Evaluation von Sprachen zur Spezifikation von Transformationen in modellbasierten Entwicklungsprozessen von Benutzungsschnittstellen

Daniel Einig, Klaus Müller (Fachhochschule Trier*)
Gerrit Meixner (Deutsches Forschungszentrum für Künstliche Intelligenz†)

1 Zusammenfassung

In diesem Beitrag werden verschiedene Transformationsprozesse miteinander verglichen und ihre Anwendbarkeit während der Entwicklung von modellbasierten Benutzungsschnittstellen bewertet. Dazu dient ein Kriterienkatalog, der auf den Einsatz der Transformationsansätze in der modellbasierten Entwicklung von Benutzungsschnittstellen angepasst wird. Im Rahmen dieses Beitrages werden drei verschiedene Ansätze analysiert und evaluiert. QVT stellt einen hybriden Ansatz dar und besteht aus den beiden deklarativen Sprachen QVT Relations und QVT Core sowie der imperativen Sprache QVT Operational Mappings. Zur Durchführung der Beispieltransformation wurde mit SmartQVT eine Implementierung von QVT Operational Mappings eingesetzt. ATL stellt ebenfalls einen hybriden Ansatz dar, der in einigen Bereichen eine hohe Ähnlichkeit zu QVT besitzt. Ähnlich wie QVT leistet ATL für die Modelltransformation gute Dienste. XSLT, ebenfalls ein hybrider Ansatz, zeigt seine Stärken klar bei der Verarbeitung von XML-Dokumenten. Der Einsatz zur Modelltransformation erfolgt über den „Umweg“ der Umwandlung des Modells in eine XML-Repräsentation.

2 Modellgetriebene Architekturen

2.1 Model Driven Architecture (MDA)

Mit der Model Driven Architecture (MDA) liegt aktuell ein Standard der Object Management Group (OMG) vor, der die Repräsentation von Software von der Programmcodeebene auf die Modellebene heben möchte (vgl. [MM03], [KM05]). „Um die Komplexität auf Modellebene zu reduzieren, werden Modelle und Plattformen verschiedener Abstraktionsebenen (Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) und Implementation Specific Model (ISM)) unterschieden. Die Grundidee des Ansatzes ist nicht neu, sondern folgt dem Grundsatz, dass die Spezifikation einer Komponente unabhängig von der technischen Umsetzung zu beschreiben ist. Neu ist, dass mit der MDA aufeinander abgestimmte Spezifikationen und ein Konzept für die modellgetriebene Softwareentwicklung existieren. Dabei kann die Realisierung eines plattformunabhängigen Modells durch die Wahl einer Plattform, also der konkreten technischen Umsetzung, teilweise oder vollständig automatisiert erfolgen. Mit der MDA wird das Modell zum zentralen Element des Softwareentwicklungsprozesses.“ [KM05] Zur Modellableitung zwischen den einzelnen Ebenen werden Transformationen benutzt. „Eine Transformation kann manuell, semiautomatisch oder vollautomatisch erfolgen. Jede Transformation besteht aus der Anwendung von Transformationsregeln, sogenannten Mappings (vgl. [WG08]), die für die (semi-)automatische Ausführung formal definiert sind. Die Transformation reichert die Information bspw. des PIMs durch zusätzliche Information der ausgewählten Plattform und der verwendeten Transformationsregeln an. Neben statischen Erweiterungen kann hier auch technisches, durch die Wahl der Plattform motiviertes, dynamisches Verhalten hinzukommen.“ [KM05]

2.2 Model-Based User Interface Development (MBUID)

Die stetig wachsende Zahl verschiedenster mobiler Geräte stellt eine Herausforderung für Entwickler dar [APQAS02]. Es muss nicht nur eine Benutzungsschnittstelle für eine Plattform, sondern eine Applikation mit einer Vielzahl von Benutzungsschnittstellen (bspw. mit verschiedenen Bildschirmgrößen, Grafikbibliotheken oder Ressourcen) für verschiedene Geräte entwickelt werden [BBSS02]. Zum einen ist es schwierig, Anforderungen über mehrere Plattformen hinweg konsistent in Bezug zu Funktionalität und Visualisierung zu erfüllen [Mol04] und zum anderen impliziert die Entwicklung von Anwendungen für verschiedene Plattformen auch, dass Entwickler eine Vielzahl von Programmiersprachen und Methoden beherrschen müssen [BS02]. Bei der modellbasierten Entwicklung von Benutzungsschnittstellen wird das CAMELEON Framework als Gegenstück zur MDA verwendet [CCT+03]. Das CAMELEON Framework definiert vier Ebenen: Aufgaben und Konzepte, Abstrakte Benutzungsschnittstelle, Konkrete Benutzungsschnittstelle und Finale Benutzungsschnittstelle. Dabei entsprechen diese Ebenen den Abstraktionsstu-

*Fachbereich Informatik, Schneidershof, 54293 Trier
†Forschungsbereich Innovative Fabriksysteme, Trippstadter Str. 122, 67663 Kaiserslautern, Gerrit.Meixner@dfki.de

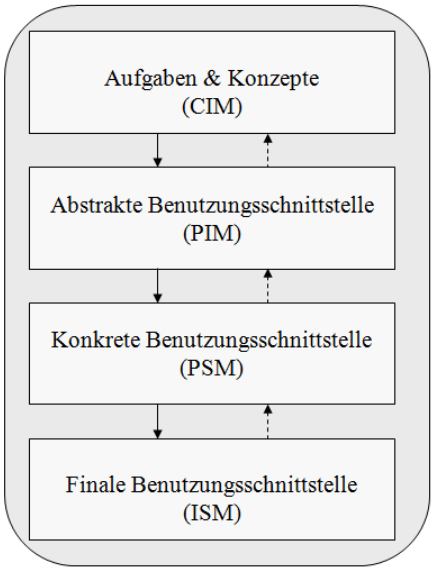

Abbildung 1: Die Ebenen des CAMELEON Frameworks

fen der MDA (vgl. Abbildung 1).

Die verwendeten Modelle der jeweiligen Ebene werden im Bereich der Entwicklung von Benutzungsschnittstellen meist mit deklarativen Sprachen als XML-Anwendung spezifiziert [Bis06]. Das CAMELEON Framework empfiehlt einen Top-Down-Ansatz, beginnend mit abstrakten Beschreibungen relevanter Informationen über die Benutzungsschnittstelle [ATK03]. Diese abstrakten Beschreibungen werden interpretiert oder transformiert, um eine partielle oder vollständige Generierung von Quellcode zu erreichen. Somit ist es nicht mehr erforderlich, für jede Zielplattform separaten Quellcode zu entwickeln [NC04]. Für einen umfassenderen Überblick zu modellbasierten Architekturen wird auf [MG09] und [Mei10] verwiesen.

3 Transformationssprachen

3.1 Query View Transformation (QVT)

QVT ist ein Standard, der von der Object Management Group (OMG) zur Durchführung von Modelltransformationen definiert wurde und drei verschiedene Transformationssprachen umfasst. QVT Relations ist eine deklarative Sprache zur Spezifikation von Beziehungen zwischen Modellen. QVT Core liegt QVT Relations zugrunde und bietet die gleiche Funktionalität auf einer anderen Abstraktionsebene. QVT Operational Mappings ist eine imperative Sprache zur Spezifikation von Transformationsabläufen.

Zur Durchführung der in diesem Beitrag verwendeten Beispieltransformation wurde eine Implementierung von QVT Operational Mappings eingesetzt. Die Zuordnung zwischen Elementen des Quellmodells zu Elementen des Zielmodells erfolgt in diesem Fall über Mapping-Operationen.

Sobald eine Transformation komplett in QVT Operational Mappings spezifiziert wurde, spricht man von einer operationalen Transformation. Eine operationale Transformation stellt die imperative Definition einer unidirektionalen Transformation dar. Darüber hinaus können bestimmte Aspekte der Transformation in QVT Relations (oder QVT Core) und andere Aspekte über Black Box-Mappings der QVT Operational Mappings Sprache ausgelagert werden. In diesem Fall spricht man von einem hybriden Ansatz. Die Zusammenhänge zwischen den verschiedenen Transformationssprachen sind in Abb. 2 dargestellt.

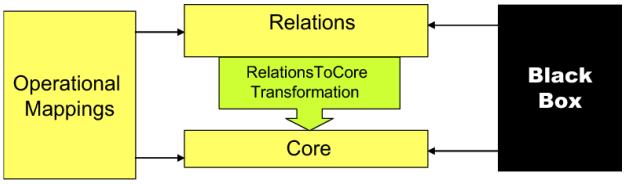

Abbildung 2: Übersicht der Zusammenhänge zwischen den verschiedenen Transformatiossprachen von QVT (aus [OMG08, S. 9])

3.2 Atlas Transformation Language (ATL)

ATL ist ein Teil der ATLAS Model Management Architecture (AMMA) und besitzt in einigen Bereichen eine hohe Ähnlichkeit zu QVT. ATL-Transformationen bestehen aus drei Teilen: Einem Header, Helper-Operationen und Regeln. Im Header werden generelle Informationen wie das Quell-Metamodell und das Ziel-Metamodell deklariert. Helper-Operationen sind vergleichbar zu Methoden anderer Programmiersprachen. Die Regeln beschreiben, wie aus dem Quellmodell das Zielmodell erzeugt wird. ATL kennt imperative und deklarative Regeln sowie eine Mischform, die hybriden Regeln.

3.3 XSL Transformation (XSLT)

XSLT ist eine Empfehlung des World Wide Web Consortium (W3C). Es handelt sich um eine flexible und leistungsfähige Sprache, um XML-Dokumente zu transformieren. Das Ergebnis der Transformation kann ein anderes XML-Dokument, ein textbasiertes Format oder auch eine Binärdatei sein (vgl. Abbildung 3). Die Transformation wird durch ein Stylesheet gesteuert. Diese XSLT- Stylesheets sind ebenfalls XML-Dokumente.

4 Taxonomie

In diesem Abschnitt werden die Kriterien vorgestellt, anhand derer die Transformationssprachen verglichen werden. Die aufgeführten Kriterien basieren auf den Ergebnissen von [Hub08], [LJMR09] sowie [Sch07] und wurden für den Einsatz in der modellbasierten Entwicklung von Benutzungsschnittstellen angepasst.

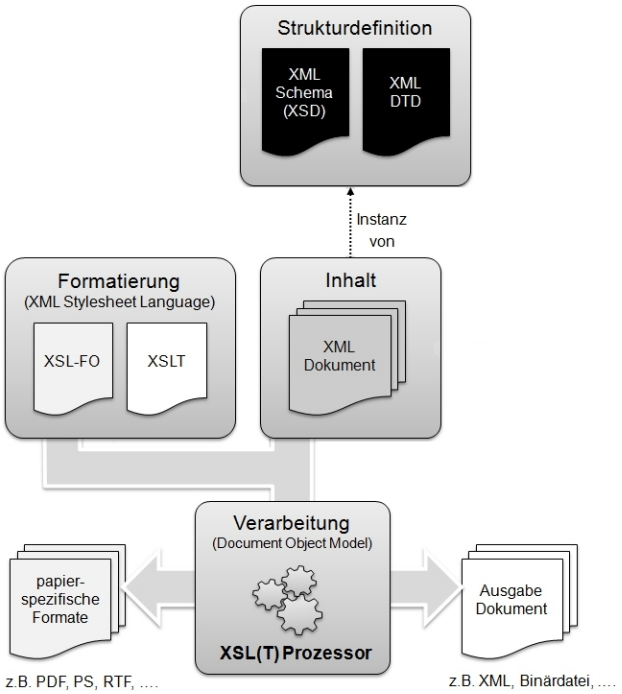

Abbildung 3: Schema einer Transformation mittels XSLT

4.1 Transformationsregeln

Transformationregeln können als kleinste Einheit einer Transformation verstanden werden. Die Unterstützung von Transformationsregeln kann in einer entsprechenden Sprache als gegeben betrachtet werden, weshalb dies kein Kriterium darstellt. Um die Modelle vergleichen zu können, werden entsprechende Unterkriterien eingeführt.

Das erste Unterkriterium bezieht sich auf die Möglichkeit, die Anwendung einer Transformationsregel durch zusätzliche Bedingungen einzuschränken. Dabei wird die Regel nur ausgeführt, wenn die Bedingung zutrifft.

Das zweite Unterkriterium betrifft Zwischenstrukturen (Intermediate Structures), die von der Modelltransformationssprache erstellt werden können, um die Transformation auszuführen. Diese Informationen sind normalerweise nicht Teil der Modelltransformation. Zum Beispiel stellen Links für die Nachverfolgung (Traceability) eine solche Zwischenstruktur dar.

Das dritte Unterkriterium betrifft die Reflektion, welches Transformationsregeln erlaubt, selbst auf die Transformationen zuzugreifen.

Das vierte und letzte Unterkriterium betrifft die Parametrisierung. Dabei können Transformationsregeln zusätzliche Informationen mitgegeben werden. Kontrollparameter ermöglichen die Übergabe von Werten. Generics ermöglichen das Übergeben von Datentypen als Parameter. Dies ist mit Generics aus Java oder C++ vergleichbar. Regeln höherer Ordnung ermöglichen zudem die Nutzung von Regeln als Parameter.

4.2 Koordination der Regelanwendung

Die Koordination der Regelanwendung dient dazu, die Reihenfolge und Art der Regelanwendung zu ermitteln. Die Reihenfolge der Regelanwendung kann auf implizite oder explizite Weise spezifiziert sein. In einer impliziten Reihenfolge hat der Entwickler keinen direkten Einfluss über die Ausführungsreihenfolge. In einer expliziten Reihenfolge wird die Ausführung durch einen Ablaufplan (Schedule) vorgegeben. Dieser Ablaufplan kann intern oder extern sein. Mit einem internen Ablaufplan (internal rule scheduling) kann der Entwickler innerhalb von Regeln andere Regeln aufrufen. Ein externer Ablaufplan ermöglicht, die Ausführungsreihenfolge getrennt von den Regeln festzulegen.

Die Regeliteration kennzeichnet die Weise, wie Regeln bei der Transformation wiederholt angewendet werden. Wenn Regeln aus sich selbst heraus erneut aufgerufen werden können, spricht man von Rekursion. Werden Regeln wiederholt ausgeführt, geschieht dies in Schleifen. Fixpunktiteration bezeichnet nach [CH06] die wiederholte Anwendung der Regeln, bis keine Änderungen mehr festgestellt werden.

4.3 Regelorganisation

Regelorganisation fasst Merkmale zusammen, die mit der Strukturierung und Zusammensetzung der Transformationsregeln in Zusammenhang stehen. Mechanismen zur Modularisierung erlauben die Gruppierung einzelner Regeln.

Organisationsstrukturen beschreiben, wie Regeln generell organisiert sind. In einer quellorientierten Organisationsstruktur sind die Regeln entsprechend des Quell-Metamodells angeordnet. Analog gilt dies für eine zielorientierte Organisationsstruktur. In einer unabhängigen Organisationsstruktur sind die Regeln demgegenüber individuell angeordnet.

4.4 Beziehung zwischen Quelle und Ziel

Die Beziehung zwischen Quelle und Ziel gibt an, wie Zielelemente aus Quellelementen erstellt werden. Modelltransformationssprachen, die nach dem Prinzip „New Target“ arbeiten, erstellen immer ein neues Zielmodell aus dem Quellmodell. Die Manipulation bereits bestehender Zielmodelle durch die Aktualisierung oder die Erweiterung der Zielmodelle wird hingegen als „Existing Target“ bezeichnet. Eine spezielle Form dieses Ansatzes ist die In-Place-Transformation, bei der Quell- und Zielmodell demselben Modell entsprechen.

4.5 Direktionalität

Eine Modelltransformationssprache kann unidirektional oder multidirektional sein. Unidirektionale Sprachen erlauben nur Transformationen eines bestimmten Quellmodells in ein bestimmtes Zielmodell. Multidirektionale Sprachen erlauben auch die entgegengesetzte Transformation vom Ziel- zum Quellmodell.

4.6 Tracing

Die Verbindung von Quell- und Zielmodell wird „Tracing“ genannt. Dazu werden Links zwischen den Quell- und den entsprechenden Zielelementen erstellt. Diese können entweder automatisch erstellt oder durch den Entwickler angestoßen sowie im Quellmodell oder separat gespeichert werden.

4.7 Weitere Kriterien

Die folgenden Kriterien sind nach [Sch07] besonders für die modellbasierte Entwicklung von Benutzungsschnittstellen relevant. Die Art des Programmiermodells, also deklarativ oder imperativ, stellt im Hinblick auf die Erfahrungen des Entwicklers ein wichtiges Kriterium dar. Nach [Sch07] sind viele finale Benutzungsschnittstellen (FUI) XML-basiert. Eine Sprache, die XML breit unterstützt, kann sich für solche FUIs als besonders nützlich herausstellen. Ein weiterer wichtiger Aspekt ist, ob der Transformationsansatz fähig ist, auch nicht XML-basierten Code zu transformieren oder zu generieren. Zuletzt soll noch betrachtet werden, ob ein Transformationsansatz erweiterbar ist.

5 Auswertung der Anwendung der Taxonomie auf die Transformationssprachen

In diesem Abschnitt werden die Ergebnisse der Anwendung der Taxonomie auf die Transformationssprachen SmartQVT, ATL und XSLT vorgestellt. Die Ergebnisse finden sich in zusammengefasster Form in Tabelle 1. SmartQVT lag in der Version 1.0 vor, ATL in der Version 2.0.2 und XSLT in Version 2.0.

5.1 SmartQVT

SmartQVT unterstützt sowohl Vor- als auch Nachbedingungen. Zwischenstrukturen werden im Wesentlichen für das unidirektionale Tracing genutzt, es lassen sich aber durchaus eigene Zwischenstrukturen in Form von Zwischenklassen bzw. Zwischeneigenschaften anlegen.

SmartQVT unterstützt weder Generics noch Regeln höherer Ordnung. Die Reihenfolge der Regelanwendung muss explizit festgelegt werden. Die Mapping-Operationen werden dabei, beginnend beim Einstiegspunkt, der Reihe nach abgearbeitet. SmartQVT folgt auf diese Weise einem festen Ablaufplan bei dem zur Iteration Rekursion zum Einsatz kommt.

Für die Regelorganisation existieren Ansätze, mit denen sich Regeln vereinen und verketten lassen. Auch das Konzept der Vererbung wird unterstützt.

Desweiteren stellt SmartQVT keine Möglichkeit zur Verfügung, bereits erzeugte Modelle zu aktualisieren, da SmartQVT lediglich QVT Operational und nicht QVT Relations implementiert.

Erweiterungen sind in QVT durch die Verwendung der Black-Box-Mappings möglich.

5.2 ATL

ATL unterstützt Bedingungen durch boolesche Ausdrücke im from-Teil einer Regel. Die Regel wird nur ausgeführt, wenn die Bedingung wahr wird. Für das integrierte unidirektionale Tracing werden Zwischenstrukturen angelegt. Auch bei ATL erfolgt die Iteration rekursiv, wobei implizite und explizite Regelaufrufe erlaubt sind.

Zur Regelorganisation können Helper-Operationen in Bibliotheken organisiert werden, die dann in Modulen Verwendung finden können. Ein Modul bezeichnet in ATL den Code einer Modell-zu-Modell-Transformation. Diese Organisationsstruktur basiert dabei grob auf der Struktur des Quellmodells. Für ein Quellelement können verschiedene Zielelemente definiert werden.

ATL kann bereits erstellte Zieldokumente verändern und unterstützt somit als einziger untersuchter Ansatz das Existing-Target-Kriterium.

In ATL ist die Erweiterbarkeit durch die Möglichkeit des Aufrufs nativer Funktionen gegeben.

5.3 XSLT

XSLT kann ebenso mit Bedingungen umgehen, unterstützt aber kein Tracing, weshalb auch keine Zwischenstrukturen angelegt werden. Da die Sprache keine Werkzeuge zur Rule Application Control bereitstellt, muss der Entwickler hier selbst Hand anlegen. Regeln können dazu innerhalb anderer Regeln aufgerufen oder vorsortiert werden, was allerdings keinesfalls einem Scheduling entspricht.

XSLT setzt auf Rekursion, weshalb für das Kontrukt einer for-Schleife nur emulierende Ansätze existieren.

Die Organisationsstruktur von XSLT ist stark quellorientiert, da die Regelanwendung vom Quelldokument ausgehend strukturiert wird. Bei jeder Transformation wird immer ein neues Zieldokument erstellt, welches im Nachhinein nicht automatisch aktualisiert oder erweitert werden kann.

Für findige Entwickler ist XSLT multidirektional. Für den Fall, dass Ein- und Ausgabe XML-Dokumente sind, kann über entsprechende Regeln oder ein weiteres Stylesheet wieder in das Quelldokument zurücktransformiert werden. Wird allerdings nicht nach XML transformiert, ist XSLT unidirektional.

XSLT ist laut [Sch07] nicht erweiterbar.

5.4 Zusammenfassung

Was die Fähigkeit der Modelltransformation angeht, sind nur QVT und ATL tatsächlich dafür ausgelegt. Allerdings können Ansätze, die auf XML arbeiten, die XML-Repräsentationen der Modelle verarbeiten. Daher ist XSLT prinzipiell in der Lage, Modelle zu transformieren.

XSLT arbeitet nach [Sch07] nur mit XML-Dokumenten und kann daher nicht zur Code-

Transformation benutzt werden. Allerdings kann XSLT auch nicht-XML-Code, also zum Beispiel Klartext, aus XML-Dokumenten erzeugen, wodurch prinzipiell Code generiert werden kann. Demgegenüber sind Code-Generierungen und Code-Transformationen mit SmartQVT und ATL nicht möglich.

Tabelle 1 zeigt die Einzelergebnisse der Auswerung für die untersuchten Transformationsprozesse. Werte in Klammern bedeuten dabei, dass der Ansatz das Kriterium „über Umwege“ unterstützt, aber nicht speziell dafür entwickelt wurde.

6 Fazit

In Anbetracht der Vielfalt der Modellierungsaufgaben kann keine klare Empfehlung für den ein oder anderen Ansatz gegeben werden. Der Transformationsansatz muss entsprechend der Aufgabe und ihren gezielten Erfordernissen ausgewählt werden. Für rein modellgetriebene Ansätze sind QVT und ATL eine gute Wahl. Aber auch der XML-basierte Ansatz XSLT kann damit umgehen, falls die Modellrepräsentation eine XML-Syntax besitzt oder sich leicht in eine solche überführen lässt. Hier und da kann auch eine Kombination verschiedener Ansätze hilfreich sein.

Laut [Sch07] arbeiten viele Modellierungswerkzeuge für Benutzungsschnittstellen mit internen Transformationen, die sich dem Zugriff und der Kontrolle des Entwicklers entziehen. Diese internen Modelle haben häufig ein XML-Schema. So kann es sinnvoll sein, XSLT zu benutzen um diese interne Repräsentation weiter zu verarbeiten um so den Funktionsumfang der Modellierungswerkzeuge zu erweitern oder andere Zielformate zu ermöglichen.

Die Wahl des richtigen Transformationsansatzes ist also stark vom Modell, dessen Repräsentation und dem angestrebtem Einsatz abhängig.

Ein Aspekt der sich bei allen untersuchten Transformationsansätzen als verbesserungswürdig herausstellte ist die Werkzeugunterstützung. So existieren für QVT bisher nur Implementierungen von Teilen des Standards, für XSLT existieren häufig nur kommerzielle Werkzeuge. Der Einsatz der vorliegenden ATL-Implementierung als Eclipse-Plugin war jedoch insofern problematisch, als das häufig sehr unverständliche Fehlermeldungen angezeigt wurden.

7 Literatur

Literatur

[APQAS02] Ali, M. F. ; Pérez-Quiñones, M. A. ; Abrams, M. ; Shell, E.: Building Multi-Platform User Interfaces with UIML. In: *CADUI*, 2002, S. 255–266

[ATK03] Abed, M. ; Tabary, D. ; Kolski, C.: Using Formal Specification Techniques for the Modeling of Tasks and the Generation of Human-Computer User Interface Specifications. In: Diaper, D. (Hrsg.) ; Stanton, N. (Hrsg.): *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates, 2003, S. 503 – 529

[BBSS02] Bergman, L. D. ; Banavar, G. ; Soroker, D. ; Sussman, J. B.: Combining Handcrafting and Automatic Generation of User-Interfaces for Pervasive Devices. In: *CADUI*, 2002, S. 155–166

[Bis06] Bishop, J.: Multi-platform user interface construction: a challenge for software engineering-in-the-small. In: *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA : ACM, 2006. – ISBN 1–59593–375–1, S. 751–760

[BS02] Binnig, C. ; Schmidt, A.: Development of a UIML renderer for different target languages: experiences and design decisions. (2002)

[CCT+03] Calvary, G. ; Coutaz, J. ; Thevenin, D. ; Limbourg, Q. ; Bouillon, L. ; Vanderdonckt, J.: A Unifying Reference Framework for multi-target user interfaces. In: *Interacting with Computers* 15 (2003), Nr. 3, S. 289–308

[CH06] Czarnecki, Krzysztof ; Helsen, Simon: Feature-based survey of model transformation approaches. In: *IBM Systems Journal* 45 (2006), Nr. 3, S. 621–646

[Hub08] Huber, Philipp: *The Model Transformation Language Jungle - An Evaluation and Extension of Existing Approaches*, Technische Universität Wien, Masters Thesis, 2008

[KM05] Kempa, M. ; Mann, Z. A.: Model Driven Architecture. In: *Informatik Spektrum* 28 (2005), Nr. 4, S. 298–302

[LJMR09] López-Jaquero, Víctor ; Montero, Francisco ; Real, Fernando: Designing user interface adaptation rules with T: XML. In: *IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces*. New York, NY, USA : ACM, 2009, S. 383–388

[Mei10] Meixner, G.: *Entwicklung einer modellbasierten Architektur für multimodale Benutzungsschnittstellen*, Fachbereich Maschinenbau und Verfahrenstechnik, TU Kaiserslautern, PhD-Thesis, 2 2010. – 201 S

Kriterium	SmartQVT	ATL	XSLT
Transformationsregeln			
Bedingungen	Ja	Ja	Ja
Zwischenstrukturen	Ja	Ja	Nein
Reflektion	Ja	Nein	Nein
Parametrisierung			
Kontrollparameter	Ja	Ja	Nein
Generics	Nein	Nein	Nein
Regeln höherer Ordnung	Ja	Ja	Nein
Koordination der Regelanwendung			
Form			
Implizit	Nein	Ja	Ja
Explizit	Ja	Ja	Nein
Regeliteration	Rekursion	Rekursion	Rekursion
Regelorganisation			
Modularisierung	Ja	Ja	Nein
Organisationsstruktur	Unabhängig	Quellorientiert, Unabhängig	Quellorientiert
Beziehung zwischen Quelle und Ziel			
New Target	Ja	Ja	Ja
Existing Target	Nein	Ja	Nein
Aktualisierung	Nein	Nein	Nein
In Place	Nein	Ja	Nein
Direktionalität	uni	uni	uni (multi)
Tracing	Ja	Ja	Nein
Erstellung	Automatisch	Automatisch	-
Speicherort	Separat	Separat	-
Weitere Kriterien			
Deklarativ	Nein	Ja	Ja
Imperativ	Ja	Ja	Ja
Modell-Transformation	Ja	Ja	(Ja)
XML-Transformation	Nein	Nein	Ja
Code-Transformation	Nein	Nein	Nein
Code-Generierung	Nein	Nein	(Ja)
Erweiterbar	Ja	Ja	Nein

Tabelle 1: Überblick über die Ergebnisse der Auswertung der Anwendung der Taxonomie auf die Transformationssprachen

[MG09] Meixner, G. ; Görlich, D.: Eine modellbasierte Architektur für den Useware-Engineering Prozess. In: *GI Jahrestagung*, 2009, S. 2784–2789

[MM03] Miller, J. ; Mukerji, J.: MDA Guide Version 1.0.1 / Object Management Group (OMG). 2003. – Forschungsbericht

[Mol04] Molina, P. J.: A Review to Model-Based User Interface Development Technology. In: *MBUI*, 2004

[NC04] Nunes, N. J. ; Campos, P. F.: Towards Usable Analysis, Design and Modeling Tools. In: *MBUI*, 2004

[OMG08] OMG: *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification.* 2008

[Sch07] Schaefer, R.: A survey on transformation tools for model based user interface development. In: *HCI'07: Proceedings of the 12th international conference on Human-computer interaction.* Berlin, Heidelberg : Springer-Verlag, 2007. – ISBN 978–3–540–73104–7, S. 1178–1187

[WG08] Wilde, Erik ; Glushko, Robert J.: Document design matters. In: *Commun. ACM* 51 (2008), Nr. 10, S. 43–49. – ISSN 0001–0782