Effiziente Modellierung durch Automatisierung

Andreas Metzger

Fachbereich Informatik, TU Kaiserslautern, Postfach 3049, 67653 Kaiserslautern
metzger@informatik.uni-kl.de

Trotz des hohen Abstraktionsgrads, der durch objektorientierte Softwaremodelle erreicht wird, gibt es immer noch eine Vielzahl von zeitaufwändigen und fehleranfälligen Modellierungsaktivitäten. Heutzutage lassen sich jedoch auch solche Aktivitäten effizient automatisieren. Der Fokus liegt dabei allerdings häufig noch auf der Abbildung von Modellen auf Code, auch wenn dies, wie z.B. im Falle der MDA, mit Hilfe intermediärer Modelle geschieht. Sicherlich besitzen aber auch die eigentlichen Modellierungsaktivitäten, also Abbildungen zwischen Softwaremodellen wie z.B. Architekturänderungen oder Wiederverwendung von Modellelementen, ein ebensolches Automatisierungspotenzial.

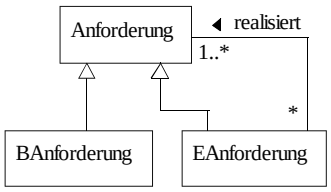
Um eine Automatisierung solcher Aktivitäten überhaupt realisieren zu können, müssen zunächst Kenntnisse über die Softwareentwicklung in *expliziten* Modellen beschrieben werden, welche die betrachteten Entwicklungsartefakte (Produktmodell) und die zu automatisierenden Aktivitäten (Prozessmodell) beschreiben. Der Zugriff auf und die Modifikation von Entwicklungsartefakten kann softwaremäßig durch die Verwendung von objektorientierten Metamodellen realisiert werden [At97], welche den Vorteil der Abstraktion von der konkreten Syntax der Softwaremodelle bieten. Insbesondere mit Technologien wie der UML oder der MOF [Ge02] steht hier eine solide Basis zur Verfügung. Die Entwicklungsaktivitäten im Prozessmodell müssen nun so spezifiziert werden, dass sie in eine ausführbare Form gebracht werden können. Da jedes Softwaremodell eine Instanz des zugehörigen Produktmodells darstellt, lässt sich dies durch die Beschreibung der Transformation von Produktmodellinstanzen realisieren, was prinzipiell auf zwei Arten erfolgen kann: beim *deklarativen* Ansatz werden Transformationen als Regeln beschrieben, beim *imperativen* Ansatz werden die Transformationsaktivitäten operational spezifiziert. In der folgenden Tabelle werden diese beiden Ansätze verglichen.

Aspekt	**deklarativer Ansatz**	**imperativer Ansatz**
Realisierungs-technik (Beispiele)	Graphtransformation [An99], Spezifikation log. Ausdrücke (OCL), Transformations-sprache (XSLT) [Ge02]	Aktionssprache (UML/ActionSemantics) [Su01], Programmiersprache (Java, C++), Skriptsprache [Po02]
Abstraktion	realisierungsunabhängig: abstrakte/unvollständige Transformationen möglich [Be03]	realisierungsabhängig, evtl. implementierungsunabhängig (Aktionssprache)
Ausführbarkeit	Operationalisierung der Regeln notwendig	direkt möglich
Mächtigkeit	Lokalität von Graphtransformationsregeln kann einschränken [An99]	komplexe Änderungen, Rekursion und temporäre Datenstrukturen realisierbar (s.u.)
Vorhersagbar-keit	bei Graphtransf. Nicht-Determinismen und Nicht-Terminierung möglich [An99][Ge02]	i.d.R. deterministisch
Effizienz	Wahl eines effizienten Algorithmus aus Regeln oft nicht garantiert [Me99]	Einsatz bekannter (effizienter) Algorithmen möglich

Wie man erkennt, können die beiden Ansätze auf Grund ihrer Eigenschaften für die jeweiligen Anwendungen besser oder schlechter geeignet sein. Nach unserer Erfahrung hat

sich der imperative Ansatz besonders für die Automatisierung komplexer Analyseaufgaben mit temporären Datenstrukturen [Me04], und für Modelltransformationen mit globalen Abhängigkeiten bewährt. Insbesondere da wir unser Produktmodell auf eine moderne Programmiersprache (Java) abbildeten (was z.B. mittels JMI [Di02] möglich ist) stand uns eine mächtige Sprache zur Realisierung der Operationen zur Verfügung. Man verliert gegenüber Aktionssprachen [Su01] zwar die Implementierungsunabhängigkeit, kann dafür aber auf viele existierende Komponenten und Werkzeuge zurückgreifen.

Als Beispiel einer automatisierten Aktivität wollen wir hier die Berechnung eines einfachen Komplexitätsmaßes für Anforderungen vorstellen, welche durch das Produktmodell in Abb. 1 beschrieben werden. Benutzeranforderungen werden dabei durch Entwickleranforderungen realisiert, die ihrerseits wieder zu Sub-Anforderungen führen können. Die Komplexität `k` einer Anforderung ergibt sich aus der Summe der jeweils mit der Tiefe `t` der „realisiert"-Ketten gewichteten Anzahl von Sub-Anforderungen (`t` beginnt bei 1). In Abb. 1 ist der Code der `berechneK()`-Methode der Klasse `Anforderung` gezeigt.




```
public int berechneK(int t) {
  int k = t;
  EAnforderung e;
  Iterator it = getEAnforderung().iterator()
  while(it.hasNext()) {
    e = (EAnforderung)it.next();
    k += e.berechneK(t+1); }
  return k; }
```

Abb. 1: Produktmodell und imperative Beschreibung einer Modellierungsaktivität

Besonders interessant kann die Verwendung von Java werden, wenn man deren reflexiven Fähigkeiten einsetzt, um generische Transformationen zu beschreiben. So könnte man ein allgemeines Komplexitätsmaß, welches für ein beliebiges Modellelement die Anzahl der Links zu anderen Elementen zählt, mit nur *einer* generischen Methode realisieren.

Literaturverzeichnis

[An99] Andries, M.; Engels, G.; Habel, A. et al.: Graph Transformation for Specification and Programming. Science of Computer Programming, 34(1), 1999; S. 1–54

[At97] Atkinson, C: Meta-Modeling for Distributed Object Environments. In: Proceedings EDOC '97, IEEE Computer Society, 1997; S. 90–103.

[Be03] Bezivin, J.; Farcet, N.; Jezequel, J.-M. et al.: Reflective Model Driven Engineering. In: Proceedings UML 2003, LNCS 2863, 2003; S. 175–189

[Di02] Dirczke, R: Java Metadata Interface (JMI) Specification, Version 1.0, 2002

[Ge02] Gerber, A.; Lawley, M.; Raymond, K. et al.: Transformation: The Missing Link of MDA. In: Proceedings ICGT 2002, LNCS 2505, 2002; S. 90–105.

[Me99] Mellor, S.J.; Tockey, S. et al.: An Action Language for UML: Proposal for a Precise Execution Semantics. In: Proceedings UML '98, LNCS 1618, 1999; S. 307–318

[Me04] Metzger, A: Feature Interactions in Embedded Control Systems. Erscheint in: Computer Networks, Special Issue on Feature Interactions in Emerging Application Domains, 2004

[Po02] Porres, I: A Toolkit for Manipulating UML Models, Technical Report No. 441, Turku Center for Computer Science, Turku, Finnland, 2002

[Su01] Sunye, G.; Pennaneac'h, F. et al.: Using UML Action Semantics for Executable Modeling and Beyond. In: Proceedings CAiSE 2001, LNCS 2068, 2001; S. 433–447