UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Jens Kosiol

Graph- and Model-Driven Engineering

Übungsblatt 2 – Lösungsskizze

1 Entwurf Graphprogramm (8 Punkte)

Schreiben Sie ein Graphprogramm in Groove, das einen gegebenen Eingabegraphen kopiert.

Hierfür betrachten wir den Typgraphen des Blumengeschäfts aus der Groovebeispielgrammatik flower-market (Abbildung 1; Achtung: Plant ist hier abstrakt).

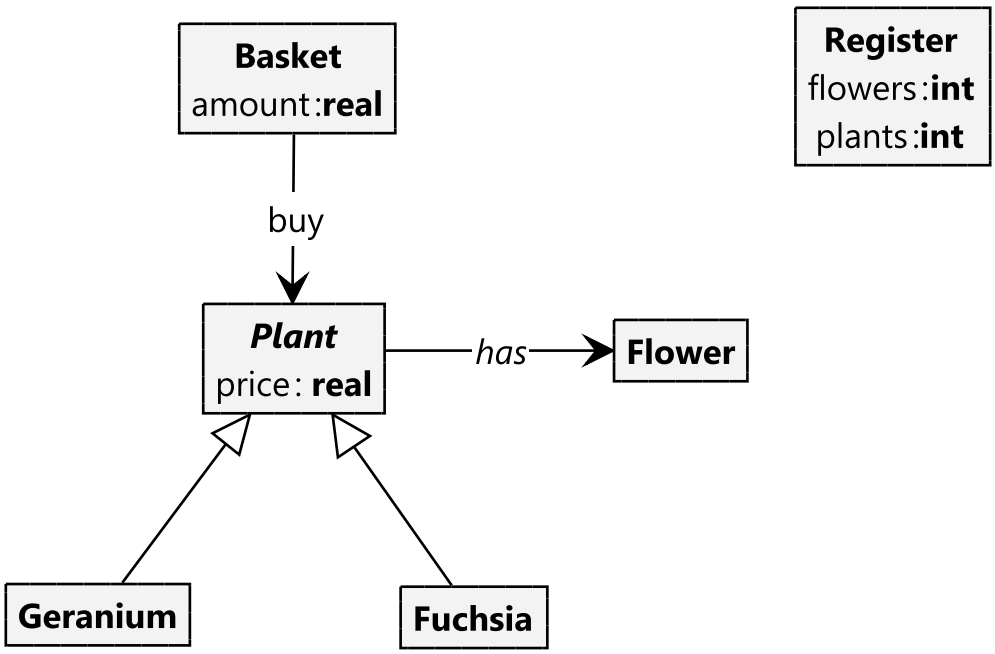

Abbildung 1: Typgraph zur Modellierung eines Blumengeschäfts

Ihr Programm soll die folgenden Anforderungen erfüllen:

- Gegeben einen beliebigen über dem Typgraphen konkret getypten endlichen Startgraphen soll ihr Programm terminieren und der Ergebnisgraph soll den Ausgangsgraphen und eine Kopie davon enthalten. (6 Punkte)
- Sie sollen drei verschiedene (sich deutlich unterscheidende) Startgraphen erstellen und an diesen testen, dass ihr Programm wie erwartet funktioniert. (2 Punkte)

- Sie dürfen selbst entscheiden, mit welchen Techniken Sie ihr Programm umsetzen wollen (explizite Kontrollstruktur, Regelpriorisierung, implizite Kontrolle wie im Vorlesungsbeispiel der Turingmaschinen, NACs, ...). Sie sollen aber **keine Multiregeln verwenden**, also nicht einfach die Idee aus der Groove-Grammatik copy-graph verwenden.

Lösung

Lösungsvorschlag in der Groove-Datei Ueb02_CopyFlowershop.gps; die Verwendung von Prioritäten oder die Verwendung der (deaktivierten) Kontrollstruktur sind äquivalent.

2 Modellierung mit Multiregeln (8 Punkte)

a) Entwerfen Sie einen Typgraphen TG, der das Modellieren von (vereinfachten) Klassendiagrammen erlaubt. Es soll Klassen, Attribute und Methoden geben; diese Objekte sollen jeweils einen Namen haben können. Klassen sollen voneinander erben können und Attribute und Methoden haben. Methoden können von anderen Methoden und von Attributen abhängig sein. Attribute sind von einem der primitiven Datentypen, die Groove bereitstellt (bool, int, string, real). (1 Punkt)

b) Spezifizieren Sie das Refactoring *Collapse Hierarchy* (vgl. z. B. hier) als Graphprogramm; **verwenden Sie hierbei an passender Stelle das Konzept einer Multiregel**. Darüber hinaus dürfen Sie alle in der Vorlesung behandelten Techniken verwenden. Ihr Programm soll ausgehend von einer zufällig gewählten Klasse alle weiteren Klassen, die (transitiv) von der gewählten Ausgangsklasse erben, löschen und ihre Methoden und Attribute stattdessen in die Ausgangsklasse umziehen. (6 Punkte)

   Sie dürfen den Abschnitt über *Reguläre Ausdrücke* als Kantenlabel in der Anleitung zu Groove lesen (Kap. 3.2) und diese Technik nutzen. Sie dürfen außerdem davon ausgehen, dass es keine Mehrfachvererbung gibt und das Attribute und Methoden immer genau einer Klasse zugeordnet sind.

c) Spezifizieren Sie zwei verschiedene (sich deutlich unterscheidende) Startgraphen und testen an diesen, dass ihr Programm wie erwartet funktioniert. In mindestens einem dieser Graphen soll es eine Vererbungshierarchie von Tiefe > 2 geben. (1 Punkt)

Lösung

Lösungsvorschlag in der Groove-Datei Ueb02_Klassendiagramme.gps.

3 Extraktion von Visual Contracts (12 Punkte)

In dieser Aufgabe sollen Sie sich mit Visual Contracts im Kontext des Werkzeugs *NanoXML* befassen. Gegeben ist ein Auszug der Klasse XMLElement (Listing 1).

a) Lesen Sie den Code, achten Sie auf vorkommende Klassen, Attribute und Referenzen und entwerfen Sie einen geeigneten Typgraphen. (1 Punkt)

b) Erstellen Sie ein Skript (Abgabe in Pseudocode), das die Methode removeAttribute mehrfach so aufruft, dass eine Branch Coverage des Codes erreicht wird. Geben Sie in ihrem Skript die Werte, mit denen relevante Parameter aufgerufen werden, konkret an. (1 Punkt)

c) Geben Sie für jeden Methodenaufruf aus ihrem Skript die jeweils daraus ableitbare Visual Contract Instanz und die zugehörige minimale Regel an. (4 Punkte)

d) Geben Sie für die Methode *addChild* drei Aufrufe und Ausgangssituationen an, die zu drei unterschiedlichen Visual Contract Instanzen führen, sodass aber zwei dieser Visual Contract Instanzen zur gleichen minimalen Regel führen. Geben Sie auch die maximalen Regeln an und etwaige logische Bedingungen, die (hoffentlich) für Attribut- und Parameterwerte gefunden werden. (6 Punkte)

Jens Kosiol

Graph- and Model-Driven Engineering

Übungsblatt 2 – Lösungsskizze

Listing 1: Auszug des Quellcodes der NanoXML-Klasse XMLElement.

```
class XMLAttribute {
  private String fullName, name;
}

public class XMLElement implements IXMLElement {
  private Vector attributes, children;
  private String name, content;
  private IXMLElement parent;

  /**
   * Adds a child element.
   */
  public void addChild(IXMLElement child) {
    if (child == null) {
      throw new IllegalArgumentException("child must not be null");
    }
    if ((child.getName() == null) && (! this.children.isEmpty())) {
      IXMLElement lastChild = (IXMLElement) this.children.lastElement();

      if (lastChild.getName() == null) {
        lastChild.setContent(lastChild.getContent() + child.getContent());
        return;
      }
    }
    ((XMLElement) child).parent = this;
    this.children.addElement(child);
  }

  /**
   * Removes an attribute.
   *
   * @param name the non-null name of the attribute.
   */
  public void removeAttribute(String name) {
      for (int i = 0; i < this.attributes.size(); i++) {
          XMLAttribute attr = (XMLAttribute) this.attributes.elementAt(i);
          if (attr.getFullName().equals(name)) {
              this.attributes.removeElementAt(i);
              return;
          }
      }
```

```
    }
  }
```

Lösung

a) Vorschlag für einen geeigneten Typgraphen ist Teil der Groove-Datei Ueb02_NanoXML.

b) Eine mögliche Lösung befindet sich in Listing 2. Zu beachten ist, dass bereits der 2. Test alleine vollen Branch-Coverage erreicht, da die Bedingung für das if einmal wahr und einmal falsch ist. Die zwei Tests werden konstruiert, um zu zeigen, dass es zwei verschiedene minimale Regeln geben kann. (Außerdem nimmt die Lösung die Existenz von Konstruktoren und Settern an, die im Code ausgeblendet sind.)

c) Lösung ist Teil der Groove-Datei Ueb02_NanoXML. Die Visual Contract Instanzen sind dabei als Graphen (jeweils Prä- und Post-Graph) angegeben, die minimalen Regeln als Regeln.

d) Aufrufsituationen: 1.) Aufruf mit null als Wert des Eingabeparameters; 2.) ein Kind ohne Name wird einem XMLElement ohne Kinder hinzugefügt; 3.) ein Kind ohne Name wird einem XMLElement hinzugefügt, dessen letztes Kind einen Namen hat. Die Lösung ist auch Teil der Groove-Datei Ueb02_NanoXML und genauso präsentiert wie oben. Die minimalen Regeln der Aufrufsituationen 2.) und 3.) stimmen überein und die maximale Regel entspricht in allen Fällen der minimalen. Für die Regel addChild1 ist noch der Visual Contract hinzugefügt (addChild1VC), in dem das parent-XMLElement noch als invarianter Kontext entfernt wird.

Listing 2: Pseudocode für Tests.

```
1  //create elements for the two test situations
2  //(assuming the necessary constructors and setters to exist)
3  XMLElement elementOne = new XMLElement();
4  XMLElement elementTwo = new XMLElement();
5
6  XMLAttribute attributeOne = new XMLAttribute("fullname1", "name1");
7  XMLAttribute attributeTwo = new XMLAttribute("fullname2", "name2");
8
9  elementOne.addAttributes(attributeOne);
10 elementTwo.addAttributes(attributeOne, attributeTwo);
11
```

```
12 //first  test
13 elementOne.removeAttribute("fullname2");
14
15 //second test
16 elementTwo.removeAttribute("fullname2");
```

4 Graphtransformationssysteme aus Code (12 Punkte)

Gegeben seien die unten folgenden Java-Klassen, die gemeinsam einen Algorithmus zum gegenseitigen Ausschluss zweier nebenläufiger Prozesse beim Zugriff auf eine Ressource implementieren. Der Algorithmus durchläuft dabei die folgenden Phasen:

- Anfordern: Einer der Prozesse fordert den Zugriff auf die Ressource an.
- Warten: Der Prozess wartet, bis er Zugriff auf die Ressource bekommt.
- Eintritt in Kritische Phase: Der Prozess bekommt Zugriff auf die Ressource.
- Veränderung durchführen: Der Prozess verändert die Ressource.
- Wechseln des nächsten Prozesses: Es wird festgelegt, welcher Prozess als nächstes Zugriff bekommt.
- Anfordern beenden: Der Prozess, der Zugriff hatte zieht seine Anforderung auf Ressourcenzugriff zurück.

Leiten Sie aus der Implementierung mit Groove ein typisiertes Graphtransformationssystem ab, das diesen Algorithmus für einen *UIProcess*, einen *LogicProcess*, eine *Ressource* und einen *Mutex* nachbildet. Sie dürfen entweder Kontrollstrukturen verwenden, die Groove für Graphprogramme bereitstellt (siehe S. 25 ff. in der Dokumentation), oder den Ablauf implizit steuern, indem Sie sich im Graphen zusätzliche Eigenschaften merken, die sich nicht eins zu eins im Java-Code wiederfinden. Um sich in typisierten Knoten Eigenschaften zu „merken“, können Sie z. B. *flags* einsetzen.

Lösung

Lösungsvorschlag in der gps-Datei Ueb02_Mutex.gps.