UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Jens Kosiol

Graph- and Model-Driven Engineering

Übungsblatt 4 – Lösungsskizze

1 Multiplizitäten als Graphformeln (6 Punkte)

Definieren Sie Multiplizitäten (Knoten- und Kantenmultiplizitäten wie in der Vorlesung als eigenständiges Konzept eingeführt) als (geschachtelte) Graphformeln. Das heißt, gegeben einen endlichen Typgraphen mit Multiplizitäten $TGM = (TGI, m_n, m_s, m_t)$, geben Sie drei Graphformeln c_n, c_s, c_m an, sodass ein Graph G den Typgraphen mit Multiplizitäten TGM erfüllt, genau dann wenn er die drei Graphformeln c_n, c_s, c_m erfüllt.

Hinweis: Die Graphformeln sind notwendig schematisch, da konkrete Formeln abhängig von der Anzahl der Knoten, Kanten und den konkreten Werten von m_n, m_s, m_t wären.

Lösung

Die Formeln lauten:

$$c_n = \bigwedge_{t \in N_{TGI}} c_n^{t,+} \wedge c_n^{t,-},$$
$$c_s = \bigwedge_{e \in E_{TGI}} c_s^{e,+} \wedge c_s^{e,-} \text{ und}$$
$$c_t = \bigwedge_{e \in E_{TGI}} c_t^{e,+} \wedge c_t^{e,-}.$$

Die Teilformeln sind als geschachtelte Graphformeln wie folgt definiert:

Für $m_n(t) = [i, j]$ gilt

$$c_n^{t,+} = \exists \left(\boxed{\text{1:t}} \quad \cdots \quad \boxed{\text{i:t}} \right)$$

und

$$c_n^{t,-} = \neg\exists \left(\boxed{\text{1:t}} \quad \cdots \quad \boxed{\text{j+1:t}} \right),$$

wobei $c_n^{t,+} = \top$, falls $i = 0$, und $c_n^{t,-} = \top$, falls $j = *$.

Für $m_s(e) = [i, j]$ gilt

Jens Kosiol

Graph- and Model-Driven Engineering

Übungsblatt 4 – Lösungsskizze

$$c_s^{e,+} = \forall \left(\text{t1:t}, \exists \begin{array}{c} 1:t' \\ \vdots \\ i:t' \end{array} \xrightarrow{e} \text{t1:t} \right)$$

und

$$c_s^{e,-} = \neg\exists \left(\begin{array}{c} 1:t' \\ \vdots \\ j+1:t' \end{array} \xrightarrow{e} \text{t1:t} \right),$$

wobei $c_s^{e,+} = \top$, falls $i = 0$, $c_s^{e,-} = \top$, falls $j = *$, und $src_{TGI}(e) = t'$ und $tar_{TGI}(e) = t$.
Für $m_t(e) = [i, j]$ gilt

$$c_t^{e,+} = \forall \left(\text{t1:t}, \exists \text{t1:t} \xrightarrow{e} \begin{array}{c} 1:t' \\ \vdots \\ i:t' \end{array} \right)$$

und

$$c_t^{e,-} = \neg\exists \left(\text{t1:t} \xrightarrow{e} \begin{array}{c} 1:t' \\ \vdots \\ j+1:t' \end{array} \right),$$

wobei $c_t^{e,+} = \top$, falls $i = 0$, $c_t^{e,-} = \top$, falls $j = *$ und $src_{TGI}(e) = t$ und $tar_{TGI}(e) = t'$.

UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Jens Kosiol

Graph- and Model-Driven Engineering

Übungsblatt 4 – Lösungsskizze

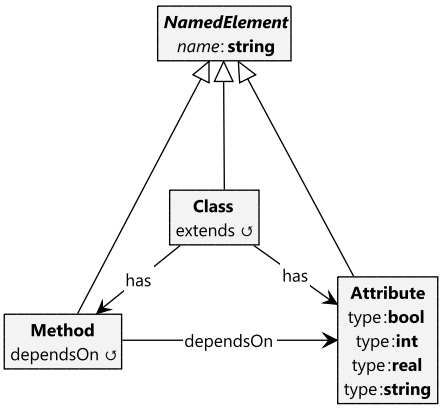

Abbildung 1: Typgraph für vereinfachte Klassendiagramme

2 Entwurf und Auswertung von Graphformeln (8 Punkte)

Gegeben sei der Typgraph für Klassendiagramme von Aufgabenblatt 2 (Sie dürfen Ihre eigene Lösung verwenden oder die in Abbildung 1 gezeigte).

Aufgaben:

a) Formalisieren Sie die folgenden Aussagen als (über dem gewählten Typgraphen getypte) geschachtelte Graphformeln. Sie dürfen die Formeln in Groove erstellen oder wie in der Vorlesung eingeführt zeichnen; die besprochenen Vereinfachungen in der Darstellung dürfen Sie dabei benutzen.

   i) Wenn eine Klasse ein Attribut besitzt, besitzt sie auch eine Methode, die dieses Attribut verwendet. (2 Punkte)

   ii) Es gibt eine Klasse `c1` mit Methode `m1` sodass jede weitere Klasse `c2` ein Attribut besitzt, auf das `m1` zugreift. (2 Punkte)

b) Geben Sie für die Formel ii) von oben jeweils einen (über dem gewählten Typgraphen getypten) Graphen an, der die Formel erfüllt, und einen, der dies nicht tut. Begründen Sie jeweils, warum dies der Fall ist (unter Verweis auf die Semantik Ihrer erstellten Graphformel, nicht der natürlichsprachlichen Bedeutung). (4 Punkte)

UNIKASSEL VERSITÄT | ELEKTROTECHNIK INFORMATIK

Jens Kosiol

Graph- and Model-Driven Engineering

Übungsblatt 4 – Lösungsskizze

Lösung

i) Die Graphformeln sind in den Abbildungen 2 und 3 zu sehen.

ii) Beispielgraphen befinden sich in der Groove-Datei Ueb03_KlassendiagrammeFormeln.

Erklärung: Die Formel hat die Struktur $c = \exists(C_1, d)$, wobei d die Struktur $\forall(C_1 \to C_2, \exists C_3)$ hat. Per Definition erfüllt ein Graph G die Formel c, wenn es eine Abbildung q von C_1 nach G gibt, sodass die Abbildung q die Bedingung d erfüllt. Nach Vorlesung sind Bedingungen der Form d erfüllt, wenn jede Abbildung q' von C_2 nach G, die die Abbildung q erweitert, auch zu einer Abbildung von C_3 nach G erweitert werden kann (oder es keine Abbildung q' gibt, die q erweitert).

Folglich erfüllt ein Graph G eine Formel der Struktur von c *nicht*, wenn es keine Abbildung q von C_1 nach G gibt oder für keine solche Abbildung q gilt, dass jede Erweiterung zu einer Abbildung q' von C_2 nach G sich wiederum zu einer Abbildung von C_3 nach G erweitern lässt.

Im Beispielgraph Ueb4-SatisfyingGraph kann die Abbildung q von C_1 dorthin definiert werden, indem Klasse und Methode aus der Formel auf die Klasse c1 bzw. Methode m1 abgebildet werden (hier und im Folgenden wird die Abbildung jeweils auf den Knoten spezifiziert und die Abbildung der Kanten ergibt sich eindeutig daraus). Mögliche Erweiterungen auf C_2 sind in diesem Beispiel die Abbildungen q'_1, die die Klasse c2 aus der Formel auf die Klasse c2 aus dem Graphen abbildet, und q'_2, die die Klasse c2 aus der Formel auf die Klasse c3 aus dem Graphen abbildet. Die Abbildung q'_1 lässt sich auf C_3 erweitern, indem das Attribut a auf das Attribut a1 aus dem Graphen abgebildet wird; die Abbildung q'_2, indem das Attribut a auf das Attribut a3 aus dem Graphen abgebildet wird. Damit erfüllt der Graph Ueb4-SatisfyingGraph die Formel.

Der Beispielgraph Ueb4-non-SatisfyingGraph erfüllt die Formel nicht. Es gibt zwei Möglichkeiten, einen Morphismus q vom Graphen C_1 aus dorthin zu definieren (auf die Klassen c1 bzw. c2, da diese Klassen Methoden haben). Beide Möglichkeiten lassen sich jeweils zu einem Morphismus q' von C_2 aus erweitern, indem man die Klasse c2 auf die Klasse c4 im Graph abbildet. Diese Morphismen lassen sich dann aber nicht auf C_3 erweitern, da die Klasse c4 kein Attribut besitzt.

Jens Kosiol

Graph- and Model-Driven Engineering
Übungsblatt 4 – Lösungsskizze

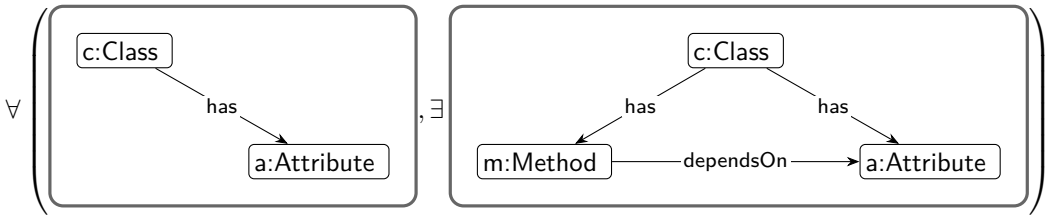

Abbildung 2: Graphformel, die ausdrückt, dass Attribute auch in der Klasse genutzt werden, in der sie enthalten sind

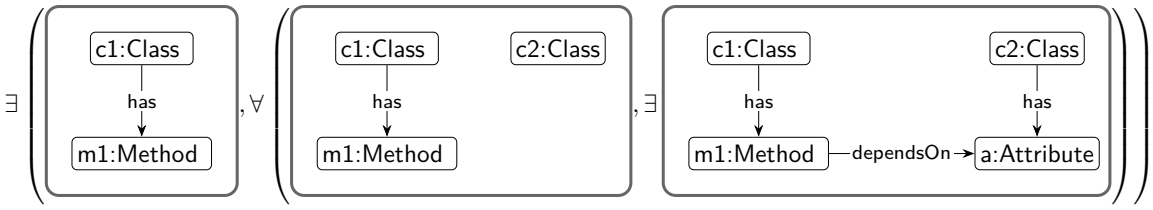

Abbildung 3: Graphformel, die ausdrückt, dass es eine Methode mit Zugriff auf ein Attribut jeder Klasse gibt

3 Berechnung von Anwendungsbedingungen (12 Punkte)

In dieser Aufgabe verwenden wir wieder die Ontologie für den Verkauf von Medien von Übungsblatt 2. Abbildung 4 zeigt eine Graphformel für diese Anwendungsdomäne.

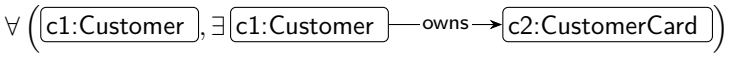

Abbildung 4: Graphformel zur Existenz von Bonuskarten

a) Erklären sie natürlichsprachlich, was die Graphformel aus Abbildung 4 bedeutet. (2 Punkte)

b) Berechnen Sie für diese Graphformel und die Regel *deleteCard_R* von Übungsblatt 3 die garantierende Anwendungsbedingung. Ihre Lösung soll die einzelnen Rechenschritte zeigen. (6 Punkte)

c) Angenommen Sie wissen, dass Sie Ihre Regel(n) nur auf valide Graphen anwenden werden, also solche, die die Graphformel erfüllen. Sie wollen sicherstellen, dass berechnete Ergebnisse (Graphen nach Regelanwendung) weiterhin valide sind.

i) Wie können Sie die oben berechnete garantierende Anwendungsbedingung in diesem Fall vereinfachen? Begründen Sie, warum Ihre Anwendungsbedingung ausreicht und welche Vorteile sie hat. (2 Punkte)

ii) Würden Sie in diesem Szenario die Regeln *useCustomerCard_R* und *order_and_create_CustomerCard* mit Anwendungsbedingungen versehen? Warum (nicht)? (2 Punkte)

Lösung

a) Jeder Kunde besitzt eine Kundenkarte.

b) Die entstehende Anwendungsbedingung hat die Struktur $\forall\Big(L \to N_1, \exists(N_1 \to N_1^1)\Big) \wedge \forall\Big(L \to N_2, \exists(N_2 \to N_2^1)\Big)$; die Berechnung und konkreten Graphen gibt es in einer extra Datei in der Dropbox (Ueb04_Aufg3b-Computation-Guaranteeing-AC.pdf).

c) Anwendung auf valide Graphen:

i) Wir können die Anwendungsbedingung in diesem Fall zu $\forall\Big(L \to N_2, \exists(N_2 \to N_2^1)\Big)$ vereinfachen (sogar zu $\exists(L \to N_2^1)$, da $L \cong N_2$). Der andere Teil der garantierenden Anwendungsbedingung, $\forall\Big(L \to N_1, \exists(N_1 \to N_1^1)\Big)$, bedeutet zu überprüfen, ob auch für jeden Customer außerhalb des gewählten Regelansatzes eine CustomerCard existiert; in diesem Szenario kann aber als bekannt vorausgesetzt werden, dass das der Fall ist. Die Regelanwendung (Suche nach einem Ansatz) kann also deutlich beschleunigt werden, indem man diesen Teil der Anwendungsbedingung weglässt.

ii) Nein; da keine der beiden Regeln einen Customer erstellt oder eine CustomerCard löscht, kann ihre Anwendung nichts an der Validität eines Graphen bezüglich der Graphformeln ändern.

Jens Kosiol

Graph- and Model-Driven Engineering
Übungsblatt 4 – Lösungsskizze

4 Definition von domänenspezifischen Modellierungssprachen (12 Punkte)

Entwerfen Sie in Analogie zum Vorgehen in der Vorlesung eine domänenspezifische Modellierungssprache für GUIs. Gehen Sie in den folgenden Schritten vor:

a) Geben Sie ein Metamodell (mit Kanten-Multiplizitäten) an, das die abstrakte Syntax definiert. (2 Punkte)

b) Formalisieren Sie (mindestens zwei) Constraints, die für die Sprache gelten sollen, aber nicht durch das Metamodell ausdrückbar sind, als geschachtelte Graphformeln (soweit möglich). (2 Punkte)

c) Entwickeln Sie eine Grammatik (Startgraph und Regeln), die es erlaubt, solche GUIs zu generieren, die keines der geforderten Constraints verletzen. Außerdem sollen Instanzen von jedem Element des Metamodells auch erzeugbar sein. Dokumentieren Sie explizit (z. B. durch Kommentarknoten in Groove), wie Sie dafür sorgen, dass Ihre Regeln die geforderten Constraints nicht verletzen. (6 Punkte)

d) Entwerfen Sie zwei Graphtransformationsregeln, die Teile der intendierten Semantik modellieren (z. B. einen Szenenwechsel, das Einblenden von Elementen o. Ä.). Diese Regeln dürfen über einem im Vergleich zu i) erweiterten Metamodell getypt sein. (2 Punkte)

Orientieren Sie sich für ihre Modellierungssprache an der Syntax, Struktur (und Semantik) von JavaFX (vereinfacht): Eine Stage zeigt eine Scene an, deren Inhalt durch einen Baum gegeben wird (der Scene Graph). Die Knoten des Baums sind die (visuellen) Komponenten aus denen die Oberfläche besteht. Es gibt (u. a.) Controls (Button, TextField, ...), um Funktionalität zur Verfügung zu stellen, und Layouts (Group, HBox, ...), um zu steuern, welches Element wo erscheint. Ihr Metamodell soll mindestens 6 Komponenten enthalten, darunter Kontroll- und Layoutkomponenten. Versehen Sie Ihre Komponenten auch mit (einer Auswahl an) sinnvollen Attributen.

Lösung

Eine (sehr rudimentäre) Lösung gibt es in der Datei Ueb04_GUIs.gps.