Testfallgenerierung aus UML- und Object-Z-Spezifikationen

Dehla Sokenou
Fachbereich 13 - Informatik, Fachgebiet Softwaretechnik,
Sekr. FR5-6, TU Berlin, Franklinstr. 28/29, D-10587 Berlin
dsokenou@cs.tu-berlin.de

1 Einleitung

Spezifikationsbasierte Testfälle können im Gegensatz zu aus der Implementierung erzeugten Testfällen Fehler aufdecken, die durch Abweichung der Implementierung von der Spezifikation entstanden sind. Zudem kann die Testfallgenerierung bereits in einer Phase beginnen, in der noch keine Zeile Programmcode geschrieben wurde.

Mit der weiten Verbreitung der UML als Sprache für den Modellierungsprozeß in der Softwareentwicklung besteht der Bedarf nach möglichst automatisierbaren Methoden und Techniken der Testfallgenerierung aus UML-Spezifikationen. Dabei erschweren die semiformalen und oft unvollständigen und nicht konsistenten Modelle die Testfallgenerierung. Dem gegenüber stehen formale Modelle wie zum Beispiel Spezifikationen in Object-Z.

Dieser Artikel beschränkt sich auf Spezifikationen in UML und Object-Z, um eine semiformale und eine formale Modellierung gegenüberzustellen. Darüberhinaus können sich UML und Object-Z ergänzen, wie die an unserem Fachgebiet um UML- und Z-Notationen modifizierte Fusion-Methode [2] zeigt, die eine Methodik zur Modellierung mit UML bietet. Zudem gibt es Ansätze, UML mit Hilfe von Object-Z zu formalisieren [6].

Da es bisher keinen integrierten Ansatz zur Testfallgenerierung aus UML-Spezifikationen gibt, werden im folgenden Techniken beschrieben, die einzelne Modelle der UML als Basis der Testfallgenerierung nehmen. Im darauffolgenden Kapitel wird eine Technik zur Testfallgenerierung aus Object-Z-Spezifikationen erläutert. Abschließend folgt ein Ausblick auf die Möglichkeiten, die ein integrierter Ansatz für UML bietet.

2 Testfallgenerierung und UML

Die UML bietet mit einer Vielzahl von Modellen verschiedene Sichten auf ein System. Die meisten dieser Modelle werden jedoch nur selten oder nie benutzt. Dieser Artikel beschränkt sich deshalb auf die meistgenutzten und damit wichtigsten Modelle: Klassendiagramm, Use Cases, Sequenzdiagramme und Statecharts.

Das Klassendiagramm kann allein nicht zur Testfallgenerierung genutzt werden, bietet jedoch die Möglichkeit der Nutzung für die Testvorbereitung. Es enthält Informationen über die Klassen des Systems und die Beziehungen zwischen den Klassen. Letztere lassen sich zur Ermittlung der Testreihenfolge heranziehen, zum Beispiel nach dem in [8] vorgestellten Algorithmus.

Auch die Use Cases sind allein für die Testfallgenerierung nicht anwendbar. Es gibt Ansätze, die Use Cases mit anderen Modellen einer Spezifikation in Beziehung zu setzen. [1] beschreibt ein Verfahren zur Testfallgenerierung mit Hilfe von Use Cases, Event Traces und der relativen Häufigkeit des Auftretens eines Use Cases im System. Die Testfallgenerierung erfolgt dann durch Überdeckung der Event Traces. Je häufiger ein Use Case auftritt, desto mehr Testfälle werden generiert.

Sequenzdiagramme enthalten nur wenig Information zur Testfallbildung, da sie das Systemverhalten nur unvollständig und ausschnittsweise beschreiben. Jedes Sequenzdiagramm sollte jedoch einen Testfall bilden, da das System mindestens das leisten muß, was dieses spezifiziert.

Die Statecharts in der UML sind der wohl formalste und in Bezug auf Testfallgenerierung der am besten untersuchte Teil der Spezifikation. Es gibt viele Ansätze zur Ableitung von Testfällen aus Statecharts oder verwandten Zustandsautomaten, sowohl kontrollflußorientierte als auch datenflußorientierte Techniken.

Eine Möglichkeit der Testfallgenerierung aus Zustandsautomaten ist die Nutzung der beim Modelchecking erzeugten Pfade, da das Modelchecking eine Verifikationsechnik ist, die eine Eigenschaft für einen Zustandsautomaten quasi vollständig testet [5].

Ein klassischer Ansatz der Generierung von Testfällen aus Zustandsautomaten ist der kontrollflußbasierte Ansatz. Analog zu den Überdeckungskriterien von Kontrollflußautomaten können Kriterien zur Überdeckung von Zuständen (Knoten), Transitionen (Kanten) und Pfaden definiert werden. Testfälle werden gebildet, indem durch Traversierung des Zustandsautomaten das gewünschte Kriterium erreicht wird, zum Beispiel die Überdeckung aller Zustände.

Die Technik von Chow [4] ermöglicht ein systematisches Vorgehen bei der Traversierung des Zustandsautomaten, das auch bei zyklischen Zustandsautomaten terminiert. Dabei werden alle Transitionen vom Initialzustand aus ausgeführt. Wird ein Zustand innerhalb eines Zweiges dabei ein zweites Mal erreicht, so wird an dieser Stelle die Ausführung der Transitionen beendet. Man erhält so einen Testbaum, der die Kriterien der Zustands- und Transitionsüberdeckung erfüllt und jeden Zyklus mindestens einmal durchlaufen hat.

Diese Technik ist jedoch zunächst nur auf flache Zustandsautomaten anwendbar. Die Arbeiten von Kim et al [7] und Kung et al [9] beschreiben Verfahren zum Umgang mit Hierarchie. [9] berücksichtigt darüberhinaus, daß es mehrere Initialzustände geben kann, die nicht gegenseitig erreichbar sind. Für diese wird jeweils ein eigener Testbaum erstellt.

Vererbung ist ein wichtiger Faktor in der Objektorientierung. McGregor beschreibt in [10] ein Verfahren, um Testfälle der Oberklasse für die Unterklasse wiederverwenden zu können. Dabei werden Zustandsautomaten der Unterklasse aus den Zustandsautomaten der Oberklasse konstruiert, so daß eine konforme Vererbung erreicht wird. So ist für die Unterklasse eine Bildung von Testfälle nur noch für die neuen Eigenschaften nötig.

Weitere Ansätze bedienen sich datenflußorientierter Techniken [7] oder kombinieren die Generierung von Testfällen auf Basis der Spezifikation mit der Testfallgenerierung auf Basis der Implementierung [1].

Es gibt viele Ansätze zur Testfallgenerierung aus UML-Spezifikationen, doch keiner dieser Ansätze nutzt alle verfügbaren Informationen.

3 Testfallgenerierung und Object-Z

Im Gegensatz zum tendenziell informellen Charakter von UML-Spezifikationen bieten die formalen Spezifikationen ein größeres Potenzial der Automatisierbarkeit des Testprozesses. So können Vor- und Nachbedingungen in formaler Form die automatische Festsetzung und Auswertung von Sollwerten ermöglichen.

Die in [3] vorgestellte Technik zur Generierung von Zustandsautomaten aus Object-Z-Spezifikationen ermöglicht die Anwendung der bereits vorgestellten Techniken für Zustandsautomaten. Allerdings wird hier lediglich ein Teil des Datenraumes - der die Vorbedingungen der Methoden betreffende - vom Benutzer partitioniert. Weitere Zustände werden automatisch aus dieser vorgegebenen Partitionierung mit Hilfe der Vor- und Nachbedingungen der Methoden berechnet.

Dieser Ansatz führt zu Zustandsautomaten, deren Partitionierung der Zustände weniger willkürlich ist als die Partitionierung der Statecharts in UML. Durch den hohen Grad an Automatisierung bieten formale Techniken zur Testfallgenerierung einen effizienten Ansatz.

4 Zusammenfassung und Ausblick

Zustandsautomaten sind bisher die einzigen Modelle der UML, aus denen in relevanter Weise Testfälle gewonnen werden können. Viele andere UML-Diagramme weren zur Zeit noch nicht genutzt, obwohl sie ebenfalls relevante Informationen enthalten. Ein integrierter Ansatz fehlt bisher, insbesondere in Kombination mit einem methodischen Vorgehen. Object-Z-Spezifikationen bieten durch ihren formalen Character Möglichkeiten, die UML nicht bietet und könnte ergänzend eingesetzt werden, zum Beispiel durch Transformation von UML nach Object-Z.

Es ist nötig, eine integrierte Technik zur Testfallgenerierung zu entwickeln, die die Sichten alle UML-Modelle berücksichtigt. Diese Technik muß weitgehend automatisierbar sein, um ihre Anwendbarkeit sicherzustellen. Dazu müssen die vorhandenen Techniken auf Anwendbarkeit und Automatisierbarkeit geprüft werden und um neue Techniken erweitert werden.

Literatur

[1] R.. Binder. The FREE Approach to Testing Object-Oriented Software: An Overview. http://www.rbsc.com/pages/FREE.html, 1996.

[2] M. Bittner and W. Koch. Die Fusion-Methode unter Verwendung von UML. Script, http://swt.cs.tu-berlin.de/lehre/swt/swt-ws00/unterlagen/Fusionbook.ps.gz, 2000.

[3] D. Carrington, I. MacColl, J. MacDonald, L. Murray, and P. Strooper. From Object-Z Specification to ClassBench Test Suites. Technical report, University of Queensland, 1998.

[4] T. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engeneering*, SE-4(3), 1978.

[5] A. Engels, L. Feijs, and S. Mauw. Test Generation for Intelligent Networks Using Model Checking. *LNCS*, 1217, 1997.

[6] S.-K. Kim and D. Carrington. A Formal Mapping between UML Models and Object-Z Specifications. Technical report, University of Queensland, 2000.

[7] Y. Kim, H. Hong, S. Cho, D. Bae, and S. Cha. Test Cases Generation from UML State Diagrams. In *IEEE Proceedings Software*, volume 146, 1999.

[8] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. A Test Strategy for Object-Oriented Systems. In *COMPSAC' 95*, 1995.

[9] D. Kung, N. Suchak, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. On Object State Testing. In *COMPSAC' 94*, 1994.

[10] J. McGregor. Constructing Functional Test Cases Using Incrementally Derived State Machines. In *11th ICTCS*, 1994.