The Certification of Software Tools with respect to Software Standards

Panuchart Bunyakiati, Anthony Finkelstein and David Rosenblum
Dept. of Computer Science, University College London
London W1CE 6BT
United Kingdom
{p.bunyakiati, a.finkelstein, d.rosenblum}@cs.ucl.ac.uk

Abstract

Software development standards such as the UML provide complex modeling languages for specifying, visualizing, constructing, and documenting the artifacts of software systems [1]. Software tools support the production of these artifacts according to the model elements, relationships, well-formedness rules and semantics defined in the standards. Due to the complexities of both standards and software tools, it is difficult to establish the compliance of the software tools to the standards. It has been suggested that many existing tools that advertise standard compliance fail to lift up to their claims. The objective of this work is to propose a framework for developing systematic, disciplined, and quantifiable certification schemes to assess the compliance of these tools to standards and to diagnose the causes of non-compliance.

1. Introduction

The importance of standard compliance is well understood in the software engineering community as software systems must operate across platforms, environments, or physical machines and interoperate with other heterogeneous software systems. Software certification has value because it assures that the software systems will operate correctly as specified in the standards. Hence, there exist many certification services for software products. The Open Group, for example, has developed conformance test suites and provides certification services for software systems including UNIX, Linux, CORBA, LDAP, and WAP. The ADA Conformity Assessment Authority (ACAA) maintains the ADA conformity assessment test suite for ADA compilers. As for software tools, the only software tool certification program is the SSADM Tools Conformance Scheme [2] developed in 1993 by the central computer and telecommunication agency (CCTA). The scheme measures the extent to which software tools support the creation of software artifacts that conform to the SSADM standard. Today, there are much more complex software development standards imposing complicated structures and much more precise constraints than those of the SSADM. We argue that an approach that properly tackles the problem of certification with respect to complex software development standards is still missing. By systematically establishing this framework for developing software tool certification schemes, we will be able to control the quality of the certification, assure the correctness and completeness of the assessment over the usual ad hoc method.

2. Related Works

Specification-based testing is the practical approach to assess standard compliance. The compliance test suite generation can be considered as a branch of constraint satisfaction problem (CSP) in which the first-order predicate is given and processed to find models that satisfy it. For testing linked data structures, TestEra [3] uses the Alloy Analyzer [4] to search for the structures constructed from the specification that satisfy the formula. Korat [5] uses method preconditions written in Java Modeling Language (JML) to automatically generate all non-isomorphic test cases up to a bound on a given input size. As there are many possible ways for the models to violate the constraint specifications, a test suite must cover all of these possibilities to be considered as an adequate standardized test suite. Instead of starting from the formula and trying to find the models that satisfy/dissatisfy the formula, we start from the construction of all possible models, up to the input size guaranteed by the certificate, and classify these models into demonstrations and counterexamples using Binary Decision Diagram (BDD). BDD is a proven technology widely used in digital design, computer-aided verification of circuits, and automatic test pattern generation [6]. We use Crocopat [7], an implementation based on BDD, to manipulate and classify these test input models.

3. Software Tool Certification Strategic Cube

Software tools are complex software systems that can not be tested as monolithic structures. Software tools have various functions and each function has a specified standard specification to adhere to. For example, the user interface is built according to the concrete syntax and notation of a modeling language, while the repository is constructed as specified in the abstract syntax. These tools therefore require systematic testing strategies to ensure their compliance to the various aspects of the standard.

For this, we introduce a holistic view of software tool certification as a strategic cube shown in figure 1. Amongst the important factors that must be considered in the certification of software tools are 1) the functional architecture of the software tools, 2) the approaches to certify the quality of software [8] including personnel accreditation, process certification and product quality assessment, and 3) perspective of the participants involving in the certification scheme including the tool vendors (first party), the tool users (second party) and the certification authorities (third party). The three-dimensional cube reveals 63 certification schemes.

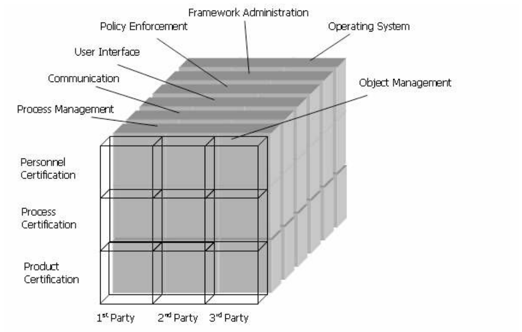


Figure 1 Tool Certification Strategic Cube

In this paper, we focus on the product certification of the object management tools (the repositories – where models are stored, retrieved and maintained) by third-party certification authorities. Our test case generation technology would allow these third-party organizations, not necessary be the standard bodies themselves, to automatically generate the compliance test suites from the software standards. This availability of the test suites makes it possible to perform product certification directly, regardless of who has developed the product and how they have done it.

4. Unified Modeling Language (UML), Object Constraint Language (OCL) and Software Modeling Tools

One of the most important modeling standards today is the Unified Modeling Language (UML). The UML standard is very large and complex visual modeling language specifications. The new version of the UML 2.0 specification consists of six separate large documents.

Being a visual language, the UML is convenient for communicating design concepts and ideas among a small group of developers at a whiteboard [9]. For this purpose, only a small subset of UML is, in fact, needed. The whole set of the UML 2.1.1 specifications [10], including the UML infrastructure and superstructure, XML metadata interchange (XMI), Diagram Interchange and OCL, is intended for supporting the implementation of software tools and promoting the interoperability among the tools, particularly, among modeling tools and Model Driven Architecture (MDA) tools for model transformation and code generation.

To establish tool interoperability and to allow the transformation of models, software tools have to rigorously comply with the UML standards. In essence, the UML standard is the description of the ways that software models can be constructed in UML. The UML metamodel defines the abstract syntax while the well-formedness rules, written in OCL, formulate the static semantics. Putting them together allows the models built according to the UML specification to be verified and checked for their consistencies. The correctness and consistency of UML models with the UML standard are prerequisite for interchanging models from one tool to another tool, and for transforming these models from one language to another language in an automated, tool-supported development environment.

4.1 The UML metamodel

UML is defined with the metamodeling approach using the Meta object facility (MOF) [11]. At the very least, a metamodel described by Essential MOF (EMOF) may consist of primitive data types, enumeration types, packages, model elements or classes, and structural relationships. The structural relationships are specified by the properties (that are typed elements with multiplicities) of the classes. Furthermore, a class may have operations that take parameters and may have super classes. This can be described as:

Definition 1 A metamodel M = (A, E, C, R) consists of a finite set A of data types Integer, Real, Boolean, and String, a finite set E of enumeration types, a finite set C of classes and a finite set R of a legal relationships that allow 1) the connections from an instance of c in C to its attribute which may be an a in A or an e in E and 2) the connections from the instance c to c' in C.

A model can be represented as an attributed graph [12] in which the data types in the metamodel become data value nodes, and the classes become instance nodes. The edges of the graph are the links with respect to the structural relationships in the metamodel. Each edge is assigned with a mapping (s, t) where s is a source node (the owner of the attribute) and t is a target node (an attribute value or an owned attribute).

Definition 2 A model m = (V, I, L) consists of a finite set V of data values, a finite set I of instances of the classes and a set L of links which are the instances of the legal relationships. The link l in L connects an instance i in I to a data value v in V or to an instance i_2 in I.

Definition 3 We say that a model m appropriate to M if V is in A, each instance i in I is an instance of class c in C and for each link l in L is an instance of a legal relationship r in R; therefore, when i is an instance of c in C and v is a data value of a in A and l(i, v), there exists r(c, a) in R.

4.2 The OCL well-formedness rules

A well-formedness rule is written in OCL [13]. OCL specifies formal constraints that restrict the construction of UML models. A well-formedness rule is a constraint over the metamodel that specifies the condition that models must satisfy to be consistent with their meta-model. The semantic of this satisfaction condition, adapted from [14], is given below.

Definition 4 A well-formedness rule W over the meta-model M is a relation $R(e_1,\ldots,e_k)$ where e is in A or E. And if W and V are well-formedness rules, then so are ~W, W or V, W and V, FAx W, and EXx W.

Definition 5 The satisfaction of model m to a well-formedness rule W is defined as
if W = ~V then M |= W if ~(M |= V),
if W = V1 or V2 then M |= W if M |= V1 or M |= V2,
if W = V1 and V2 then M |= W if M |= V1 and M |= V2.
For the quantifier FA (for all), M |= FAx W if for any e in A U E, $M_{x=e}$ |= W. And we shall treat EX (there exists) as ~(FAx ~W).

4.3 Compliance tools

Now we are in the position to define the term "software modeling tools" precisely. Most importantly, this definition must reflect the characteristics of the software tools for they must allow syntactic and semantic interoperability. The software tools may 1) support the construction of models by manually creating the models through graphical user interface (GUI), programming, importing the models stored in model interchange format such as the XMI or any other means and 2) provide verification functionality for checking whether a software model satisfies the well-formedness rules.

Definition 6 Given a metamodel M and a set of well-formedness rules Ws, a modeling tool S is a piece of software that supports the construction of any model m that appropriate to M and satisfy Ws.

Definition 7 The satisfaction of model m to the well-formedness rule W can be verified with the verification function V of the tool S. The function V returns TRUE when M |= W and returns FALSE otherwise.

5. Compliance Requirements

Compliance is nothing else but the satisfaction of software implementation to the standard specification. Compliance points are purely a breakdown of the entire specification into smaller facets. As we see below there are primarily four types of compliance points:

5.1 Abstract syntax

Model elements can be best defined as the components or the building blocks used to construct the models. As we know, there are a great number of model elements that may be used to construct a model but, of course, not all of them are required for every model. However, the compliance requirement for each tool is exactly the same in that all the model elements specified in the standard must be supported. We may assess the fulfillment of this compliance requirement by tracing the *one-to-one correspondence* between the model elements in the tool and those in the standard.

Definition 8 Let E_s be a set of metamodel elements specified in a standard and E_t be a set of metamodel elements implemented in a compliance tool. Since the properties of each metamodel element in E_s are known, we can match up every metamodel element in E_s to a unique, identical metamodel element in E_t. We say that there is a one-to-one correspondence between E_s and E_t if every element in E_s has a corresponding element in E_t.

5.2 Well-formedness rules

Each well-formedness rule is a compliance requirement in itself as each rule defines a constraint. To assess the compliance to each well-formedness rule, a set of test input that scrutinizes the behavior of the tools is required. We will further discuss how the test input is used to test the compliance of the modeling tools to the well-formedness rules in section 6.

5.3 Behavioral Semantics

The problem of behavioral compliance of modeling languages such as those of state machines arises from the fact that there are many versions of the state machines. For example, there are classical state chart, Rhapsody state chart, and the UML state chart. The differences of these state machines are discussed in [15]. Some techniques for conformance testing based on state machine are developed and reported in [16]. The reviewer of this paper points out the significant issues of handling temporal constraints in tools and the hierarchical semantics of state machines. We believe that these issues are very important and will address the solutions to these problems in the next stage of the work.

5.4 Model Interchange

A significant step towards tool interoperability is to establish the compliance of software tools to the XML Model Interchange (XMI). This compliance would let tool users exchange models between tools correctly. The XMI Interoperability Testing Task Force (TTF), a subgroup of the OMG, has been engaged in development of the test suites that certify the compliance with respect to the XMI specification. This XMI compliance test suite for UML tools is very large and very complex. We have learnt that the syntactical problem of XMI/UML tool certification by itself is non-trivial and in fact very challenging.

6. An Approach to Conformance Assessment

Given a software standard for modeling software systems, and a software tool that supports the software modeling activities; we claim that, it is possible to systematically ascertain that the software tools really comply with the software development standards. To measure the degree of compliance, Emmerich et al. [17] suggest that

"Standards generally define practices in terms of constraints that must hold for documents… The degree of standards conformance can be established by checking these documents against the constraints."

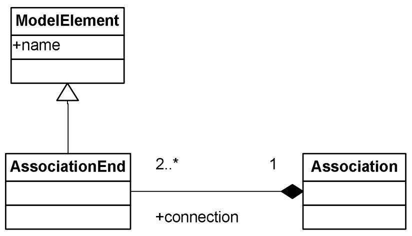
The implementation of this approach is the experimentation on the conditions of compliance, represented as the work products – the documents - that the software tools are suppose to create and maintain. A compliance test case for software modeling tools is a pair of a test input which is an actual software model and an expected output which is the preferred result determined by the constraints in the standard. For each compliance test case, the software tool imports the test input models, verifies the models and reports the verification result. This verification result generated from the software tool under test is compared with the preferred result and a pass/fail conclusion for the test case can be made.

6.1 Compliance test suite

The compliance test suite is composed of the test input models from two categories: those that satisfy the well-formedness rules (demonstrations) and those that violate the well-formedness rules (counterexamples).

The demonstrations are required as null hypotheses that assess the state in which the tools do not reject the correct models. This is due to the fact that the well-formedness rules might be implemented incorrectly in an over-constrained manner. As a result, a correct model may be rejected because the constraint is to strong. The counterexamples are required as alternative hypotheses that assess the state in which the tools do not detect the errors in the models and even accept incorrect models. This is of course due to the fact that the well-formedness rules might be implemented in an under-constrained manner.

For example, consider an OCL well-formedness rule and its relevant metamodels in Figure 2. The rule enforces that an AssociationEnd must have a unique name within an Association.



self.allConnections->forAll(r1,r2 | r1.name = r2.name implies r1 = r2)

Figure 2 metamodel and well-formedness rule

As a smallest possible example, for the models that are bound to one Association, two AssociationEnds, and two names, we can generate the demonstration (left) and counterexample (right) as shown in Figure 3 below.

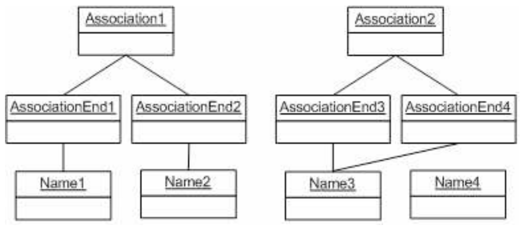


Figure 3 demonstration and counterexample

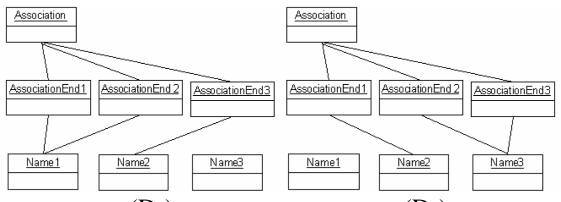
7. Implementation

7.1 The challenges

Software modeling standards consist of many model elements and relationships. Add to them the well-formedness rules constraining these complex structures; generating a correct and complete compliance test suites is not straightforward. Firstly, many structures may not be considered as test input models because they are not valid. For example, we randomly arrange a model with one Association, three AssociationEnds, and three names by taking every instance as a node in a digraph may generate 2^{49} models (over 562 billion), of which only 108 (4 * 3 * 3 * 3) models are valid. Secondly, it is hard to generate a complete set of test inputs that cover all possible structures as the size of instances grows, because the size of the valid models grows exponentially, in some case, at the rate of $O(n^n)$. To generate valid test inputs that contain up to ten AssociationEnds and ten names will result in over 10 billion valid test input models. Finally, since well-formedness rules expressed in first-order logic are undecidable, to know when to stop the test poses another challenge. This must take into consideration the size of instances in the models in the real world and the practicality of running a very large test suite on the software tools under test.

7.2 Model Isomorphism

Testing all models in the range typically involves testing many structurally equivalent models. For example, the model consisting three AssociationEnds in Figure 4(D_0) is equivalent to the model in Figure 4(D_1) structurally (we call this model isomorphic). Both models have two AssociationEnds share the same name and the other AssociationEnd has a different name. As there is no advantage in testing more than one input from one structurally equivalence class, only one representative of the two models needs to be tested.



(D_0) (D_1)

Figure 4: two isomorphic UML models

To prove that only one representative is required from each equivalence class, we first give a definition of structural equivalence, denoted by $\cong$. Then we will show that structural equivalence is equivalence relation and that the family of structural equivalence is a partition of all model structures. Therefore, a partition is a distinct model configuration.

Definition 9 Let $I_0,\ldots,I_e$ be the sets of instances of e meta elements and $I = I_0 \cup \ldots \cup I_e$. A model D is the set $M = M_0 \cup \ldots \cup M_e$ where $M_0 \subseteq I_0 \ldots M_e \subseteq I_e$ and a set of links $L = L_0 \cup \ldots \cup L_r$ such that, for each relationship $R_l(I_{source}, I_{target})$, $L_l = \{(m_i, m_j) \mid m_i \in M_{source} \;\&\; m_j \in M_{target}\}$

Definition 10 Let $\wp$(D) be the function mapping a model to a list of partition pairs $\{PP_0,\ldots,PP_r\}$ such that PP_l represents a pair of integer partitions $[P_{source}, P_{target}]$ where $P_{source} = (p_0, p_1,\ldots, p_p)$ denotes an integer partition with $p_0 \geq p_1 \geq \ldots \geq p_p$. For every partition pair $PP_l = [P_{source}, P_{target}]$, P_{source} is the list of the numbers of outward links from each instance m_i in M_{source} and P_{target} are the list of the numbers of inward links to each instance m_j in M_{target} respectively.

Definition 11 Suppose that D_0 and D_1 are two models consisting of instances from I, $D_0 \cong D_1$ iff $\wp(D_0) = \wp(D_1)$.

For example, two UML models consist of the set $I_0 = \{A0\}$ for Association element, the set $I_1 = \{E0, E1, E2\}$ of AssociationEnd elements, and the set $I_2 = \{N0, N1, N2\}$ of String values. The two models D_0 and D_1 will be regarded as isomorphic iff all partition pairs of the models are identical. As show in figure 4, the two models are isomorphic: witness the partition pairs {[(3), (1,1,1)], [(1,1,1), (2,1,0)]} representing the model D_0 on the left and the partition pairs {[(3), (1,1,1)], [(1,1,1), (2,1,0)]} representing the model D_1 on the right respectively.

Theorem Suppose M is the set of all model structures, and R is the model isomorphic relation, in other words $R = \{(m,m') \in M \times M \mid m \cong m'\}$, then R is an equivalence relation. Thus, a family

$$F_c = M_c \subseteq M, \exists c \in C \forall m \in M_c \mid R(c,m)$$

is by definition a partition of M. Therefore, C is the set of all distinct model configurations such that

$$\forall m \in M \exists m' \in C \mid R(m,m').$$

Proof A relation is an equivalence relation if it is reflexive, symmetry, and transitive. Clearly every model is structurally equivalent to itself, $\wp(m) = \wp(m)$, so R is reflexive. Also if model m is structurally equivalent to model n, the n is structurally equivalent to model m, $\wp(m) = \wp(n) \Leftrightarrow \wp(n) = \wp(m)$, so R is symmetry; and if l is structurally equivalent to m and m is structurally equivalent to n, then l is structurally equivalent to n, $\wp(l) = \wp(m) \;\&\; \wp(m) = \wp(n) \Leftrightarrow \wp(l) = \wp(n)$, so R is transitive. Therefore R is equivalence relation on M. we prove that a family on an equivalence relation is a partition, in other words, $\cup F_c = M$ and $\cap F_c = \phi$.

Constructing C will give all configurations required in the test suite.

7.3 Test Generation Tool

The abstract test models are generated with the combinatorial algorithms that enumerate all test models within a given bound on the number of instances of the model elements. We have implemented the tool JULE that takes a metamodel and the number of instances for the test input models and generates abstract test models in Rigi Standard Format (RSF). These abstract test models and classified into demonstrations and counterexamples using Crocopat. To generate the compliance test suite, both demonstrations and counterexamples are concretized into XMI documents. We report two examples of the results on using JULE and Crocopat for generating the test suite. The first example continues our example on the uniqueness of the AssociationEnds. In practice, it is very rare to have an Association containing as much as ten AssociationEnds, but still the test cases are generated within a reasonable size.

Example 1 The AssociationEnds must have a unique name within the Association.

self.allConnections->forAll (r1, r2 | r1.name = r2.name implies r1=r2)

#n	#structures	#test cases	#demon-strations	#counter examples
2	4	2	1	1
3	108	6	3	3
4	2,816	15	7	8
5	81,250	28	13	15
6	2,659,392	55	24	31
7	98,825,160	90	39	51
8	414,392,352	154	64	90
9	194,485,085,478	240	98	142
10	10,130,000,000,000	378	150	228

As for another example, the below OCL constraint on the Classifier may require a much larger number of test cases. We reduce its complexity by generating the instances of Operation and Method but not Reception. This can be done easily in JULE by setting the size of Reception to 0. For this rather complex rule (involving six model elements), the result also confirms that the test suite that covers all possible false can be generated within a reasonable size of the test input models.

Example 2 No BehavioralFeature of the same kind may match the same signature in a Classifier

self.feature->forAll(f, g |
(((f.oclIsKindOf(Operation) and g.oclIsKindOf(Operation))
or(f.oclIsKindOf(Method) and g.oclIsKindOf(Method))
or(f.oclIsKindOf(Reception) and g.oclIsKindOf(Reception))) and
f.oclAsType(BehavioralFeature).matchesSignature(g)) implies f = g)

#n	#structures	#test cases	#demon-strations	#counter examples
1	4	2	2	0
2	256	12	8	4
3	46,656	42	22	20
4	16,777,216	120	56	64
5	10,000,000,000	300	126	174
6	8,916,100,448,256	696	275	421
7	11,112,006,825,558,000	1470	549	921

8. Conclusion and Future Works

This paper presents the strategic framework for software tool certification together with a compliance test suite generation technique. In the next step, we will evaluate these framework and techniques on realistic software tools including the UCLUML repository which was automatically generated with the UCLMDA tool [18]. Also we will run the test suite on some open source and commercial tools including ArgoUML, Poseidon, Enterprise Architect, MagicDraw and AndroMDA. These tools are known to support the well-formedness verification of the UML models. From these case studies, the correctness of the assessment, and the validity of the certification will be evaluated.

9. Acknowledgements

Panuchart Bunyakiati acknowledges the support from the University of the Thai chamber of commerce, the comments of Wolfgang Emmerich and his colleagues and the reviewers of this paper who provide useful suggestions.

10. References

[1] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual, Addison Wesley, 2004.
[2] CCTA, Testing criteria for the SSADM version 4 tools conformance scheme, HMSO, London 1994.
[3] D. Marinov and S. Khurshid, "TestEra: A novel framework for automated testing of Java programs", Proc. 16th IEEE International Conference on Automated Software Engineering (ASE), San Diego, CA, November 2001.
[4] D. Jackson, Software Abstractions, MIT Press, MA, 2006.
[5] C. Boyapati, S. Khurshid and D. Marinov,"Korat: Automated testing based on Java predicates", Proc. the 2002 International Symposium on Software Testing and Analysis (ISSTA), Rome, Italy, July 22--24, 2002.
[6] A.Miczo, Digital Logic Testing and Simulation 2nd edition, John Wiley & Sons, New Jersey, 2003.
[7] D. Beyer,"Relational Programming with CrocoPat", Proc. the 28th international conference on Software engineering, pages 807-810, 2006.
[8] J. Voas,"The Software Quality Certification Triangle", The Journal of Defence Software Engineering, Nov 1998.
[9] R. C. Martin, UML for Java Programmers, Prentice Hall, 2003.
[10] OMG, Unified Modeling Language (UML) version 2.1.1, http://www.omg.org/technology/documents/formal/uml.htm, 2007.
[11] OMG,The Meta Object Facility (MOF) 2.0 Core Specification, Object Management Group, 2004.
[12] K. Hölscher, P. Ziemann and M. Gogolla,"On translating UML models into graph transformation systems", Journal of Visual Language and Computing, 2006.
[13] OMG, The Object Constraint Language (OCL) Specification version 2.0, Object Management Group, 2006.
[14] C. H. Papadimitriou, Computational Complexity, Addison Wesley, 1994.
[15] M. L. Crane and J. Dingel,"UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal", Proc. MoDELS/UML, Jamaica, October 2005.
[16] A. Gargantini, "Conformance Testing", Model-Based Testing of Reactive Systems, Springer, 2004.
[17] W. Emmerich, A. Finkelstein, S. Antonelli, S. Armitage, and R. Stevens,"Managing standards compliance", IEEE Transactions on Software Engineering, vol. 25, no. 6, 1999.
[18] J. Skene and W. Emmerich,"Specifications, not Meta-Models", Proc. ICSE 2006.