Context-Free Slicing of UML Class Models

Huzefa Kagdi, Jonathan I. Maletic, Andrew Sutton
Department of Computer Science
Kent State University
Kent Ohio 44242
{hkagdi, jmaletic, asutton}@cs.kent.edu

Abstract

The concept of model slicing is introduced as a means to support maintenance through the understanding, querying, and analysis of large UML models. The specific models being examined are class models as defined in the Unified Modeling Language (UML). Model slicing is analogous to classical program slicing. Since UML class models do not explicitly embody any behavioral aspect by themselves, models slices are computed in a context-free manner. The paper defines and formalizes the concept of context-free model slicing. A concrete application of model slicing in software maintenance is presented to support the usefulness and validity of the method.

1. Introduction

There is an ever increasing importance being put on design models to support the evolution of large software systems. Design models such as UML class models are being maintained and updated from initial development as well as being reverse engineered to more accurately reflect the state of evolving systems. However, herein lays the problem - a UML class model for a large system is typically comprised of thousands of classes and relationships. Viewing the entire model at one time is impractical and typically of little use for a particular maintenance task.

There are few, if any, existing methods (or tools) for supporting the automated or semi-automated extraction of meaningful subsets of a class model. Currently, this is done manually. An engineer must wade through the entire class model and (using some tool) construct a specific class diagram within the context of this model. While a diagram may exist that is close to what is desired (e.g., the entire class hierarchy for a particular concept) it may be far too unwieldy and include many classes of little interest or consequence to the problem at hand (e.g., entity classes contained by all classes in an inheritance hierarchy).

In the work presented here, we introduce a method to automatically generate a subset of a UML class model based on a user-defined criterion. The goal of this work is to allow us to automatically extract a pertinent and meaningful UML class diagram from a very large UML class model.

In the next section (2) we describe and formally define our approach to UML class model slicing. A concrete application of this approach is given in section 3. Related work on examining subsets of UML diagrams are discussed in section 4. Conclusions are given in section 5.

2. UML Model Slicing

Our method is rooted in the classical definition of program slicing but extends that concept to the UML models. In general, we term this approach *model slicing*. Here, we focus our discussion on the UML class models. However, since class models are devoid of explicit behavioral information (by themselves) we further define the concept of *context-free* model slicing. Program slicing has the implicit context of the definition-use relationship with respect to a supplied slicing criterion. In model slicing of a UML class model, we must specify some sort of non-behavioral aspect to construct the slice. In short, model slices are defined via a generalized slicing criterion that is specified with predicates over the model's features.

Program slicing, as defined in [11], takes a program and a slicing criteria to compute a slice or subset of the source code. More formally, given a slicing criterion, $s(v, n)$ with a set of variables v and a location of a statement of interest s, program slicing determines a set of statements contributing directly or indirectly to the values of variables, v, before the statement s is executed. Those resultant statements comprise the program slice. Statements in a program slice have a specific behavioral context. In this case, the behavior is the statements affected by or affecting the states of the variables involved in a computation at a particular point in a program. These statements included in the program slice are those extracted from the investigation of the definition-use relationship (i.e., statements with definition and usage of variables given in the slicing criteria).

It should be noted the definition-use relationship is the only relationship of interest in program slicing. This is evident in the definition of the program-slicing criteria. The slicing criterion does not provide any means for the explicit specification of relationships other than definition-use relationship. The usage of this relationship is implicitly assumed. This assumption restricts program slicing to the singular relationship among the elements (statements) of the source code.

UML model slicing extends this concept of program slicing via a generalized, albeit more complex, slicing criteria. These extensions elevate the capabilities of program slicing from the source-code level to UML structural models (i.e., class models) and behavioral models (i.e., sequence, collaboration, and state behavior models). All elements and relationships defined for the UML class models can be used in the computations of a model slice. This includes elements such as classes, packages, components and operations, and relationships such as associations, dependencies and generalizations. The slicing criteria for the extended domain must account for all the elements and relationships available in the UML metamodel.

Unlike program slicing, model slicing does not necessarily require the physical location of an element of interest (i.e., an observation point of a behavior). UML class models, representing only abstracted structural elements, contain no behavioral elements (e.g., instances of classes or statements). However, other views such as sequence diagrams, object diagrams, and collaborations define contexts in which objects may be explicitly located. In these cases, we define *context* to be the location of the object. The context can be a particular set of scenarios in which a set of objects are involved or a particular range in the lifeline of a set of objects when dealing with an interaction model such as one pictured by a sequence diagram.

Here, we distinguish between slicing of models that require or do not require context information and introduce the terms *context-free* and *context-sensitive* slices. The context-free slices are applicable to models that do not require a context for the computation of a model slice. Context-sensitive slices are applicable to models that do require such context information. Here we focus on the definition of context-free model slicing and reserve the definition of context-sensitive slicing for future work (as it requires the former definition at the very least).

2.1. Context-free Model Slice

A context-free model slice is defined primarily to encapsulate static and structural aspects of a UML model and precludes the inclusion of behavioral, computation, or interaction information. For the purpose of model slicing, we define a model, *M*, as a directed multi-graph $M = (E, R, \Gamma)$ where

- $E = \{e_1, e_2,.., e_n\}$ is the finite set of elements,
- $R = \{r_1, r_2,...,r_m\}$ is the finite set of relationships,
- $\Gamma{:}R \Rightarrow E \times E$ is a function that maps elements to elements via a relationship. $\Gamma{:}R \Rightarrow E \times E$ defines multiple relations between each element. The relations r_i and r_j are multiple relations on the same elements if $\Gamma(r_i) = \Gamma(r_j)$.

Each *element, e is defined by a finite set of properties* $_i$ $\{p_1, p_2 \ldots p_k\}$, such that each element has a finite set of properties. Likewise each relation, *r is defined by a finite set of properties* $\{p_1, p_2, \ldots p_k\}$. Each property, p_i is an ordered pair that defines a name and a value (e.g., {(*type*, *Class*), (*name*, "*stack*")…}).

The model consists of a finite set of elements *E* and a finite set of relationships *R*. The set *E* contains instances of all metamodel elements such as class, namespace, package, component etc., and the set *R* provides all instances of relationships including association, generalization, dependency, etc. These correspond to the meta-classes defined in the UML metamodel. As can be seen from the above definition elements and relationships are both first class entities.

The elements and relations are mapped with the function, Γ. Given a relationship r_i, the mapping Γ tells which elements are its end points.

A property of a member of the set *E* could be the type of element such as (*type*, *Class*), and a property of a member of the set *R* could be the type of the relationship such as (*type*, *Generalization*). Thus, this definition of model makes the elements and relations along with their property sets available for model slicing

The *context-free model slice*, S_{cf}, of a given UML model *M*, is defined as a function over a model and determined by the specified slicing criteria, C_{cf},

$$S_{cf}(M, C_{cf}) = M' = (E', R', \Gamma') \subseteq M$$

The *context-free slicing criteria* C_{cf} is defined as a triple of constrains that must all be satisfied to construct *M'*.

$$C_{cf} = (I, S, D)$$

The initial-element set, $I = \forall e \in E \mid P_I(M)$ specifies the initial elements of the slice. The predicate $P_I(M)$ is constructed to be satisfied for elements in the initial set.

The selected-element set, $S = \forall e \in E \mid P_S(M)$ specifies the elements selected for inclusion in the resultant slice. The predicate $P_S(M)$ is defined so that only elements of interest are selected.

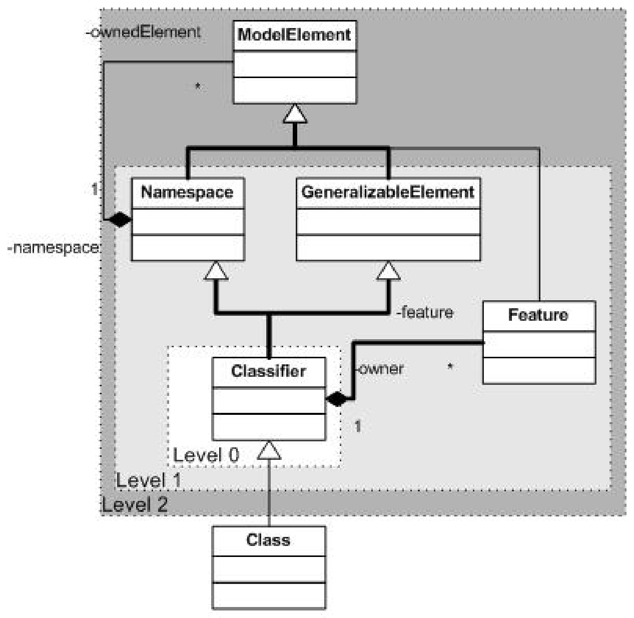

Figure 1. The model slice is shown within the grey boxes. The levels reflect the traversal of two relationships. The class Class is not included in the slice.

The dimension-set, $D = \forall r \in R \mid (P_D(M) \wedge T(M) \wedge B(M))$ specifies the relationships of interest (a.k.a., dimensions) to be included in the slice and traversed in its computation. The predicate P_D defines which relationships are included in the slice. The predicate *T(M)* defines a terminating condition of the computation with respect to each of, or all, the relationships. The bounding predicate *B(M)* is the computational upper bound on the path length between elements with respect to each of, or all the relationships, of the slice.

3. Applying Model Slicing

In this section, we demonstrate by an example the ability of model-slicing to satisfy a typical design understanding question that is asked of UML class models during software maintenance.

Question: *How can a programmer discover relationships between a specific class and other classes in a UML system model?*

Programmers, when faced with such problems, might typically browse through project software artifacts including reference manuals, UML class diagrams, and source code to discover relationships between one class and its associates (base classes, aggregations, dependencies, etc.). Rarely, even in good software documentation, is this information localized for easy consumption. Model slicing can be used as a query mechanism to provide concise views of the programmer's informational needs. Consider a snippet of the UML metamodel shown in *Figure 1*. A programmer looking for the inheritance hierarchy and immediate associations of the *Classifier* class could use the following slicing criteria to determine related classes.

$P_I(M) := e \supseteq \{(name, \text{" Classifier"})\}$
$P_S(M) := T$
$P_D(M) := r \supseteq [R1 \vee R2]$
$R1 = \{(type, \text{"Generalization"})\}$
$R2 = \{(type, \text{"Association"})\}$
$T(M) := F$
$B(M) := (\forall r \supseteq R1) \vee$
$\quad (\forall r \supseteq R2 \wedge (\forall e\ \exists e' \in E\ |path(e, e', r)| \leq 1))$

The slice consists of elements in the initial set, shown by the level zero in *Figure 1*. The traversal starts with *Classifier* and considers elements at level one. The relationships traversed to the candidate elements are shown in bold. Two elements along the *Generalization* path (*Namespace* and *GeneralizableElement*) and one element along the *Association* path (*Feature*) are included in the slice. After the first iteration, the slice now includes those elements contained in level zero and those contained in level one. In the second iteration, only elements involved in the *Generalization* relationship are considered because the bounding condition of the *Association* relationship was satisfied at level one. All the elements in the *Generalization* relationship at this level are included in the slice and are shown to be contained in level 2. The computation of the model slice terminates after this iteration as there are no more elements in the *Generalization* path. The final slice is depicted in *Figure 1*, consisting of all elements contained within the outermost level.

The class, *Feature*, is included as part of *Association* because aggregation is a kind of association in the UML metamodel. Note the *Association* computation is not transitive as only the immediate associations of *Classifier* are considered (i.e., *Feature*). Associations of base classes are not considered for the slice. If those associations are of interest, the base classes must be included in the initial element set of the slicing criteria.

4. Related Work

In this section, we discuss existing methods to query and extract information from UML class models – the subset of a model that deals with the structural design of the system (i.e., classes, attributes, and operations). These approaches include constraint and query languages, XML processing, and model processing.

The UML object constraint language (OCL) [1, 3, 10] allows querying of UML models. It is primarily used for model validation and constraint checking. Recent work in Model Driven Architecture (MDA) has proposed using OCL as a query language to satisfy the query component to the QVT (Query/View/Transformation) specifications [2]. As a query language, OCL can be used to extract model elements that satisfy some condition (e.g., all abstract classes in a model).

Many approaches to querying UML models involve analysis or operations on the XML-based interchange format for UML models, XMI [5, 7, 9]. These approaches apply XML processing techniques such as XPath, XQuery and XSLT to extract data from the XMI files. This approach is similar to those used in XML-based source code or AST analysis.

Other approaches involve model analysis in the context of additional semantic information. In [4] a metric-based approach is proposed to derive subparts of UML class diagrams. A high level view of the subparts exhibiting a particular metric-based feature such as coupling (referred to as a coupling diagram) is obtained, and then classes within a particular range of metric values are extracted and visualized via a diagram. This approach is particularly good at "pruning" large UML diagrams to show only the relevant classes in the diagram. However, this approach is very limited in the types of pruning that can be done.

Other approaches involve the use of additional languages and technologies to query or validate UML models. In [6], OCL expressions are translated to SQL statements to query and evaluate models stored in a relational database. In [8], Python and OCL are used together to provide more procedural control for the evaluation of such queries.

5. Conclusions & Future Work

In this paper we introduced the concept of model slicing pertaining to UML class models. The work generalizes the concept of program slicing so that it can be applied to more abstract models. The ultimate goal is to provide an automatic mechanism that will enable developers to extract task-specific UML sub-models by giving specification in terms of UML-level constructs. This type of approach will support the development of sophisticated tools to automatically extract meaningful sub-models of large system design model so they can be visualized or analyzed to facilitate maintenance and evolution tasks.

Model slicing is realized as a set of predicates that specify a slicing criterion. We envision that languages such as OCL can be used to implement the predicates required to compute the model slices. In this vein, we are extending our prototype implementation of the model slicing algorithm and plan to apply it to large models of real systems (e.g., an open source systems such as HippoDraw).

We are actively investigating the concept of context-sensitive slicing of UML models using a behavioral model, such as a sequence diagram that is implicitly linked to the static class model.

This research was supported, in part, by a grant from the National Science Foundation (CCR-02-04175).

6. References

[1] Akehurst, D. H. and Bordbar, B., "On Querying UML Data Models with OCL", in Proceedings of Fourth International Conference on the Unified Modeling Languages (UML'01), Toronto, Canada, October 1-5 2001, pp. 91-103.

[2] Appukkutan, B., Tratt, L., Clark, T., Reddy, S., Venkatesh, R., Evans, A., Maskeri, G., Sammut, P., and Willans, J., "QVT-Partners Revised Submission to MOF 2.0 Query/View/Transformations RFP", Object Management Group, Document ad/03-08-08, August 2003.

[3] Gogolla, M. and Richters, M., "On Constraints and Queries in UML", in Proceedings of Workshop on the Unified Modeling Language - Technical Aspects and Applications, Mannheim, Germany, November 10-11 1997, pp. 109-121.

[4] Kollmann, R. and Gogolla, M., "Metric-Based Selective Representation of UML Diagrams", in Proceedings of Sixth European Conference on Software Maintenance and Reengineering(CSMR'02), Budapest, Hungary, March 11 - 13 2002, pp. 89-98.

[5] Kurtev, I. and van der Berg, K., "Model Driven Architecture Based XML Processing", in Proceedings of ACM Symposium on Document Engineering (DOCENG'03), Grenoble, France, 2003, pp. 246-248.

[6] Marder, U., Ritther, N., and Steiert, H.-P., "A DBMS-based Approach for Automatic Checking of OCL Constraints", in Proceedings of OOPSLA'99 Workshop on Rigourous Modeling and Analysis with the UML: Challenges and Limitations, Denver, Colorado, November 1-5 1999.

[7] Peltier, M., Bézivin, J., and Guillaume, G., "MTRANS: A general framework, based on XSLT, for model transformations", in Proceedings of ETAPS'01 Workshop on Transformations in UML, Genova, Italy, April 7 2001.

[8] Siikarla, M., Peltonen, J., and Selonen, P., "Combining OCL and Programming Languages for UML Model Processing", in Proceedings of UML'03 Workshop on OCL 2.0 - Industry Standard or Scientific Playground?, San Francisco, California, October 21 2003.

[9] Stevens, P., "Small-Scale XMI Programming: A Revolution in UML Tool Use?" Automated Software Engineering, vol. 10, no. 1, January 2003 2003, pp. 7-21.

[10] Warmer, J. and Kleppe, A., The Object Constraint Language : Precise Modeling with UML, 1st ed., Addison-Wesley Pub Co, 1998.

[11] Weiser, M., "Program slicing", in Proceedings of International Conference on Software Engineering (ICSE'81), San Diego, California, United States, March 09 - 12 1981, pp. 439 - 449.