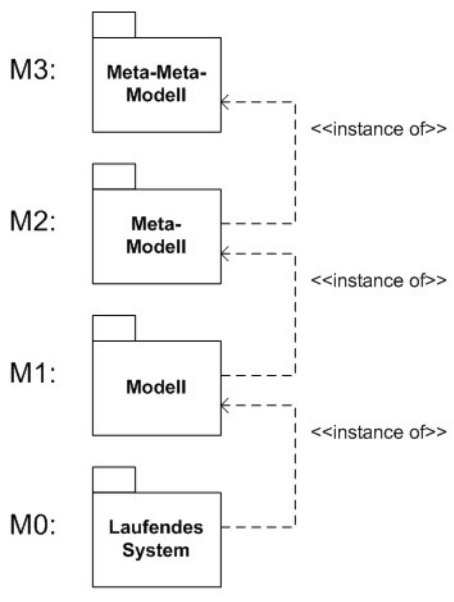
informatiCup
Der GI-Wettbewerb für Studierende

# informatiCup 2009 • Aufgabe 3

# Modelltransformationen

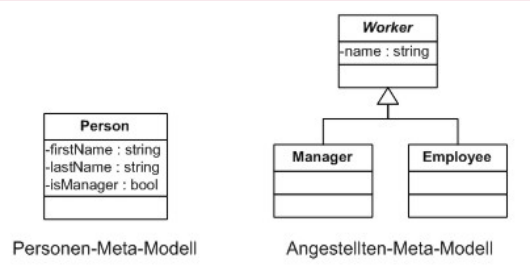
## Einführung

Im Rahmen der modellgetriebenen Softwareentwicklung werden Modelle als primäre Entwicklungsartefakte verwendet. Diese beschreiben die Struktur und das Verhalten der zu entwickelnden Software. Mit Hilfe von Codegeneratoren können dann große Teile des Quellcodes aus den Modellen generiert werden. Dadurch reduziert sich der Implementierungsaufwand und die Qualität des Codes steigt. Hierbei spielen auch Domänen-spezifische Modellierungssprachen (DSML) eine große Rolle. DSMLs sind Modellierungssprachen, die auf eine einzelne, sehr genau umrissene Anwendungsdomäne zugeschnitten sind. Dadurch abstrahieren sie sehr stark von der Implementierung und sind für Domänenexperten besser benutzbar als universelle Modellierungssprachen, wie UML. Aus diesen sehr abstrakten Modellen werden im Laufe der Entwicklung Schritt für Schritt detailliertere Modelle abgeleitet, um schließlich Quellcode zu generieren. Für diese Ableitungsschritte werden Modelltransformationssysteme verwendet. Diese erzeugen aus einem Quellmodell auf Basis einiger Transformationsregeln ein Zielmodell. Die Transformationsregeln beschreiben jeweils, wie ein bestimmtes Quellelement in ein Zielelement überführt wird. Allerdings ist zumeist Expertenwissen über die Modelltransformationssprache notwendig, bevor die Transformationsregeln aufgestellt werden können. Modelltransformationen werden auf Ebene der Meta-Modelle definiert, während die Transformation auf den Instanzen der Meta-Modelle, also den Modellen selbst, ausgeführt werden. Ein Meta-Modell beschreibt die Elemente und deren Beziehungen, die in einem Modell auftauchen können. Das Modell ist dann eine Instanz des Meta-Modells. Das Meta-Modell wird wiederum durch ein Meta-Meta-Modell definiert. Dieses beschreibt die Elemente, aus denen ein Meta-Modell bestehen kann. Daraus ergibt sich ein Meta-Modellierungs-Stack mit vier Ebenen, wie er von der Object Management Group definiert wird:

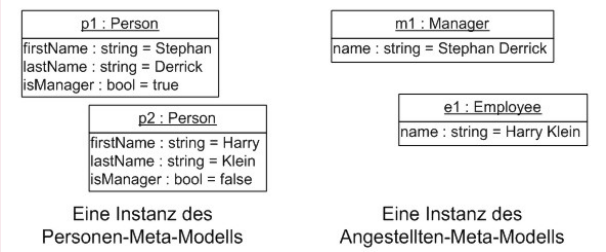



Wichtigster Vertreter eines Meta-Meta-Modells ist MOF (Meta Object Facility) sowie dessen Imple-

mentierung, das Ecore-Meta-Meta-Modell, das von EMF (Eclipse Modeling Framework) bereitgestellt wird. Auf der Ebene M2 befinden sich die Meta-Modelle der bekannten Modellierungssprachen, z.B. das UML-Meta-Modell, sowie die Meta-Modelle selbst-definierter Sprachen. Auf Ebene M1 finden sich konkrete Modelle, z.B. ein konkretes Klassendiagramm. Der untersten Ebene M0 werden schließlich die Objekte im Speicher eines laufenden Systems zugeordnet. Für die im weiteren Verlauf verwendeten Beispiele werden die folgenden beiden Meta-Modelle benutzt:




Das Personen-Meta-Modell enthält lediglich die Klasse Person, während das Angestellten-Meta-Modell eine einfache Vererbungsbeziehung enthält. Die Klasse Worker ist abstrakt, hiervon kann es also keine Instanzen geben. Die beiden anderen Klassen erben von Worker das Attribut name. Gültige Modelle zu diesen Meta-Modellen sind z.B.:



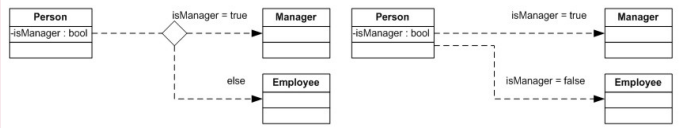

## Aufgabenstellung

Im Rahmen dieser Aufgabe soll ein Editor entwickelt werden, mit dem einfache Transformationen grafisch definiert werden können. Daraus sollen dann Transformationsregeln für ein Modelltransformationssystem abgeleitet werden können, ohne dass spezielles Wissen über dieses Transformationssystem notwendig ist. Ëinfach"bedeutet hier, dass nur 1-zu-1-Abbildungen unterstützt werden müssen, d.h. ein Element des Quellmodells wird auf ein Element des Zielmodells abgebildet, nicht auf mehrere. Der Editor bekommt das Quell- und Zielmetamodell als Eingabe[1] und soll deren Elemente anzeigen, z.B. in einer klassendiagram-ähnlichen Darstellung. Mit Hilfe von Verbindungslinien sollen die Quellelemente den Zielelementen zugeordnet werden können. Die folgenden Abbildungen dienen nur der Illustration und sind keine Vorschrift, wie der zu entwickelnde Editor aussehen muss.

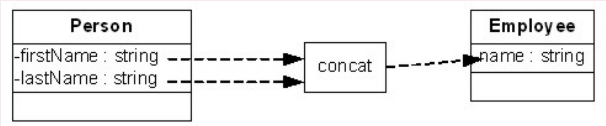
Hier wird ein Element vom Typ Person in ein Element vom Typ Employee transformiert. Der Wert des Attributes firstName wird einfach kopiert. Bei der Ausführung dieser Transformationsregel muss

[1] Es ist auch möglich, dass Quell- und Zielmetamodell identisch sind.


also für jedes im Modell vorhandene Element vom Typ Person ein neues Element vom Typ Employee erzeugt werden, dessen Attribut name den Wert des Attributs firstName der Person bekommt. Weiterhin soll der Editor Funktionsbausteine bieten, mit denen Bedingungen formuliert, Konstanten definiert, oder bestimmte Operationen ausgeführt werden können.




In diesem Beispiel soll ein Quellelement vom Typ Person in ein Element vom Typ Manager transformiert werden, wenn das Attribut isManager den Wert true hat. Ansonsten wird es in ein Element vom Typ Employee transformiert.




Hier wird ein Funktionsbaustein concat verwendet, um die Werte der Attribute firstName und lastName zu verketten. Das Resultat wird dem Attribut name des Employee zugewiesen. Die folgenden Funktionsbausteine müssen unbedingt unterstützt werden:

- Konstanten, um z.B. einem Attribut einen festen Wert zuzuordnen.
- Bedingungen, um die Transformation eines Elements von bestimmten Bedingungen abhängig zu machen.
- Verkettung, Zerlegung von Zeichenketten
- Addition, Subtraktion, Multiplikation, Division von numerischen Attributen
- Value-Maps: Eine Value-Map definiert Schlüssel-Wert-Paare. Wenn ein Attribut im Quellmodell einen bestimmten (Schlüssel-)Wert hat, wird dem korrespondierenden Attribut im Zielmodell der Wert aus der Value-Map zugewiesen, der zu dem Schlüssel gehört. Für den Fall, dass ein Schlüsselwert in der Value-Map nicht enthalten ist, soll ein Standardwert definiert werden können.

Darüber hinaus können natürlich weitere sinnvoll erscheinende Funktionsbausteine implementiert werden. Die Elemente der grafischen Transformationsspezifikationssprache und deren Bedeutung müssen

außerdem dokumentiert werden. Aus der grafischen Spezifikation sollen dann Modelltransformationsregeln abgeleitet werden, mit deren Hilfe ein Quellmodell in ein entsprechendes Zielmodell transformiert werden kann. Quell- und Zielmodell müssen natürlich Instanzen der entsprechenden Meta-Modelle sein. Für diese Aufgabe sollen Transformationsregeln für ATL (ATLAS Transformation Language) oder XSLT generiert werden können[2]. Die Kopplung an ein spezielles Transformationssystem sollte aber möglichst lose sein, sodass die Regelgenerierung für andere Transformationssysteme sehr leicht hinzugefügt werden kann. Da die Transformationsregeln üblicherweise als Textdateien vorliegen müssen, können Codegenerierungsframeworks wie z.B. Java Emitter Templates oder Xpand verwendet werden, um diese Textdateien zu erzeugen. Die grafische Spezifikation soll weiterhin unabhängig von irgendeinem Transformationssystem als Datei speicherbar sein. Als technologische Grundlage empfiehlt sich die Verwendung des Eclipse Modeling Frameworks, da dieses ebenfalls von vielen Modelltransformationssystemen unterstützt wird.

## Teilaufgaben

Im ersten Teil der Aufgabenstellung geht es zunächst darum, sich mit einem Meta-Modellierungs-Framework (z. B. EMF), dessen Meta-Modell-Spezifikationsformat (z. B. ECore oder XML Schema) und vorhandenen Transformationswerkzeugen (z. B. EcoreToEcore Mapping-Editor, ATL) vertraut zu machen. Anschließend soll ein Meta-Modell für die grafische Transformationssprache entwickelt werden. Dieses muss mächtig genug sein, um die grafischen Modelltransformationsregeln inklusive der benutzbaren Funktionsbausteine abzubilden. Ob dieses Transformationsmetamodell bereits Informationen über die konkreten Positionen der grafischen Elemente beinhaltet, ist freigestellt[3]. Der erste Teil der Aufgabenstellung ist erfüllt, wenn auf Basis einer konkreten Instanz dieses Transformations-Meta-Modells, d.h. einer Transformationsspezifikation, automatisiert ATL- oder XSLT-Transformationsregeln abgeleitet werden können, die ein Quellmodell in ein Zielmodell überführen. Im zweiten Teil der Aufgabenstellung soll ein grafischer Editor implementiert oder ein vorhandener grafischer Editor erweitert werden, der es ermöglicht, Instanzen des Transformations-Meta-Modells grafisch zu erzeugen, sobald Quell- und Ziel-Meta-Modell bekannt sind.

## Hinweise

Die wichtigsten Bewertungskriterien für diese Aufgabe sind die Benutzerfreundlichkeit und die Erweiterbarkeit des grafischen Editors. Die Benutzerfreundlichkeit kann z.B. durch Funktionen verbessert werden, die an einer bestimmten Stelle nur solche Funktionsbausteine anbieten, die dort eingefügt werden können, oder Funktionen, die kleinere Korrektheitsprüfungen auf den Transformationsregeln durchführen und auf grobe Fehler hinweisen. Zur besseren Unterstützung der Definition von Transformationen zwischen ähnlichen Meta-Modellen kann auch eine Funktion eingebaut werden, die korrespondierende Elemente auf Basis einer Heuristik automatisch einander zuordnet, z.B. Elemente mit gleichem oder ähnlichem Namen. Die Erweiterbarkeit betrifft im Wesentlichen die Möglichkeit, Unterstützung für andere Transformationssysteme hinzuzufügen, ohne dass tiefgreifende Veränderungen

[2] Das bedeutet, dass nur eines der beiden Transformationssysteme unterstützt werden muss.
[3] Bei grafischen Editoren auf Basis von GMF werden die Layout-Informationen getrennt vom eigentlichen Modell gespeichert.

am Editor vorgenommen werden müssen. Ein weiterer Anwendungsfall für solch einen Editor ist die Meta-Modell-Evolution. Wenn sich Meta-Modelle verändern, können deren Instanzen ungültig werden. Mit Hilfe von Modelltransformationen können die veralteten Modelle dann aktualisiert werden. Als Orientierung, wie ein Editor aussehen kann, wie er hier entwickelt werden soll, lohnt ein Blick auf Altova MapForce.

## Referenzen

1. Altova MapForce http://www.altova.com/de/produkte/mapforce/daten_integration.html
2. Atlas Transformation Language http://www.eclipse.org/m2m/atl/
3. Eclipse Modeling Framework http://www.eclipse.org/modeling/emf/
4. Graphical Modeling Framework http://www.eclipse.org/modeling/gmf/
5. Java Emitter Templates http://www.eclipse.org/modeling/m2t/?project=jet
6. XML Schema http://www.w3.org/XML/Schema
7. XML Schema To Ecore Mapping http://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf
8. Xpand http://www.eclipse.org/modeling/m2t/?project=xpand
9. XSLT http://www.w3.org/TR/xslt20/