.

6 CONCLUSIONS

A hybrid system for addressing the significant practical problems of Model Transformation for difficult problem domains such as Reverse Engineering is proposed which combines the advantages of rule-based and adaptive techniques for Model Transformation in such a way that the advantages of both techniques are retained whilst alleviating the disadvantages inherent within both techniques.

ACKNOWLEDGEMENTS

This research was supported at the Department of Electronics at University of Kent though the European Union ERDF Interreg IIIA initiative under the MODEASY grant.

REFERENCES

E. J. Chikofsky and J. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, pp. 13-17, 1990.

MDA, "Model Driven Architecture, Object Management Group, www.omg.org/mda/," 2005.

T. Stahl and M. Volter, *Model Driven Software Development; Technology engineering management*: Wiley, 2006

XMI, "XML Metadata Interchange (XMI), v2.1, available at www.omg.org," 2005.

Planetmde, "Planet MDE, www.planetmde.org," 2005.

D. H. Akehurst, W. G. J. Howells, and K. D. McDonald-Maier, "Implementing Associations: UML2.0 to Java 5," *Journal of Software and Systems Modeling*, March, 2007.

L. A. Barowski and J. H. Cross, " Extraction and Use of Class Dependency Information in Java," presented at Ninth Working Conference on Reverse Engineering (WCRE'02), 2002.

Y.-G. Gueheneuc and H. Albin-Amiot, "Recovering binary class relationships: putting icing on the UML cake," presented at 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2004.

M. Gogolla and R. Kollman, "Re-Documentation of Java with UML Class Diagrams," presented at 7th Reengineering Forum, , 2000.

A. Sutton and J. I. Maletic, "Mappings for Accurately Reverse Engineering UML Class Models from C++," presented at 12th Working Conference on Reverse Engineering (WCRE 2005), 2005.

J. Pu, Z. Zhang, Y. Xu, and H. Yang, "Reusing legacy COBOL code with UML collaboration diagrams via a Wide Spectrum Language," presented at IEEE International Conference on Information Reuse and Integration, IRI, 2005.

S.Haykin, "Neural Networks, A Comprehensive Foundation" Prentice Hall 1999.

Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs. 3rd ed". Springer-Verlag, Berlin Heidelberg New York (1996)

G. Howells, K.Sirlantzis Improving Robotic System Robustness via a Generalised Formal Artificial Neural System. In symposium on Learning and Adaptive Behaviour in Robotic Systems (LAB-RS 2008)

S. Letovsky and E. Soloway, "Delocalized Plans and Program Comprehension," *IEEE Software*, vol. 3, pp. 41-49, 1986.

D. Bojic and D. Velasevic, "Reverse engineering of use case realisations in UML," presented at ACM Symposium on Applied Computing (SAC'00), 2000

N. Walkinshaw, M. Roper, and M. Wood, "Understanding Object-Oriented Source Code from the Behavioral Perspective," presented at 13th IEEE International Workshop on Program Comprehension (IWPC'05), 2005

P. Tonella and A. Potrich, "Reverse engineering of the interaction diagrams from C++ code," presented at International Conference on Software Maintenance (ICSM'03).

K. Sirlantzis, G. Howells, and S. Paschalakis, "A functional neural network prototype for multidimensional data analysis," *Image Processing and Its Applications, 1999. Seventh International Conference on (Conf. Publ. No. 465)*, vol. 1, 1999.

S. Thompson, *Type theory and functional programming*: Addison-Wesley Wokingham, England, 1991.

A. F. R. Rahman, W. G. J. Howells, and M. C. Fairhurst, "A multiexpert framework for character recognition: a novel application of Clifford networks," *Neural Networks, IEEE Transactions on*, vol. 12, 2001.

A. Sutton and J. I. Maletic, "Mappings for Accurately Reverse Engineering UML Class Models from C++," presented at 12th Working Conference on Reverse Engineering (WCRE 2005), 2005.

L. Zuck, A. Pnueli, Y. Fang, and B. Goldberg, "VOC: A methodology for the translation validation for optimizing compilers," *Journal of Universal Computer Science*, vol. 9, pp. 223-247, 2003.