Reengineering von Software-Komponenten zur Vorhersage von Dienstgüte-Eigenschaften

Klaus Krogmann
Lehrstuhl für Software-Entwurf und -Qualität
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe (TH)
krogmann@ipd.uka.de

1 Einleitung

Die Verwendung von Komponenten ist ein anerkanntes Prinzip in der Software-Entwicklung. Dabei werden Software-Komponenten zumeist als *Black-Boxes* aufgefasst [1], deren Interna vor einem Komponenten-Verwender verborgen sind. Zahlreiche Architektur-Analyse-Verfahren, insbesondere solche zur Vorhersage von nicht-funktionalen Eigenschaften, benötigen jedoch Informationen über Interna (bspw. die Anzahl abgearbeiteter Schleifen oder Aufrufe externer Dienste), die von den vielen Komponentenmodellen nicht angeboten werden.

Für Forscher, die aktuell mit der Analyse nicht-funktionaler Eigenschaften von komponentenbasierten Software-Architekturen beschäftigt sind, stellt sich die Frage, wie sie an dieses Wissen über Komponenten-Interna gelangen. Dabei müssen existierende Software-Komponenten analysiert werden, um die benötigten Informationen über das Innere der Komponenten derart zu rekonstruieren, dass sie für anschließende Analyse-Verfahren nicht-funktionaler Eigenschaften genutzt werden können. Bestehende Verfahren konzentrieren sich auf die Erkennung von Komponenten oder bspw. das Reengineering von Sequenzdiagrammen gegebener Komponenten, fokussieren aber nicht auf die Informationen, die von Vorhersageverfahren für nicht-funktionale Eigenschaften benötigt werden (vgl. Abschnitt 2).

Der Beitrag dieses Papiers ist eine genaue Betrachtung der Informationen, die das Reengineering von Komponenten-Interna liefern muss, um für die Vorhersage der nicht-funktionalen Eigenschaft Performanz (im Sinne von Antwortzeit) nutzbringend zu sein. Dazu wird das Palladio Komponentenmodell [2] vorgestellt, das genau für diese Informationen vorbereitet ist. Schließlich wird ein Reengineering-Ansatz vorgestellt, der dazu geeignet ist, die benötigten Informationen zu gewinnen.

2 Grundlagen

Um sinnvolle Analysen der nicht-funktionalen Eigenschaften einer Komponente zu ermöglichen, werden Informationen über die Interna einer Komponente benötigt. In Abbildung 1 werden die Interna als Abhängigkeiten zwischen einer angebotenen und benötigten Schnittstelle einer Komponente dargestellt. So kann der Aufruf von angebotenen Diensten

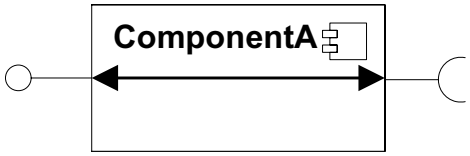

Abbildung 1: Abhängigkeiten zwischen angebotener und benötigter Schnittstelle

der Komponente (links) dazu führen, dass auch Dienste auf der benötigten Schnittstelle der Komponente (rechts) aufgerufen werden. Abhängig davon, wie der benötigte Dienst von einer externen Komponente bereitgestellt wird (beispielsweise niedrige oder hohe Ausführungszeit), verändert sich auch die vom Benutzer wahrgenommene Antwortzeit der betrachteten Komponente [3].

Um diese grob skizzierten Abhängigkeiten in ein Modell abzubilden, das die Grundlage für die Vorhersage nicht-funktionaler Eigenschaften einer Komponenten-Architektur ist, wurde das Palladio Komponentenmodell (PCM) geschaffen. Das PCM ist eine domänenspezifische Sprache, für die Analyse- und Simulations-Methoden [4, 2] existieren, die zur Vorhersage von Antwortzeiten, Ressourcenauslastung, usw. dienen. Zur Definition von Abhängigkeiten zwischen angebotenen und benötigten Diensten sind dabei die sogenannten Dienst Effekt Spezifikationen (SEFF) essentiell. SEFFs beschreiben das innere Verhalten einer Komponente in Form von Kontroll- und Datenfluss. Das Ziel eines SEFFs ist dabei, eine *Abstraktion* des inneren Verhaltens zu bieten.

In SEFFs wird grundsätzlich zwischen komponenteninternem Verhalten und externen Dienstaufrufen unterschieden. Darüber hinaus existieren Kontrollflusskonstrukte für Schleifen, Verzweigungen, Ressourcen-Aquisition und -Freigabe (insgesamt als „Aktionen“ bezeichnet). *Parametrisierung und parametrische Abhängigkeiten* erlauben es Abhängigkeiten zwischen Eingabeparametern eines Dienstes und Schleifenausführungen, Verzweigungswahrscheinlichkeiten, Parametern von externen Dienstaufrufen, usw. zu definieren. Die Erfassung von *Ressourcennutzung* (bspw. CPU-Einheiten, Festplattenzugriffe) ermöglicht eine Abbildung auf Hardware. Verzweigungswahrscheinlichkeiten und Schleifenausführungszahlen können

darüber hinaus auch *stochastisch* beschrieben werden – bspw. wenn die Angabe einer parametrischen Abhängigkeit hochkomplex und berechnungsintensiv würde.

Bislang existieren keine automatisierten Reengineering Ansätze, die alle benötigten Daten von SEFFs aus bestehenden Komponenten gewinnen – manuelle Ansätze scheiden aus Aufwandsgründen aus.

3 Reengineering Ansatz

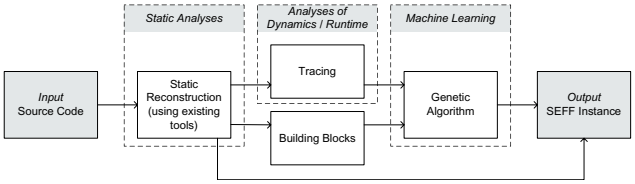

Abbildung 2: Angestrebter Reengineering Prozess

Wie im vorangehenden Kapitel beschrieben wurde, sind zahlreiche Daten für ein Modell der Komponenteninterna zur Vorhersage nicht-funktionaler Eigenschaften zu erfassen. Das Ziel eines solchen Modells ist es nicht, existierenden Quellcode möglichst genau abzubilden, sondern eine Abstraktion zu schaffen, die nur noch Parameter und parametrischen Abhängigkeiten usw. enthält, die für die Vorhersageverfahren signifikant sind.

Zu diesem Zwecke wollen wir statische Analyse, wie sie von existierenden Werkzeugen unterstützt wird, mit der Laufzeitanalyse in Form von Tracing kombinieren. Die aus beiden Schritten resultierenden Daten werden einem genetischen Algorithmus [5] zugeführt. Die Aufgabe des genetischen Algorithmus' ist das Auffinden einer Abstraktion parametrischer Abhängigkeiten. Dabei werden insbesondere dienstgüte-invariante Parameter herausgefiltert und funktionale Abhängigkeiten vereinfacht. In Abbildung 2 wird der vorgeschlagene Reengineering Prozess skizziert.

Der Reengineering-Vorgang beginnt mit der Eingabe von Quellcode, das Ziel der Vorgangs ist eine Modellinstanz eines SEFFs. Die statische Analyse liefert den Kontrollfluß der SEFFs. Zugleich wird eine für das Tracing benötigte Instrumentierung auf Basis der statischen Struktur vorgenommen. Ziel ist es hierbei, *gezielt* Laufzeitdaten zu erheben, um die anfallenden Datenmengen zu reduzieren. Daneben liefert die statische Analyse Bausteine – den initialen Genom – für den genetischen Algorithmus. Dies kann beispielsweise eine lineare Funktion für die Ausführungsdauer einer `for`-Schleife sein.

Die im Tracing-Schritt erhobenen und bereinigten Daten werden als Zielfunktion des genetischen Algorithmus' verwendet, um den Genom zu verbessern. Dabei muss der *Trade-Off* zwischen einfacher Berechenbarkeit der funktionalen Abhängigkeiten und die bereinstimmung mit den Tracing-Daten gelöst werden.

4 Fazit

Der angestrebte Mischung aus statischem und dynamischem Reengineering versucht die Vorteile beider Varianten zu vereinen. Die statische Analyse liefert wichtige Informationen für die Kontrollflussstrukturen des SEFFs, wohingegen die dynamische Analyse (Tracing) typische Probleme, wie die Bestimmung komplexer Schleifenabbruchbedingungen, vereinfacht.

Das Verhalten von Komponenten wird für die dynamische Analyse zur Laufzeit beobachtet. Daher ist es notwendig, dass Testdaten in die beobachteten Komponenten eingegeben werden. Dabei werden die gleichen Probleme wie beim Testen von Software auftreten: es kann niemals der gesamte Eingaberaum in endlicher Zeit abgedeckt werden. Entsprechend sind aussagekräftige Testdaten eine Grundvoraussetzung um sinnvolle Ergebnisse zu erlangen.

Zukünftige Arbeiten werden sich mit der Implementierung und Evaluation der vorgestellten Verfahrens-Idee beschäftigen.

Literatur

[1] Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. 2 edn. ACM Press and Addison-Wesley, New York, NY (2002)

[2] Becker, S., Koziolek, H., Reussner, R.: Model-based performance prediction with the palladio component model. In: Workshop on Software and Performance (WOSP2007), ACM Sigsoft (2007)

[3] Reussner, R.H., Schmidt, H.W.: Using Parameterised Contracts to Predict Properties of Component Based Software Architectures. In Crnkovic, I., Larsson, S., Stafford, J., eds.: Workshop On Component-Based Software Engineering (in association with 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems), Lund, Sweden, 2002. (2002)

[4] Koziolek, H., Firus, V.: Parametric Performance Contracts: Non-Markovian Loop Modelling and an Experimental Evaluation. In: Proceedings of FESCA2006. Electronical Notes in Computer Science (ENTCS) (2006)

[5] Koza, J.R.: Genetic Programming – On the Programming of Computers by Means of Natural Selection. third edn. The MIT Press, Cambridge, Massachusetts (1993)