Anleitung zum Zeichnen von MOF-Metamodellen für das Netbeans-MDR mit MagicDraw

Robert Lechner

16. April 2004

1 Einleitung

Die Werkzeuge des *Netbeans Metadata Repository* (MDR[1]) ermöglichen es, ein MOF-Metamodell direkt in Java-Code zu übersetzen (siehe *Java Metadata Interface* JMI). Dabei sind folgende Schritte auszuführen:

1. Zeichnen des Metamodells (z.B. mit MagicDraw) als UML-Klassendiagramm und speichern als XMI-Datei.
2. Konvertieren des UML-Diagramms in ein MOF-Modell.
3. Erzeugen der Interfaces aus dem Modell.
4. Verwenden der Interfaces mit der Netbeans-MDR-Library.

Die zur Ausführung der Tools, sowie des generierten Codes benötigten Libraries wurden von *Stefan Mitterdorfer* [2] aus dem Netbeans-Code extrahiert und stehen als 1,4 MB große JAR-Datei zur Verfügung.

2 Bedienung

Damit das Erstellen von MOF-Modellen mit MagicDraw funktioniert, ist folgendes zu beachten:

- Als Vorlage dient die Datei `MagicDraw60.xml.zip` von der MDR-Homepage. Dort ist bereits ein Modell voreingestellt. Das MOF-Modell muss dort als Klassendiagramm gezeichnet werden.

[1] `http://mdr.netbeans.org`
[2] `Stefan.Mitterdorfer@salomon.at`

- Der Name des Modells ist sehr wichtig, weil er für zwei Dinge benötigt wird:
  1. Es wird ein Package für die Interfaces angelegt. Der Name des Package ist der Modellname in Kleinbuchstaben (ohne Leerzeichen).
  2. Im Package gibt es ein Interface, das als Proxy für die eigentlichen Element des Modells dient. Der Name wird ebenfalls aus dem Modellnamen erzeugt.
- Es ist nicht möglich die Interfaces in ein beliebiges Package zu geben, weil Packages im UML-Diagramm ignoriert werden und Modellnamen wie etwa `ccmtools.Modell.OCL` zu Fehlern bei der Code-Erzeugung führen.
- Als Datentypen sind nur die vordefinierten Typen (String, Integer, etc.) oder die Klassen des Modells zulässig.
- Obwohl es laut MOF- und JMI-Spezifikation möglich ist, eigene Methoden zu den Klassen hinzuzufügen, sollte davon Abstand genommen werden. Man verliert sonst den Vorteil, dass die Implementation der Klassen von der MDR-Library automatisch durchgeführt wird.
- Für Attribute mit der Vielfachheit **1** oder **0..1** werden in der Klasse Methoden zum Lesen und Schreiben definiert.
- Für Attribute mit einer anderen Vielfachheit gibt es nur eine Methode zum Lesen der ganzen Collection. Alle anderen Operationen müssen über Assoziationen erfolgen.
- Eine Collection von Basistypen (Integer, ...) ist nicht möglich, weil eine Assoziation mit Basistypen zur Laufzeit zu einem Fehler führt. Basistypen sind daher nur mit der Vielfachheit **1** und **0..1** möglich.
- Attribute die nicht von einem Basistyp sind, müssen immer über eine Assoziation definiert werden (auch bei Vielfachheit **1** oder **0..1**).
- MagicDraw speichert bei Vielfachheiten wie etwa **0..*** das Zeichen * nicht als **-1** ab. Das führt zu einem Fehler im Programm `uml2mof`. Abhilfe: entweder in der XMI-Datei * durch **-1** ersetzen oder im MagicDraw direkt anstelle von * den Wert **-1** eingeben.
- Für jede Assoziation wird ein Interface erzeugt ⇒ alle Assoziationen müssen eindeutige Namen haben. Es ist nicht erlaubt, keinen Namen zu vergeben.
- Beide Enden einer Assoziation müssen einen Namen haben. Das gilt auch für Assoziationen die nur in eine Richtung gehen.

Ein einfaches Beispiel ist in Abbildung 1 zu sehen:
Die Beziehung zwischen `Tree` und `Leaf` geht in beide Richtungen, d.h. auf beiden Seiten gibt es ein sichtbares Ende mit einer Vielfachheit.

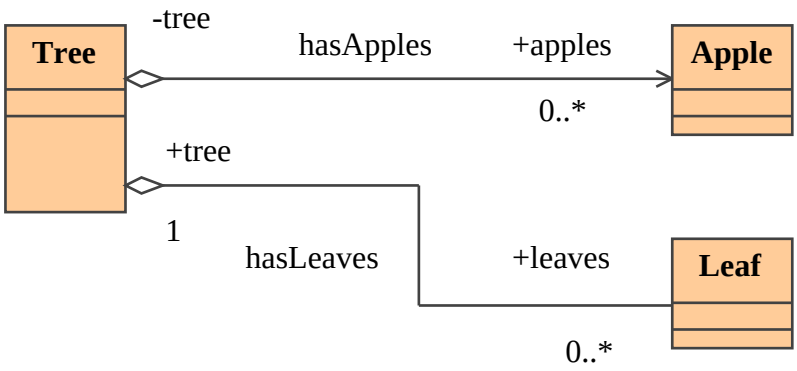

Abbildung 1: „Apfelbaum“-Modell

Die Beziehung zwischen `Tree` und `Apple` geht nur in eine Richtung. Das Assoziationsende „tree“ ist nicht sichtbar und benötigt auch keine Vielfachheit.

3 Probleme

1. Das Java-Package (für den generierten Code) ist nicht frei wählbar.
2. Es gibt als Datentyp nur die vordefinierten Basistypen und die Klassen des Modells.
3. Attribute, deren Typ eine Klasse des Modells sind, müssen immer über Assoziationen definiert werden → viele „Linien“ am Papier, viel unnötiger Code.
4. Attribute, deren Type ein Basistyp ist, können nur mit den Vielfachheiten **1** oder **0..1** definiert werden.
5. Der generierte Code ist ohne MDR-Library nicht lauffähig.
6. Eines der Hauptvorteile von Modellen ist die Möglichkeit, die Implementation auszutauschen. Das ist bei Verwendung des Netbeans-MDR nur schwer möglich (siehe Abschnitt 4).
7. Es kommt häufig vor, dass es bei der Verwendung des MDR zu nichtdeterministischen Fehlern kommt. Abhilfe: die Repository-Dateien werden vor jedem Programmstart gelöscht. Das ist entweder ein Fehler der MDR-Library oder deren Verwendung. Es ist auch sehr eigenartig, dass die Größe diese Dateien (`mdr.*`) nicht konstant ist (für ein bestimmtes Modell), sondern von der Anzahl der Programmstarts bzw. der Laufzeit abhängt.

4 Erweiterbarkeit

4.1 optimale Lösung

Eine optimale Lösung hat folgende Struktur:

- Für jede Klasse und jede Assoziation des Modells wird ein Interface generiert.
- Zum Erzeugen der Klassen und Assoziationen wird jeweils eine Factory erzeugt.
- Weiters wird ein Proxy generiert, der den Zugriff auf die Factories ermöglicht.
- Zu jedem Interface wird eine entsprechende Implementation (Klasse) erzeugt.

Um die gesamte Implementation zu ersetzen, werden einfach die Interfaces neu implementiert. Im Programm muss nur der neue Proxy instanziert werden.
Um eine einzelne Klasse des Modells neu zu implementieren, wird wie folgt vorgegangen:

1. Die Klasse wird neu implementiert (z.B. durch Ableiten von der erzeugten Klasse).
2. Die entsprechende Factory wird ebenfalls neu geschrieben.
3. Vom bestehenden Proxy wird eine neue Version abgeleitet, die nur den Zugriff auf diese Factory überschreibt.
4. Im Programm wird der neue Proxy instanziert.

4.2 Netbeans Metadata Repository

Der vom Netbeans-MDR erzeugte Code hat diese Struktur:

- Für jede Klasse und jede Assoziation des Modells wird ein Interface generiert. Diese Interfaces implementieren aber zusätzlich Interfaces der MDR-Library (Java-Package `javax.jmi.reflect`).
- Das gleiche gilt für die Interfaces der Factories und des Proxy.
- Die Implementation erfolgt zur Laufzeit über das Repository. Es gibt keinen Sourcecode der Implementation.

Beim Ersetzen oder Erweitern der Implementation treten folgende Probleme auf:

1. Das Erweitern der Funktionalität durch Ableiten ist nicht möglich, da es nur die Interfaces als Sourcecode gibt.
2. Die neue Implementation muss auch die zusätzlichen Methoden der JMI-Interfaces implementieren.
3. Beim Proxy kann ebenfalls nicht mit Vererbung gearbeitet werden. Im neuen Proxy muss also auch der Zugriff auf die bestehenden Factories implementiert werden. Das ist aber nur möglich, indem man den „alten“ Proxy instanziert und alle Methoden (ausgenommen der Zugriff auf die neue Factory) an ihn weiterleitet.