

Minimierung aufwändiger Berechnungen als Grundlage für konsistente und interaktive Programmvisualisierungen

Markus Knauß

Abteilung Software Engineering

Institut für Softwaretechnologie

Universität Stuttgart

www.informatik.uni-stuttgart.de/iste/se

Jochen Wertenaue

mail@jwertenaue.de

Einleitung

Um ein Programm zu bearbeiten, muss es zunächst verstanden werden. Das Verstehen wird unterstützt durch berechnete Zusatzinformation, wie zum Beispiel Programmstruktur oder Aufrufbeziehungen. Die Zusatzinformation kann zu dem Programm visualisiert werden. Die Visualisierung ist besonders nützlich, wenn sie konsistent mit dem bearbeiteten Programm und interaktiv bedienbar ist. Die Berechnung der Zusatzinformation ist aber aufwändig, was die Realisierung einer konsistenten und interaktiv bedienbaren Visualisierung schwer macht.

Eine Grundlage für eine konsistente und interaktiv bedienbare Visualisierung kann die Minimierung der Berechnung der Zusatzinformation sein. Die Minimierung wird erreicht, indem nicht mehr konsistente Zusatzinformation identifiziert und nur diese neu berechnet wird. Diese Idee wurde in einer Arbeit an der Universität Stuttgart untersucht (Wertenaue, 2007). Resultat ist das Werkzeug *codation*, mit dem nicht mehr konsistente Zusatzinformation identifiziert wird. Dieser Artikel berichtet über die Arbeit und die erreichten Ergebnisse.

Im nächsten Abschnitt beschreiben wir den Lösungsansatz. Anschließend stellen wir die technische Realisierung als Eclipse-Plugin vor. Danach berichten wir die Ergebnisse einer kleinen Fallstudie, in der das Werkzeug evaluiert wurde. Abschließend fassen wir zusammen und geben einen Ausblick auf mögliche weitere Arbeiten.

Lösungsansatz

Der Lösungsansatz beruht auf der Idee, Änderungen am Programmcode möglichst genau zu erkennen und nur die Zusatzinformation neu zu berechnen, die von den Änderungen betroffen ist. Um diese Idee zu realisieren, sind zwei Fragen zu beantworten:

1. Wie können Änderungen erkannt werden?
2. Wie kann die von einer Änderung betroffene Zusatzinformation erkannt werden?

Um die beiden Fragen zu beantworten und die Vorgaben einzuhalten, Änderungen möglichst genau zu erkennen und nur die Information neu zu berechnen, die tatsächlich von der Änderung betroffen ist, muss zunächst die Frage beantwortet werden:

3. Auf welche Teile des Programms bezieht sich die Information?

Im Rahmen der Arbeit an der Universität Stuttgart wurden die Fragen speziell für Java-Programme beantwortet. Darum beziehen sich die weiteren Ausführungen auf Java-Programme.

Die Berechnung der Zusatzinformation für Java-Programme beruht meist auf dem AST. Darum bezieht sich die Zusatzinformation für Java-Programme auf einzelne Knoten des ASTs.

Um Änderungen zu erkennen, wurde das Programm vor der Änderung mit dem nach der Änderung verglichen. Da die Bezüge zwischen Programm und Zusatzinformation über die AST-Knoten realisiert sind, müssen die beiden ASTs vor und nach der Änderung verglichen werden. Der Vergleich verwendet den Algorithmus für die Berechnung der Tree-Edit-Distance. Der Algorithmus berechnet die notwendigen Änderungen, die vom AST vor der Bearbeitung zum AST nach der Bearbeitung führen, in $O(n_1 n_2)$ (n_1, n_2 : Knoten des AST vor und nach der Änderung) (Bille, 2005).

Da der Vergleich der vollständigen ASTs des Programms vor und nach der Änderung zu aufwändig ist, wurde der Algorithmus für Java-Programme angepasst. Der Algorithmus vergleicht nur noch ASTs von Methoden, lokalen Typen oder Variablen, die tatsächlich verändert wurden.

Ergebnis der Änderungsberechnung sind die geänderten AST-Knoten. Da sich die Zusatzinformation auf die AST-Knoten bezieht, kann schnell festgestellt werden, welche Zusatzinformation von Änderungen betroffen ist.

Technische Realisierung

Codation wurde als Eclipse-Plugin realisiert. Es ist in Eclipse als Project-Builder registriert. Das hat die folgenden Vorteile: *codation* ist unabhängig von der

Programmiersprache, wird bei jedem Speichern über die geänderten Dateien informiert und kann Ressourcen im Workspace ändern.

Codation implementiert ein Rahmenwerk für die Änderungserkennung. Das Rahmenwerk besteht aus den drei Schnittstellen: Annotation-Provider, Storage-Provider und File-Link-Provider. Mit dem Annotation-Provider werden Zusatzinformationen in codation eingebracht, der Storage-Provider kümmert sich um die Speicherung der Zusatzinformationen und der File-Link-Provider verbindet die Zusatzinformationen mit dem Programmcode. Der File-Link-Provider ist auch zuständig für die Identifikation der nicht mehr konsistenten Zusatzinformationen. Der Storage-Provider verwaltet die Zusatzinformationen und die Bezüge zwischen den Zusatzinformationen und dem Programmcode in eigenen Dateien. Hierdurch bleibt der Programmcode unverändert, was die Nutzung existierender Werkzeuge ermöglicht. Soll codation in einem eigenen Projekt genutzt werden, dann müssen diese drei Schnittstellen für die eigene Anwendung implementiert werden.

Der Ablauf der Änderungsberechnung in codation ist wie folgt:

1. Mit den Annotation-Providern werden die von der Änderung betroffenen Zusatzinformationen bestimmt.
2. Von der Änderung betroffene Zusatzinformation wird geprüft, ob eine Aktualisierung notwendig ist.
3. Annotation-Provider nicht mehr konsistenter Zusatzinformationen werden benachrichtigt.

Die Berechnung der Änderungsinformation wird im Hintergrund ausgeführt, um den Entwickler bei seiner Arbeit nicht zu behindern.

Fallstudie

Die Nützlichkeit von codation wurde in einer kleinen Fallstudie untersucht. Für die Fallstudie wurde eine Anwendung entwickelt, mit der Use-Cases mit dem Programmcode verknüpft werden können. So ist es möglich Use-Cases zu erkennen, deren Realisierung nach einer Änderung des Programms, zum Beispiel durch Testfälle, geprüft werden müssen. Es zeigte sich, dass codation gut geeignet ist für die Realisierung der Anwendung und die Änderungsberechnung sehr genau anzeigt, welche Use-Cases geprüft werden müssen. Allerdings ist festgestellt worden, dass die im Hintergrund laufende Änderungserkennung sehr aufwändig ist und die Arbeit behindern kann.

Um die Auswirkungen von Änderungen auf das Laufzeitverhalten von codation näher zu untersuchen, wurden Performanzmessungen durchgeführt. Es wurde festgestellt, dass die Laufzeit von der Größe der geänderten Methoden abhängt. Die Größe einer Methode spiegelt sich in einem größeren AST wieder, was zu einem höheren Aufwand in der Änderungsberechnung führt. Die Anzahl der Methoden in einer Klasse oder die Art der Änderung spielt hingegen keine Rolle.

Zusammenfassung und Ausblick

Dieser Artikel hat einen Ansatz vorgestellt, mit dem die aufwändige Berechnungen von Zusatzinformation zu einem Programm auf ein Minimum reduziert werden kann. Die Minimierung ist notwendig, um konsistente und interaktiv bedienbare Visualisierungen der Zusatzinformation zu ermöglichen. Durch die konsistente und interaktiv bedienbare Visualisierung wird das Verstehen und Bearbeiten des Programms, zwei wichtige Arbeiten in der Software-Wartung, unterstützt. Der Ansatz wurde im Werkzeug codation als Eclipse-Plugin realisiert. Für die Änderungserkennung in Java-Programmen auf Basis des AST wurde der Algorithmus für die Berechnung der Tree-Edit-Distance verwendet und angepasst. Die Nützlichkeit und Effizienz des Werkzeugs wurde in einer kleinen Fallstudie untersucht.

In zukünftigen Arbeiten kann die Berechnung der Änderungsinformation noch weiter optimiert werden, zum Beispiel indem Heuristiken eingesetzt werden, um die Anzahl der Knoten im untersuchten AST zu reduzieren. Ein anderer Ansatzpunkt für Optimierungen ist, zu berechnen, ab welcher AST-Größe es sinnvoller ist, eine Methode vollständig als geändert zu markieren, als detaillierte Änderungsinformationen zu berechnen.

codation kann von der Website: www.jwertenauer.de/ger/uni/da/index.shtml (3.4.2007) heruntergeladen werden. Das Werkzeug steht unter der Eclipse Public License (EPL) und kann in eigenen Projekten eingesetzt werden.

Literatur

- Bille, Philip (2005): A Survey on Tree Edit Distance and Related Problems. *Theoretical Computer Science*, 337(1-3), 217-239.
- Wertenaue, Jochen (2007): codation – Verbindung von Code mit Zusatzinformation. Diplomarbeit Nr. 2521, Universität Stuttgart.