Erzeugung dynamischer Objektprozessgraphen zur Laufzeit

Jochen Quante
Arbeitsgruppe Softwaretechnik
Fachbereich 3, Universität Bremen
http://www.informatik.uni-bremen.de/st/
quante@informatik.uni-bremen.de

1 Einführung

Objektprozessgraphen beschreiben den Kontrollfluss eines Programms aus Sicht eines einzelnen Objekts. Sie enthalten Informationen darüber, wie Objekte benutzt werden und wie diese Benutzungen in Beziehung zueinander stehen. Damit können sie für das Programmverstehen hilfreich sein (z.B. Protokollerkennung, Visualisierung). Allerdings ist die Extraktion solcher Graphen sehr aufwändig. Wir stellen eine neue dynamische Extraktionstechnik vor, die entsprechende Graphen schon während des Programmlaufs aufbaut und damit neue Anwendungen ermöglicht.

2 Extraktion dynamischer Objektprozessgraphen

Ein *Objektprozessgraph* ist ein Teilgraph des interprozeduralen Kontrollflussgraphen, der nur die Knoten und Kanten enthält, die für den Kontrollfluss aus Sicht eines bestimmten Objekts relevant sind [2]. Zur dynamischen Extraktion eines solchen Graphen wird zunächst das zu analysierende Programm so instrumentiert, dass alle Kontrollfluss- und Objekt-relevanten Informationen erfasst werden. Indem Instrumentierungspunkte auf Knoten und Ausführungsbeziehungen auf Kanten abgebildet werden, kann daraus ein entsprechender "Rohgraph" konstruiert werden, der durch Transformationen zum dynamischen Objektprozessgraphen (DOPG) wird. Das genaue Verfahren ist in [2] beschrieben. Abb. 1 zeigt ein Beispiel für die Konstruktion eines solchen Graphen während der Transformation.

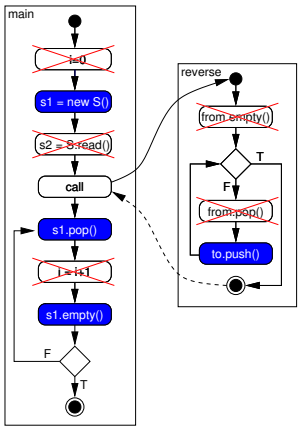
Im bisherigen Ansatz ("Offline") werden die Informationen aus der Instrumentierung in eine Datei geschrieben und erst nach Programmende analysiert. Der grobe Ablauf ist in Abb. 2(a) skizziert. Er beinhaltet einige aufwändige Schritte:

- Schreiben der Protokolldatei mit ca. 1-6 Mio. Ereignissen pro Sekunde. Dieser Schritt kann durch Kompression und Pufferung beschleunigt werden.
- Herausfiltern der relevanten Daten für einzelne Objekte. Die ursprüngliche Protokolldatei enthält die Daten für *alle* gewählten Objekte.
- Einlesen der gefilterten Daten zur Konstruktion des Graphen.

```
void main() {
  int i = 0;
  S s1 = new S();
  s2 = S.read();
  reverse(s2, s1);
  do {
    s1.pop();
    i = i + 1;
  } while (!s1.empty());
}

reverse(S from, S to) {
  while (!from.empty())
    to.push(from.pop());
}
```

(a) Quelltext



(b) DOPG für s1

Abbildung 1: Quelltext und DOPG für Anwendung eines Stack-Objekts. Die durchkreuzten Knoten werden im nächsten Schritt entfernt.

Um die Konstruktion zu beschleunigen, schlagen wir ein Verfahren vor, das auf das Schreiben der Protokolldatei verzichtet und die Graphen aufbaut, während das Programm noch läuft. Dies wird im folgenden genauer beschrieben.

3 Extraktionsalgorithmus

Die Konstruktion des Graphen zur Laufzeit ist in Abb. 2(b) skizziert und verläuft in folgenden Schritten (Details in [1]):

- Mitführung eines Graphen, der den aktuellen Aufrufpfad beschreibt. Dazu wird anhand der auftretenden Ereignisse ein Graph konstruiert (wie oben beschrieben). Wird eine Methode wieder verlassen, so wird der gesamte Methodenaufruf aus dem Graphen entfernt.
- Wenn eine zu beobachtende Klasse instanziiert wird, so wird dieser Graph kopiert. Die Kopie stellt den konkreten "Rohgraph" für dieses Objekt dar.
- Für alle weiteren Ereignisse werden sowohl zum Aufrufpfad-Graph als auch zu allen Objekt-Graphen entsprechende Kanten und Knoten hinzugefügt. Dabei werden Objekt-spezifische Ope-

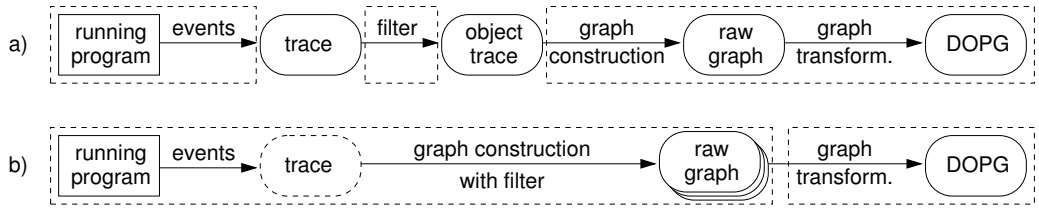


Abbildung 2: Offline- (a) und Online-Verfahren (b) im Vergleich.

rationen nur auf den zugehörigen Graphen angewendet. Bei den Kopien werden Methodenaufrufe nur dann entfernt, wenn der jeweilige Aufruf nicht relevant war. *Relevant* ist ein Aufruf genau dann, wenn er den Aufruf einer Methode oder den Zugriff auf ein Attribut des Objekts *oder* einen relevanten Aufruf enthält.

Damit wird zu jeder Instanz einer zu beobachtenden Klasse ein eigener Graph mitgeführt. Die Datenstrukturen werden so gewählt, dass ein effizientes Prüfen und Hinzufügen von Methodenaufrufen möglich ist. Da Java-Programme analysiert werden sollen, muss bei der Implementierung auch Multithreading berücksichtigt werden. Dies bedeutet, dass der Graph möglicherweise an mehreren Stellen gleichzeitig erweitert wird.

4 Fallstudie

Wir vergleichen nun den Laufzeitoverhead der Online- und Offline-Extraktionstechniken anhand mehrerer Beispiele. Dazu werden drei Java-Anwendungen unterschiedlicher Größe betrachtet: ArgoUML, J (ein Editor) und JHotDraw. Für jede der Anwendungen werden verschiedene Klassen herausgegriffen und für deren Instanzen DOPGs sowohl mit der Online- als auch mit der Offline-Variante erzeugt. Die benötigte CPU-Zeit wird gemessen. Da es sich bei allen Anwendungen um interaktive Systeme handelt, werden alle Szenarien mehrfach durchlaufen und Mittelwerte gebildet, um Variationen in der Bedienung auszugleichen.

Abb. 3 zeigt die Ergebnisse dieser Studie. Der durch die Instrumentierung verursachte Laufzeitoverhead ist sehr unterschiedlich: Für J liegt er zwischen Faktor 7 und 22, während er für JHotDraw bei 2 bis 3 liegt. In den meisten Fällen ist die Online-Variante aber mindestens genauso performant wie die Offline-Variante. In allen Fällen waren die Anwendungen ohne größere Verzögerungen weiterhin benutzbar.

Für das Szenario "Quad" brauchte die Online-Extraktion mehr Rechenzeit als für die Offline-Variante. Dies liegt darin begründet, dass in diesem Fall zehn Instanzen parallel verfolgt wurden, während in den anderen Fällen nur für jeweils eine Instanz ein Graph erstellt werden musste. Dies deutet darauf hin, dass die Online-Extraktion insbesondere in Fällen mit wenigen zu verfolgenden Objekten sinnvoll einsetzbar ist.

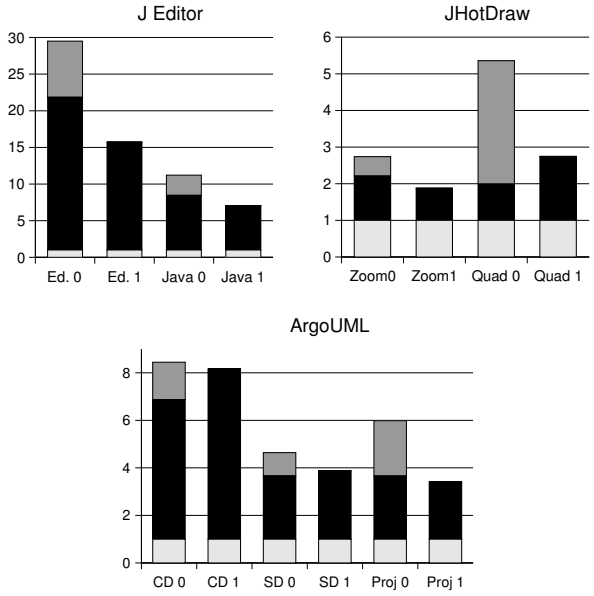


Abbildung 3: Ergebnisse der Fallstudie. Auf der Y-Achse ist der Laufzeitoverhead (schwarz) als Faktor im Vergleich zur normalen Laufzeit angegeben. Der linke Balken (0) steht jeweils für die Offline-Extraktion, der rechte Balken (1) für die Online-Variante. Für die Offline-Extraktion ist zusätzlich die Zeit für das Filtern angegeben (grau).

5 Fazit

Die Fallstudie zeigt, dass die Konstruktion von dynamischen Objektprozessgraphen in der Praxis effizient möglich ist. In vielen Fällen, insbesondere bei einer geringen Anzahl betrachteter Objekte, bremst die Online-Extraktion das beobachtete Programm weniger als das reine Schreiben der Protokolldatei. Damit ergeben sich ganz neue Anwendungsmöglichkeiten: Die Graphen können zur Laufzeit visualisiert werden und so einen Einblick geben, wie sich bestimmte Benutzeraktionen in Bezug auf ein Objekt auswirken.

Literatur

[1] Jochen Quante. Online construction of dynamic object process graphs. In *Proc. of 11th CSMR*, pages 113–122, 2007.

[2] Jochen Quante and Rainer Koschke. Dynamic object process graphs. In *Proc. of 10th CSMR*, pages 81–90, 2006.