Modelltransformation mit der QVT Relationssprache - Fallstudie einer werkzeugspezifischen Realisierung

Oliver Alt
Robert-Bosch GmbH, CM-DI/EAA3
Daimlerstr. 6, D-71229 Leonberg
oliver.alt2@de.bosch.com

Abstract: Modelltransformationen spielen innerhalb einer Modell-basierten Softwareentwicklung eine entscheidende Rolle. Ein relativ neuer Standard solche Modelltransformationen zu beschreiben ist Query View Transformation (QVT) der Object Management Group (OMG). Neben einer imperativen Sprache zur Beschreibung von Modelltransformationen bietet QVT auch einen deklarativen Teil - die QVT Relationssprache. Ein großer Vorteil der QVT Relationssprache besteht darin, dass im Gegensatz zur imperativen Sprache neben einer textuellen Syntax auch eine grafische definiert wurde. Damit können Modelltransformationen für grafische Modelle selbst als grafische Modelle beschrieben werden. Dies ermöglicht innerhalb einer Modell-basierten Entwicklung eine durchgängig grafisch orientierte Beschreibung der am Prozess beteiligten Elemente (Modelle und Transformationen) beizubehalten und eine Rückkehr zu textuellen Formaten zu vermeiden. Dieses Papier beschreibt die Erkenntnisse einer Fallstudie, in der die Realisierung der QVT Relationssprache zum Einsatz in der Praxis untersucht wurde. Dabei sollte die Umsetzung spezifisch für das eingesetzte UML Werkzeug Sparx Systems Enterprise Architect (EA) erfolgen um kein neues, zusätzliches Werkzeug einsetzen zu müssen.

1 Einleitung

Überall dort wo (UML-)Modelle im Softwareentwicklungsprozess eingesetzt und zu mehr als Dokumentationszwecken verwendet werden spielt Modelltransformation eine entscheidende Rolle. Seit einiger Zeit existiert mit QVT [OMG05] ein OMG Standard zur Beschreibung solcher Modelltransformationen. Außer einem imperativen Teil wird auch ein deklarativer Teil spezifiziert - die QVT Relationssprache. Die QVT Relationssprache besitzt zudem eine grafische Notation, basierend auf UML Objektdiagrammen. Die grafische Notation ist mit der textuellen äquivalent und bietet dadurch neben einem intuitiven Zugang auch die Möglichkeit Modelltransformationen als UML Diagramm zu dokumentieren. Leider ist momentan noch kein kommerzielles UML Werkzeug auf dem Markt, mit dem man solche grafischen QVT Diagramme zeichnen kann.

Um die beschriebenen Vorteile von QVT nutzen zu können, wurde daher eine Fallstudie durchgeführt, mit dem Ziel herauszufinden, inwieweit sich das in der Entwicklung bei der Robert Bosch GmbH eingesetzte UML Werkzeug Enterprise Architect [Spa06a] um die Unterstützung des grafischen QVT Standards erweitern lässt. Insbesondere sollen sich Modelltransformationen für entwickelte Konzepte (z.B. [Alt06]) durch die grafische QVT-Relationssprache dokumentieren und bestenfalls auch ausführen lassen.

1.1 Untersuchungsgegenstand

Folgende Punkte sollten auf ihre Umsetzbarkeit untersucht und mögliche Lösungsansätze dafür erarbeitet werden:

1. Es soll ermöglicht werden Transformationsregeln im grafischen QVT Relationsstandard mit Enterprise Architect zu erstellen. Dabei sollen nach Möglichkeit Informationen aus dem/den der Transformation zugrunde liegenden Metamodells/Metamodellen für die Benutzerführung genutzt werden (z.B. Nachfrage bei Mehrdeutigkeiten von Assoziationen, etc.).
2. Die Generierung der textutellen Entsprechung der Transformation soll aus dem grafischen Modell ermöglicht werden (z.B. zum Einsatz in anderen QVT Werkzeugen oder zur kompakten Dokumentation der Regeln).
3. Es soll untersucht werden, ob und wenn ja mit welchem Aufwand es möglich ist einen Interpreter für die grafischen QVT Regeln zu realisieren. Wichtig dabei ist es zu untersuchen, ob eine solche Realisierung durch Berücksichtigung werkzeugspezifischer Aspekte erleichtert werden kann; beispielsweise durch Beschränkung auf Transformationen innerhalb von EA unter ausschließlicher Nutzung dessen Metamodells.

Das weitere Papier gliedert sich wie folgt. Im folgenden Abschnitt werden die Erkenntnisse und Realisierungsideen im Einzelnen diskutiert, die Aufgrund der oben beschriebenen Anforderungen gefunden wurden, bevor Abschnitt 3 ein abschließendes Fazit zieht.

2 Erkenntnisse und Realisierungsideen

Folgende Erkenntnisse und Realisiserungsideen konnten für die in Abschnitt 1.1 aufgezählten Punkte gewonnen werden:

Punkt 1: Editor zur Erstellung grafischer QVT Regeln

Die grafischen QVT Transformationsregeln basieren auf den UML Objektdiagrammen. Hinzu kommen einige neue Modellierungselemente wie z.B. der sechseckige Relationsknoten. Zur Realisierung solcher domänenspezifischer Elemente bietet Enterprise Architect die Möglichkeit UML Profile zu definieren und mit der werkzeugeigenen Sprache `ShapeScript` Einfluss auf das Aussehen der Modellelemente zu nehmen.

Damit konnten die notwendigen Modellierungselemente als UML Profil realisiert werden. Abbildung 1 zeigt ein Beispiel einer mit EA gezeichneten QVT Relation. Ein weiterer Punkt war die Unterstützung des Benutzers während der Erstellung der Transformationen, beispielsweise durch speziell angepasste Dialoge zur schnelleren Eingabe der notwendigen Parameter. Enterprise Architect besitzt eine sogenannte Addin-Schnittstelle für eigene Erweiterungen. Diese kann mit allen .net Sprachen programmiert werden und es

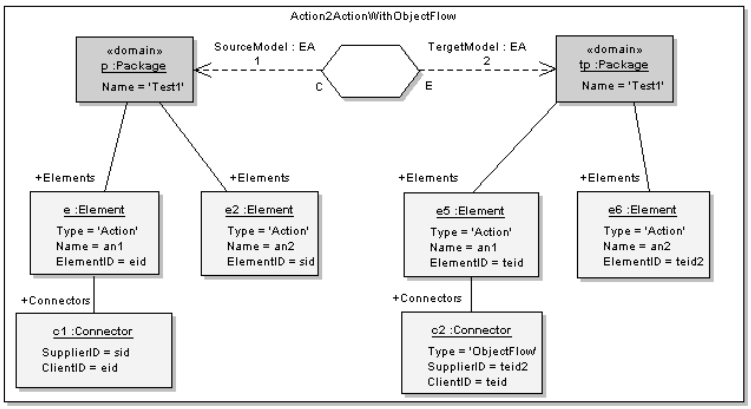

Abbildung 1: Beispiel einer mit Enterprise Architect erstellten QVT Relation

ist möglich eigene Dialogelemente im Kontext der gerade ausgeführten Benutzeraktion in EA anzuzeigen. Durch die Realisierung eines solchen Addin kann die gewünschte Benutzerführung erreicht werden. Durch die in EA bereits standardmäßig vorhandene Möglichkeit Meta-Modelle zu erstellen, lassen sich zudem die für eine Benutzerführung zugrunde liegenden Metamodelle gleichfalls erstellen.

Ein entsprechendes QVT-Editor-Addin wurde zwischenzeitlich realisiert und ermöglicht eine komfortable Eingabe von grafischen Transformationsregeln innerhalb von Enterprise Architect.

Punkt 2: Generierung der textuellen Repräsentation

Wie bereits oben erwähnt, lässt sich über die Addin-Schnittstelle auf die EA Modelldaten zugreifen und diese manipulieren. Damit kann ein Konverter realisiert werden, der die grafischen Modelle analysiert und die textuell-äquivalente QVT Transformation erzeugt. Ein funktionierender Prototyp eines solchen Umsetzers wurde dann auch auf diese Weise mit geringem Aufwand - innerhalb von 2 Tagen - realisiert.

Auch eine Umkehrung, also eine Erzeugung grafischer QVT Regeln aus der textuellen Repräsentation ist denkbar, da beide Sichten äquivalent sind. Einen entsprechenden QVT-Parser vorausgesetzt, sollte sich ohne Probleme ein grafisches QVT Regelmodell erzeugen lassen. Im Hinblick auf die Diagramme bestehen dabei bekannte Probleme einer Text-Modell Transformation, wie z.B. ein lesbares Layout nach dem Import. Im Rahmen der Fallstudie wurde ein solcher Import allerdings bislang nicht implementiert, da wo immer möglich die Modelle und Regeln in grafischer Form erstellt und eingesetzt werden sollen.

Punkt 3: Realisierung eines QVT Interpreters

Um die grafischen QVT Regeln nicht nur für Dokumentationszwecke zu gebrauchen, wurde untersucht inwieweit und mit welchem Aufwand sich ein Interpreter für die Regeln realisieren lässt. Mit einer empirischen Untersuchung konnten folgende Erkenntnisse ge-

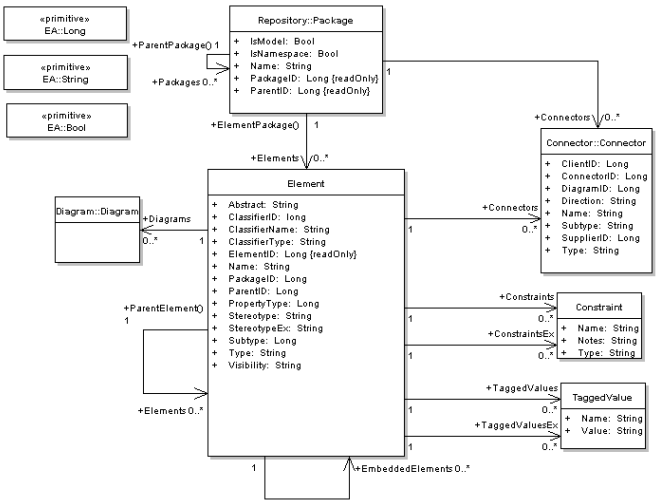

Abbildung 2: Auszug aus dem EA Metamodell

wonnen werden: Durch die Beschränkung auf Transformation innerhalb von EA lässt sich der Aufwand einer solchen Implementierung stark verringern. Grundlage solcher Transformationen bildet dann das EA-Addin-Metamodell (Abb. 2) [Spa06b].

Dadurch wird die Universalität von QVT zwar eingeschränkt, jedoch stellen Transformationen von UML nach UML in der Praxis doch den häufigsten Anwendungsfall dar.

Die Implementierung lässt sich u.a. dadurch vereinfachen, dass die zur Umsetzung der Transformation notwendigen Such- und Erzeugungsoperationen auf dem EA-Modell den .net Reflection-Mechanismus verwenden. Dabei werden Attribute und Methoden nicht mehr direkt angesprochen, sondern stattdessen Attribut- oder Methodenamen zur Laufzeit als String-Parameter an die Reflection-Methoden übergeben. Auf diese Weise können alle zum Zugriff auf die EA-Modelldaten benötigten Informationen direkt und zur Laufzeit aus dem erstellten EA-Metamodell gewonnen werden. Eine solche Implementierung hat darüber hinaus den Vorteil, dass sie auch nach Änderungen an der Addin-Schnittstelle unverändert funktioniert, sobald die Änderungen in das Metamodell eingepflegt werden.

Dieser Ansatz ermöglichte innerhalb einiger Wochen die Erstellung eines ersten lauffähigen Prototypen, der in der Lage ist eine eingeschränkte Menge von Transformationen des vollständigen QVT-Relationsstandards durchzuführen. Es lassen sich damit aber bereits eine Vielzahl der benötigten Transformationen evaluieren.

Dem Prototypen fehlen allerdings noch wichtige Funktionen, wie beispielsweise die Unterstützung von OCL-Konstrukten[1] zur Umsetzung komplexer Transformationen. Deren Realisierung nimmt schätzungsweise ein vielfaches der Zeit in Anspruch, die die Realisierung bis dato benötigte.

In Anbetracht der in Kürze zu erwartenden QVT Realisierungen aus Forschungs- und In-

[1] OCL = Object Constraint Language [OMG06]

dustrieprojekten (z.B. [BD06], [AKRS06], [L$^+$06]) wurde die Realisierung des Interpreters jedoch zunächst nicht erweitert, sondern stattdessen Arbeit in die Verbesserung und Vervollständigung des QVT-Editors investiert.

3 Fazit

Im Rahmen dieser Fallstudie konnte gezeigt werden, dass die grafische QVT Relationssprache mit dem in der Industrie weit verbreiteten UML Werkzeug Enterprise Architect mit vertretbarem Aufwand unterstützt werden kann. Neben einem benutzerfreundlichen Editor zur grafischen Erstellung der Transformationsregeln entstand zudem ein Konverter zur Generierung der textuell-äquivalenten Repräsentation. Darüber hinaus wurde der Prototyp eines QVT-Interpreters mit Beschränkung auf das EA Metamodell realisiert. Grafische QVT-Regeln lassen sich nun effizient innerhalb eines Standard Modellierungswerkzeuges wie Enterprise Architect erstellen, als textuelle Repräsentation exportieren und in begrenztem Maße auch evaluieren, bzw. ausführen.

Dies ermöglicht erstmals den Einsatz der QVT Relationssprache im industriellen Umfeld - zunächst für Dokumentationszwecke. Mit einer, in nächster Zeit zu erwartenden, steigenden Zahl an Werkzeugen, die QVT Relationen (vollständig) ausführen können, kann dann auch die Ausführung der erstellten standardkonformen Regeln Teil des Entwicklungsprozesses in der täglichen Praxis der Modell-basierten Softwareentwicklung werden.

Literatur

[AKRS06] C. Amelunxen, A. Königs, T. Rötschke und A. Schürr. MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In A. Rensink und J. Warmer, Hrsg., *Model Driven Architecture - Foundations and Applications: Second European Conference*, Seiten 361–375, Heidelberg, 7 2006. Springer Verlag.

[Alt06] O. Alt. Generierung von Systemtestfällen für Car Multimedia Systeme aus domänenspezifischen UML Modellen. In C. Hochberger und R. Liskowsky, Hrsg., *INFORMATIK 2006 Informatik für Menschen Band 2*. Lecture Notes in Informatics, 10 2006.

[BD06] M. Belaunde und G. Dupe. *MDDi QVT Tool*. France Telecom, September 2006. http://universalis.elibel.tm.fr/qvt/, zuletzt besucht am 22.2.07.

[L$^+$06] T. Levendovszky et al. *Realizing QVT with VMTS (Graph Transformation Based Model Transformation)*, 2006. http://avalon.aut.bme.hu/∼tihamer/research/vmts/qvt/vmts_qvt.html, zuletzt besucht am 22.2.07.

[OMG05] OMG. *MOF QVT Final Adopted Specification*. OMG, ptc/05-11-01. Auflage, 11 2005.

[OMG06] OMG. Object Constraint Language OMG Available Specification Version 2.0. formal/06-05-01, OMG, 5 2006.

[Spa06a] Sparx Systems Pty Ltd. *Enterprise Architect 6.x*, 2006. http://www.sparxsystems.com, zuletzt besucht am 22.2.07.

[Spa06b] Sparx Systems Pty Ltd. *Entreprise Architect Version 6.5 User Guide - The Automation Interface*, 2006. http://www.sparxsystems.com/EAUserGuide/automationinterface.htm, zuletzt besucht am 22.2.07.