Adaptionstechniken für GMF-basierte Modellierungswerkzeuge

Jörg Hartmann, Heiko Kern, Stefan Kühne
Universität Leipzig, Institut für Informatik, Betriebliche Informationssysteme
Johannisgasse 26, 04103 Leipzig
{jhartmann, kern, kuehne}@informatik.uni-leipzig.de

Abstract: Editoren, die auf Basis des Graphical Modeling Framework erstellt wurden, besitzen zunächst einen durch die Plattform begrenzten Funktionsumfang. Die Erweiterung um individuelle Funktionalität erfordert die Adaption des generierten Codes, was zu einer Vermischung von individuellem und generiertem Code führt. Dies führt bei einem Plattformwechsel auf eine neue GMF-Version und der Neugenerierung aus vorhandenen GMF-Modellen zu teils erheblichen Adaptionsaufwänden. In diesem Beitrag werden verschiedene Adaptionstechniken beschrieben und hinsichtlich des resultierenden Adaptionsaufwands anhand einer Fallstudie bewertet. Die Ergebnisse können als Hilfestellung zur Wahl einer geeigneten Adaptionstechnik für analoge Problemstellungen verwendet werden.

1 Einleitung

Das Graphical Modeling Framework (GMF) ist ein Framework innerhalb des Eclipse Modeling Project [Gro09] zur Erstellung von Editoren für grafische Modellierungssprachen. Die Entwicklung basiert dabei auf dem Prinzip des Model-Driven Software Development [SVEH07]. Ausgehend von Modellen wird durch Generatoren ein entsprechendes Modellierungswerkzeug auf Basis der Eclipse-Plattform erzeugt.

Die generierten Editoren implementieren eine von GMF festgelegte Standardfunktionalität. Oft entspricht das Generat nicht den Nutzeranforderungen. Somit muss der generierte Code adaptiert werden, was zu einer Mischform aus generiertem und individuellem Code führt. Die zunehmende Vermischung führt zu schwer beherrschbaren Redundanzen und verletzt die Modell-Code-Synchronität. Bei der Migrierung des Editors auf eine neue GMF-Version ist dadurch ein erheblicher Adaptionsaufwand zu berücksichtigen. Ausgehend von einer erneuten Generierung des Editors aus vorhandenen GMF-Modellen, sind die ehemals implementierten Erweiterungen einzupflegen.

Der Aufwand zur erneuten Adaption ist unterschiedlich und hängt u. a. von der verwendeten Adaptionstechnik ab. Ziel dieser Arbeit ist es, verschiedene Adaptionstechniken zu beschreiben und eine Bewertung bzgl. ihres Aufwands vorzunehmen. Das Ergebnis der Untersuchung soll Hilfestellung bei der Wahl einer geeigneten Adaptionstechnik geben, um den Aufwand einer Editor-Entwicklung über mehrere GMF-Versionen möglichst gering zu halten.

2 Adaptionstechniken

2.1 Direkte Adaption

Ein von GMF generierter Editor kann durch direkte Adaption am generierten Code angepasst werden. Dazu werden entsprechende Klassen oder Methoden im generierten Code ergänzt oder bestehende aus dem generierten Code adaptiert. Der individuelle Code kann zwar durch Protected Regions [SVEH07] geschützt werden, allerdings werden die Protected Regions des alten Editors bei einer vollständigen Neugenerierung aus den alten Modellen auf die neue Plattform nicht berücksichtigt. Dadurch ist der Aufwand zur Übernahme der Protected Regions in den neuen Editor gleich dem Aufwand zur direkten Adaption.

2.2 Adaption am Generator

GMF-Editoren werden durch einen Template-basierten Ansatz mit XPand von openArchitectureWare[1] generiert. Templates dienen als eine Art Schablone für den generierten Code. Sie können zur Adaption herangezogen werden, indem individueller Code direkt in den Templates ergänzt und so während der Generierung automatisiert in jeder Ausprägung der Templates hinzugefügt wird. Bei einer Neugenerierung mit Plattformwechsel müssen die Anpassungen an den Templates der alten Plattform in die Templates der neuen Plattform migriert werden.

2.3 Adaption durch Code-Seperation

Code-Seperation beschreibt die explizite, physiche Trennung des individuellen vom generierten Code, wodurch die Wiederverwendung erleichtert werden soll. Nachfolgend werden zwei Ansätze zum Code-Seperation beschrieben.

Objektorientierte Adaption Die Vererbung stellt eines der grundlegensten Konzepte der objektorientierten Programmierung dar und wird von GMF genutzt, um schrittweise die Grundfunktionalität eines Editors über eine Vererbungshierachie aufzubauen. Zur Adaption wird diese Hierarchie um eine Klasse erweitert, die den individuellen Code kapselt. Diese Klasse dient dabei als Bindeglied zwischen Plattform und Generat und wird zur späteren Adaption der neuen Plattform wiederverwendet. Bei einem Plattformwechsel ist so nur die Vererbungshierarchie des neuen Editors zu adaptieren.

Aspektorientierte Adaption Bei diesem Ansatz wird die Objektorientierung über Aspekte erweitert [Böh06], die als Behälter für Advices dienen und mit anderen Code-Abschnitten

[1] http://www.openarchitectureware.org

über Joinpoints interagieren [GV09]. Eine Umsetzung dieses Ansatzes in Java ist Object Teams [HM07], welches die Erweiterung jeder Klasse innerhalb von Eclipse Plugins ermöglicht. Zur Adaption werden Advices unsichtbar für den generierten Code in seperaten Plugins eingeführt. Advices selbst sind Methoden, die über Joinpoints an generierte Methoden gebunden werden und beim Erreichen des Joinpoints während der Programmausführung aufgerufen werden. Durch die Einführung der Advices bleibt der generierte Code stets unberührt, weswegen Auswirkungen durch Neugenerierungen vermieden werden können. Nach einem Plattformwechsel werden Advices und Joinpoints wiederverwendet, da deren Plugin nur hinzugefügt werden muss.

3 Bewertung der Adaptionstechniken

3.1 Vorgehen und Anwendungsfälle

Die Bewertung der Adaptionstechniken wird anhand der bflow* Toolbox[2] mit zwei Anwendungsfällen durchgeführt. Die bflow* Toolbox ist ein GMF-basierter Editor zur Geschäftsprozessmodellierung, der u. a. objektorientierte und erweiterte Ereignisgesteuerte Prozessketten unterstützt. In den beiden Anwendungsfällen wird zunächst die bflow* Toolbox auf der alten Plattform[3] mit der jeweiligen Adaptionstechnik angepasst, wobei der Aufwand für das Hinzufügen der individuellen Funktionen festgehalten wird. Nach dem Wechsel auf eine neue GMF-Version[4] müssen die individuellen Funktionen in die neu generierte bflow* Toolbox übernommen werden. Der entsprechende Adaptionsaufwand hierfür wird ebenfalls festgehalten. Zu beachten ist, dass die Aktualisierung der Plattform eine bereits vorgenommene Adaption obsolet machen könnte. Daher ist die Einschränkung offen zu legen, wonach nachfolgend vorgestellte Anwendungsfälle auf beiden Plattformen analog zu implementieren sind.

Prinzipiell können zwei Adaptionsformen unterschieden werden: (1) Adaptionen, die unabhängig von der GMF-Klassenhierarchie implementiert werden und (2) Adaptionen, die in Abhängigkeit der Klassenhierarchie durch Überschreiben bereits im Framework implementierter Methoden realisiert werden. Zur Bewertung wird nachfolgend korrespondierend zu den Adaptionsformen jeweils ein Anwendungsfall beschrieben, die aus Kundenanforderungen der bflow* Toolbox hervorgehen.

Celleditor Der Celleditor regelt die Größe des Labels innerhalb eines Zeichenelements, wenn der Text des Labels bearbeitet wird. Im standardmäßig von GMF erzeugten Celleditor (implementiert durch `DirectEditManager`) wird beim Bearbeiten eines Textes lediglich nur eine Zeile angezeigt. Bei längerem Text muss so umständlich über Pfeiltasten im Label navigiert werden. Ziel ist, den gesamten Text anzuzeigen, um so das Editieren zu erleichtern.

[2] http://www.bflow.org
[3] Eclipse Ganymede Modeling Tools, GMF 2.1.3
[4] Eclipse Galileo Modeling Tools, GMF 2.2.0

Textalignment Das Textalignment steht für die Ausrichtung des Textes innerhalb eines Labels. Es wird zwischen den Ausrichtungen „left“, „right“ und „center“ unterschieden. Im Standard-Editor kann der Modellierer die Ausrichtung auswählen und persistieren. Sie wird aber nicht an das Label geleitet, wodurch der Text stets linksbündig dargestellt wird. Das Ziel besteht darin, das Textalignment korrekt an das Label zu leiten und abzubilden.

3.2 Definition der Bewertungskriterien

Zur Bewertung werden die folgenden Bewertungskriterien herangezogen.

- Die **Anzahl der Adaptionsstellen** ist die Anzahl (`a`) an Stellen, die im generierten oder individuellen Code angepasst wurden.
- Die **Anzahl der adaptierten Artefakte** (`A`) gibt an, wie viele Dokumente (Java-Dateien, Templates, etc.) zur Adaption bearbeitet wurden.
- **Lines of Code** (`LoC`) gibt an, wie viele Quellcode-Zeilen für die Adaption benötigt wurden.

Die Bewertung der Adaptionstechniken erfolgt am intra-individuellen Vergleich der vorgestellten Kriterien pro Plattform. Demnach verringert eine Technik den Adaptionsaufwand zur Migration, wenn die festgehaltenen Werte im Vergleich zum initialen Aufwand sinken. Ziel einer Technik ist es, den Migrierungsaufwand auf Null zu senken. Bleiben die Werte dagegen konstant, konnte die Technik den Adaptionsaufwand nicht verringern. Um den kumulierten Adaptionsaufwand möglichst gering zu halten, ist weiterhin ein niedriger initialer Aufwand wünschenswert. Weitere Arbeiten, die zur Aktualisierung der Plattform nötig sind, können vernachlässigt werden, da sie im gleichen Maß bei allen Techniken auftreten.

3.3 Bewertung der Adaptionstechniken

Im Kapitel 3.1 wurde bereits das allgemeine Vorgehen zur Evaluation der Bewertungstechniken beschrieben. Die gemessenen Aufwände zur Adaption der initialen Plattform P_i und der migrierten Plattform P_m werden für beide Anwendungsfälle in Tabelle 1 zusammengefasst. Nachfolgend werden die erzielten Adaptionsaufwände der Anwendungsfälle genauer betrachtet.

Celleditor Zur Umsetzung mit Hilfe der direkten Adaption muss die Methode `DirectEditManager getManager()` jeder der 23 Realisierungen von `ITextAwareEditPart` adaptiert werden, indem beim Methodenaufruf des entsprechenden Setters `WrapTextCellEditor.class` als zweiter Parameter übergeben wird. Unter der migrierten Plattform ist die Adaption erneut vorzunehmen.

Beim Template-basierten Ansatz ist das Template `TextAware.xpt` zu adaptieren, wobei der DEFINE-Block `getManager` analog zur obigen Technik adaptiert wird.

Bei der Adaption durch Objektorientierung wird eine individuelle Klasse erzeugt. Zu beachten ist, dass der `DirectEditManager` in den generierten Klassen attributiert ist, weshalb er als `protected` in der individuellen Klasse hinzugefügt wird. In den generierten Klassen ist das Attribut samt Methode zu löschen, um eine Überschreibung zu verhindern, was auch nach der Neugenerierung zu beachten ist.

Zur aspektorientierten Adaption wird ein neues Eclipse-Plugin eingeführt, in dem mehrere Advices mit Joinpoints entsprechend der Object Teams Spezifikation angelegt werden. Dadurch kann der individuelle Code nach der Neugenerierung wiederverwendet werden.

Die Ergebnisse (siehe Tabelle 1) zeigen, dass direkte und objektorientierte Adaption den Adaptionsaufwand nicht entscheidend senken konnten. Nachteilig für die Objektorientierung ist außerdem, dass nach jeder Neugenerierung die Vererbungshierarchie adaptiert werden muss. Im gemessenen Aufwand wird dies durch `A` und `Loc` deutlich. Auffällig war die Reduzierung des Migrationsaufwandes auf Null durch die aspektorientierte Adaption, weshalb diese effizient erscheint. Trotzdem ist sie aufgrund des hohen initialen Aufwands abzulehnen. Favorisiert wird die Adaption am Template. Bei dieser blieben die Werte zwar konstant, allerdings ist der allgemeine Aufwand verschwindend klein.

Textalignment Der Anwendungsfall wird über direkte Adaption der 23 Realisierungen von `ShapeNodeEditPart` implementiert, wobei die Methoden `refreshVisuals()` und `handleNotification(Notification)` überschrieben werden. Weiter werden vier Hilfsmethoden hinzugefügt, die die gesetzte Textausrichtung abfangen und an das Label leiten.

Beim Template-basierten Ansatz wird ein GMF-unabhängiges Template erzeugt, in das der individuelle Code verlagert wird. Dieses Template wird lediglich aus dem Template `NodeEditPart.xpt` (im Verzeichnis `editparts`) aufgerufen und zur Neugenerierung wiederverwendet.

Die Implementierung mit Hilfe der Objekt- und Aspektorientierung erfolgt prinzipiell analog zum vorherigen Anwendungsfall. Individueller Code wird durch Klassen bzw. Aspekte gekapselt, die zur Neugenerierung wiederverwendet werden. Nach der Neugenerierung bleibt bei der objektorientierten Adaption nur die Vererbungshierarchie anzupassen.

An diesem Anwendungsfall ist nachhaltig erkennbar, dass die direkte Adaption den Adaptionsaufwand nicht verringern konnte und gar unerwünschte Redundanz im Generat erzeugt. Auch die Objektorientierung konnte den Aufwand nicht befriedigend verringern. Bei diesem Ansatz wirkt sich die Adaption der Vererbungshierarchie nach der Neugenerierung entscheidend aus, was erneut an `A` und `LoC` erkennbar ist. Geringster Aufwand wurde durch die Adaption am Template erreicht, sie konnte den Aufwand deutlich senken. Etwas nachteilig ist, dass bei der Neugenerierung Templates hinzugefügt und bestehende erneut adaptiert wurden. Diesen Nachteil besitzt die Aspektorientierung nicht, welche ebenfalls den Aufwand stark reduzieren konnte und bei diesem Anwendungsfall favorisiert wird.

Anwendungsfall	Textalignment						Celleditor					
	a		A		LoC		a		A		LoC	
Kriterien / Technik	P_i	P_m	P_i	P_m	P_i	P_m	P_i	P_m	P_i	P_m	P_i	P_m
Direkte Adaption	138	138	23	23	851	851	23	23	23	23	23	23
Adaption am Template	5	1	2	2	60	1	1	1	1	1	1	1
Objektorientierte Adaption	31	23	24	24	62	23	73	69	24	24	41	23
Aspektorientierte Adaption	12	0	4	1	70	0	73	0	5	1	303	0

Tabelle 1: Adaptionsaufwand im Überblick

4 Zusammenfassung und Fazit

Die Migration eines bereits mit individuellen Code angepassten Editors stellt ein Problem dar, zu dessen Lösung in diesem Beitrag Adaptionstechniken untersucht und anhand ihres Adaptionsaufwands unter konkreten Anwendungsfällen verglichen wurden.

Die Ergebnisse der Evaluation aus Kapitel 3.3 zeigen, dass das Adaptionsproblem nicht von einer der vorgestellten Techniken alleine zufriedenstellend gelöst werden konnte. Wird jedoch zwischen den Adaptionsformen differenziert, können einzelne Techniken durchaus überzeugen, wobei zur Adaption am Framework der aspektorientierte und zur Adaption des Generates der Template-basierte Ansatz als vorteilhafter eingeschätzt werden kann.

Für vertiefende Untersuchungen sind weitere Adaptionstechniken zu betrachten. Möglichkeiten ergeben sich durch aspektorientierte Templates, die bspw. mit oAW-Techniken umgesetzt werden können, sowie objektorientierte Templates, in denen die Adaption der Vererbungshierarchie durch Templates übernommen wird. Ebenfalls interessant ist die Aufhebung der in Kapitel 3.1 erwähnten Einschränkung. Weitere Arbeiten könnten somit das Problem mit unterschiedlich implementierbaren Anwendungsfällen betrachten.

Literatur

[Böh06] Oliver Böhm. *Aspektorientierte Programmierung mit AspectJ 5*. dpunkt.verlag, 2006.

[Gro09] Richard C. Gronback. *eclipse Modeling Project - A Domain-Specific-Language (DSL) Toolkit*. the eclipse series. Addison-Wesley Professional, 2009.

[GV09] Iris Groher und Markus Völter. Aspect-Oriented Model-Driven Software Product Line Engineering. In *Transactions on Aspect-Oriented Software Development VI*. Springer Berlin Heidelberg, 2009.

[HM07] Stephan Herrmann und Marco Mosconi. Integrating Object Teams and OSGi: Joint Efforts for Superior Modularity. In *Journal of Object Technology*, Jgg. 6, Seiten 105–125, 2007.

[SVEH07] Thomas Stahl, Markus Völter, Sven Efftinge und Arno Haase. *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*. dpunkt.verlag, 2007.