von wolfgang neuhaus und boris holzer

MODELLGETRIEBENE SOFTWAREENTWICKLUNG: ALLES UML, ODER?

Bei modellgetriebener Softwareentwicklung werden aus kompakten Modellbeschreibungen lauffähige Softwareprogramme generiert. Solche Modellbeschreibungen werden meist mit Hilfe der Unified Modeling Language (UML) erstellt. Eine Beschränkung auf die UML ist aber nicht sinnvoll, da es auch andere direkt nutzbare Modellquellen gibt. Solche stellt dieser Artikel anhand von Beispielen aus dem Bereich grafischer Benutzerschnittstellen vor. Insbesondere eine Kombination aus UML- und GUI-Modell erweist sich als sehr nützlich.

Modellgetriebene Softwareentwicklung gewinnt immer mehr an Bedeutung und wird zunehmend öffentlich diskutiert – nicht zuletzt lesen Sie deshalb gerade diese Sonderausgabe zum Thema. Aus kompakten Modellbeschreibungen sollen bei der modellgetriebenen Softwareentwicklung (teil-)automatisch lauffähige Softwareprogramme generiert werden. Häufig jedoch wird die Diskussion über die Beschreibungssprache von Modellen auf die Nutzung der Unified Modeling Language (UML) reduziert – unter anderem weil auch die Object Management Group (OMG) mit ihren MDA-Spezifikationen diese Sicht prägt.

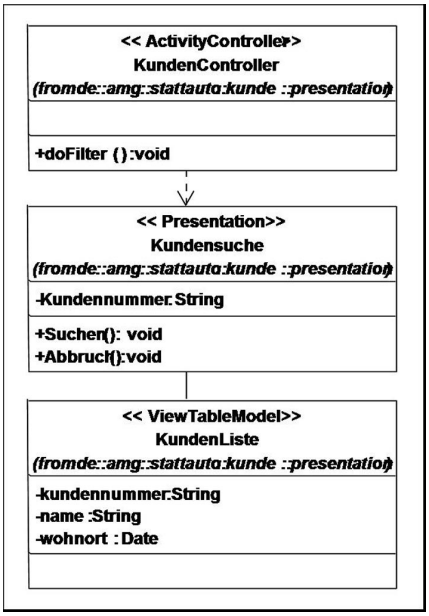

Abb. 1: UML-Klassendiagramm für die in Abb. 2 dargestellte Kundensuche

Die Praxis hingegen zeigt, dass diese Beschränkung die Sicht auf pragmatische, direkt nutzbare Modellquellen verstellt. Schlimmer noch: für einige Anwendungsbereiche eignet sich die UML in der heutigen Form nicht besonders gut. Und das führt zu Modellen, die nicht optimal auf das betrachtete Problem zugeschnitten sind.

Dieser Artikel verdeutlicht diese Problematik anhand von Beispielen aus dem Bereich grafischer Benutzerschnittstellen. Nach einer Einleitung wird ein Lösungsansatz diskutiert und anhand von Beispielen vertieft. Ziel ist es, eine Entwicklungsmethodik aufzuzeigen, die in einem „best of breed"-Ansatz die Kombination unterschiedlicher Beschreibungssprachen erlaubt.

Problemstellung

Sehen wir uns zunächst einmal die grundsätzliche Problematik anhand eines Beispiels an: Das UML-Klassendiagramm in Abbildung 1 ist der Versuch, den HTML-Dialog aus Abbildung 2 zu beschreiben. Im Diagramm sind allerdings nur strukturelle Eigenschaften wie Eingabefelder, Auswahllisten oder Funktionsknöpfe beschrieben. Für eine vollständige Beschreibung fehlen weitere, rein für das Layout notwendige Informationen wie Farben, Größen, Zeichensätze, Grafiken oder Positionen.

Obwohl es sich um einen sehr einfachen Dialog handelt, wird das Klassendiagramm bereits sehr unübersichtlich, wenn man versucht, diese Informationen zusätzlich unterzubringen. Möglichkeiten dazu bieten beispielsweise Tagged Values – das sind Schlüssel/Wert-Paare, von denen beliebig viele jedem UML-Modellelement zugeordnet werden können. Darüber hinaus hat man bei der Erstellung eines solchen Modells keine Möglichkeit, das Endergebnis direkt grafisch zu prüfen. Verbessern lässt sich diese Situation nur durch Abstraktion, die durch stark standardisierte Dialogbeschreibungen erreicht werden kann.

Allerdings lassen sich nicht für alle Anwendungsbereiche Dialoge derart standardisieren. Daher haben sich für die Erstellung grafischer Benutzeroberflächen GUI Builder durchgesetzt – Spezialwerkzeuge, die eine Bearbeitung im WYSI-

die autoren

Wolfgang Neuhaus (E-Mail: Wolfgang.Neuhaus@itemis.de) ist Geschäftsführer und Gründer der itemis GmbH & Co. KG, einem innovativen Beratungshaus für objektorientierte Softwareentwicklung. Seine Spezialgebiete sind objektorientierte Vorgehensmodelle, Softwarearchitektur und MDA. Darüber hinaus ist er Mitbegründer der Architecture Management Group (www.architecture-management-group.de).

Boris Holzer (E-Mail: boris.holzer@itemis.de) ist Senior Berater und Coach bei der itemis GmbH & Co. KG. Seine Schwerpunkte sind die Konzeption J2EE basierter Systeme, modellgetriebene generative Softwareentwicklung und das Coaching von Entwicklungsteams. Zuletzt führte er die im Artikel geschilderte Lösung bei der Continentale Versicherung ein.

Abb. 2: HTML-Dialog zur Kundensuche

WYG-Modus erlauben. Mit ihnen lässt sich der statische Teil einer Benutzeroberfläche sehr gut und effizient beschreiben. Leider fehlt diesen Werkzeugen in der Regel aber eine adäquate Unterstützung zur Beschreibung der dynamischen Aspekte – also der Navigationsfolge und der Beschreibung der dazu notwendigen Zustandsübergänge. Gerade in diesem Bereich bietet die UML mit Aktivitäts- und Zustandsdiagrammen gute Hilfsmittel, um eine übersichtliche und kompakte Beschreibung für die dynamischen Aspekte erreichen zu können. Abbildung 3 zeigt ein einfaches Beispiel.

In vielen Projekten haben wir die Erfahrung gemacht, dass solche Diagramme auch mit fachseitigen Ansprechpartnern diskutiert werden können. Gemeinsam mit Skizzen zu den einzelnen Dialogen entsteht ein sehr detailliertes und vollständiges Bild der späteren Anwendung. Ideal wäre es also, diese beiden Beschreibungsformen für Statik und Dynamik kombiniert nutzen zu können, um daraus die Präsentationsschicht einer Anwendung generieren zu können.

UML-Modell und GUI-Modell kombiniert

Damit diese Kombination gelingen kann, müssen einige Randbedingungen beachtet werden. Der verwendete GUI Builder sollte ein maschinenlesbares Speicherformat für die Dialogbeschreibung unterstützen. In unserem Beispiel haben wir für die Beschreibung der Dialoge den Casabac GUI Builder verwendet (siehe Abb. 4). Er schreibt alle für die statische Beschreibung notwendigen Informationen in eine XML-Datei. Um diese Informationen nun mit den dynamischen Informationen aus der „UML-Welt“ verbinden zu können, brauchen wir noch zwei Dinge: Einen Parser, der das Speicherformat lesen kann, und einen Generator, der UML-Modell und GUI-Beschreibung parallel einlesen und transformieren kann. Dies führt zu einem Zusammenspiel unterschiedlicher Tools, das in Abbildung 5 skizziert ist. Jetzt fehlt noch der Bezug zwischen den beiden Modellen. Hier gibt es unterschiedliche Möglichkeiten, die wir im nächsten Abschnitt diskutieren wollen.

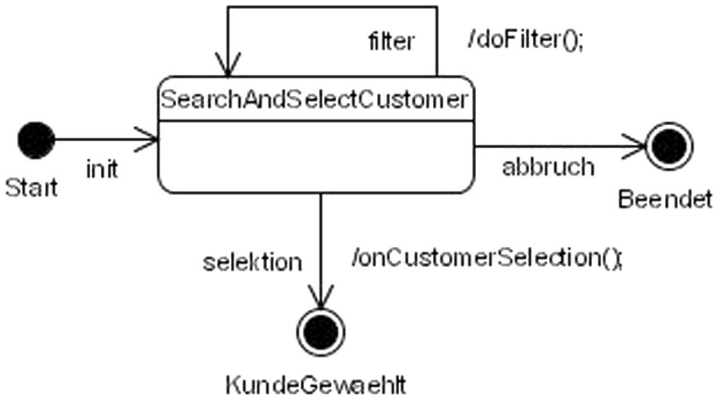

Abb. 3: Beispiel eines Zustandsdiagramms

Für die Kombination der beiden Welten müssen wir uns zunächst einmal mit den Metamodellen der gewählten Sprachen beschäftigen. Da wir uns zur Beschreibung der Dynamik für die „UML-Welt“ entschieden haben, ist das Metamodell das der UML für Zustandsdiagramme. Abbildung 6 zeigt einen Ausschnitt daraus. Das Metamodell der Casabac-Dialoge repräsentiert die unterschiedlichen Widget-Typen, die Casabac unterstützt. Abbildung 7 zeigt auch daraus einen Ausschnitt.

In den Tools – UML-Werkzeug und GUI Builder – werden Instanzen dieser Metamodelle erzeugt und bearbeitet. Anschließend werden sie in unterschiedlichen Formaten exportiert: XMI und Casabac-XML. Diese Exportformate stellen jeweils eine konkrete Syntax zu Repräsentation der Modelle dar. Moderne MDSD-Generatoren arbeiten nicht direkt auf der konkreten Syntax – beispielsweise mit XSL/XSLT –, sondern erzeugen zunächst selber eine Repräsentation der eingelesenen konkreten Syntax in Form eines instanzierten Metamodells. Die Transformation in den Quellcode erfolgt dann durch Schablonen, die ausschließlich auf das instanzierte Metamodell zugreifen. Damit wird die Transformation von der konkreten Syntax entkoppelt. Abbildung 8 zeigt die Arbeitsweise eines solchen Generators.

Damit bei der Generierung die richtige Steuerungsinformation zur passenden Maske zugeordnet werden kann, muss die Information über die Zuordnung im instanzierten Metamodell vorhanden sein. Damit der Instanziator die Metamodellinstanzen richtig „verheiraten“ kann, muss in den Modellen eine der drei folgenden Varianten verwendet werden:

1) Gleiche/redundante Modellelemente in beiden Modellen,
2) Proxys als Stellvertreter für Modellelemente oder
3) Referenzen auf Modellelemente.

Bei Variante 1 werden in beiden zu verbindenden Modellen redundante, gleiche Modellelemente gepflegt und erst bei der Instanzierung der Metamodellinstanzen zu einem Element zusammengefügt. Damit können in beiden Modellen (unter-▸

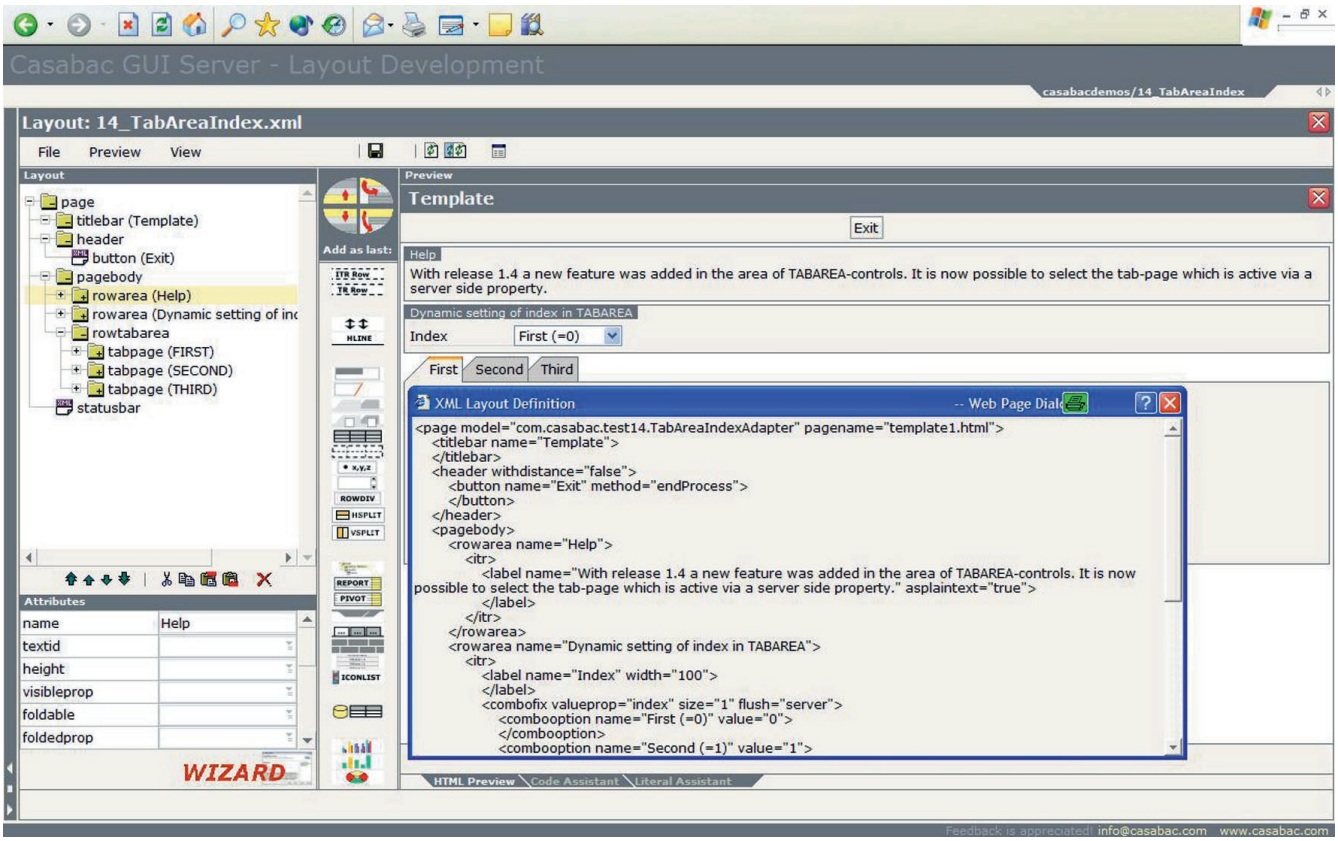

Abb. 4: Beschreibung der Dialoge mit Hilfe des GUI Builders Casabac

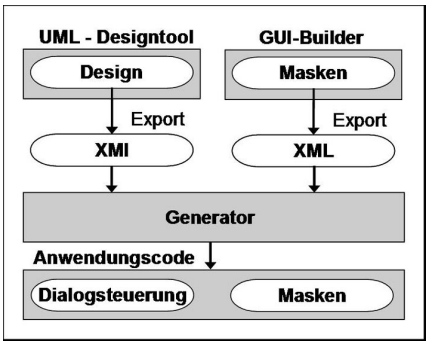

Abb. 5: Zusammenspiel der Werkzeuge

schiedliche) Assoziationen zu weiteren Modellelementen aufgebaut werden, die somit die Verbindung zwischen beiden Modellen darstellen. Dieser Weg ist für unsere Zielsetzung, UML-Modelle mit GUI-Modellen zu verschmelzen, nicht gut geeignet, da es bedeuten würde, entweder im Casabac GUI Builder UML-Elemente anlegen zu müssen, oder im UML-Werkzeug Casabac Controls beschreiben zu können. Für beide Wege wäre eine spezifische Erweiterung des jeweiligen Werkzeugs notwendig.

Variante 2 funktioniert ähnlich, mit dem einen Unterschied, dass statt gleicher Modellelemente Stellvertreter (Proxys) genutzt werden. Auch diese werden bei der Metamodellinstanzierung durch die tatsächlichen Modellelemente aus dem jeweils anderen Modell ersetzt. Dieser Weg würde im UML-Werkzeug noch ohne spezifische Erweiterung funktionieren, da man beispielsweise speziell ausgezeichnete Klassen als Stellvertreter verwenden könnte. Im Casabac GUI Builder müssten wir allerdings auch dafür wieder eine spezifische Erweiterung des Werkzeugs vornehmen.

Die am einfachsten umzusetzende Variante ist damit die Variante 3 – die Referenzierung. Referenzen können beliebige, eindeutig identifizierende Merkmale von Modellelementen sein – beispielsweise eindeutige Namen oder Modell-IDs. Wie bei Fremdschlüsseln in relationalen Datenbanken können so Verweise auf Modellelemente definiert werden. Da für diesen Weg keine spezifische Werkzeuganpassung erforderlich ist, haben wir uns für Referenzen entschieden und verwenden dazu feste Namenskonventionen.

Unabhängig von der Wahl der Variante zur Verbindung der Modelle ist die

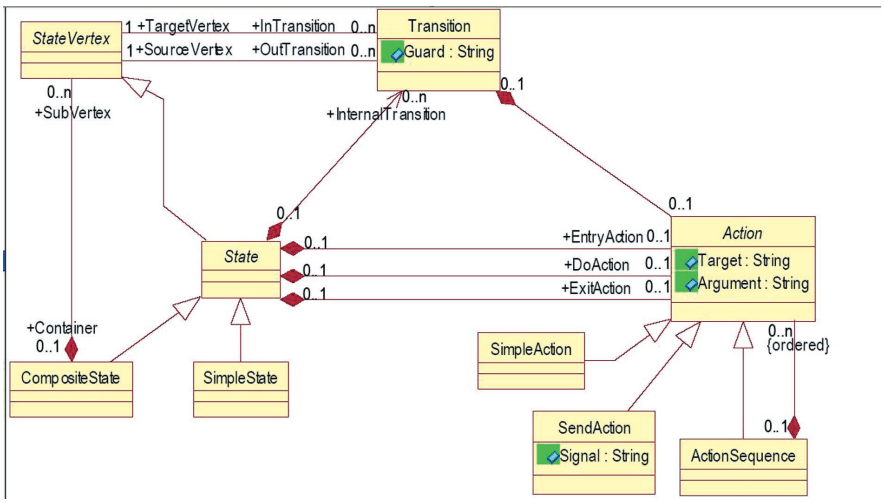

Abb. 6: Zustandsautomat des Metamodells

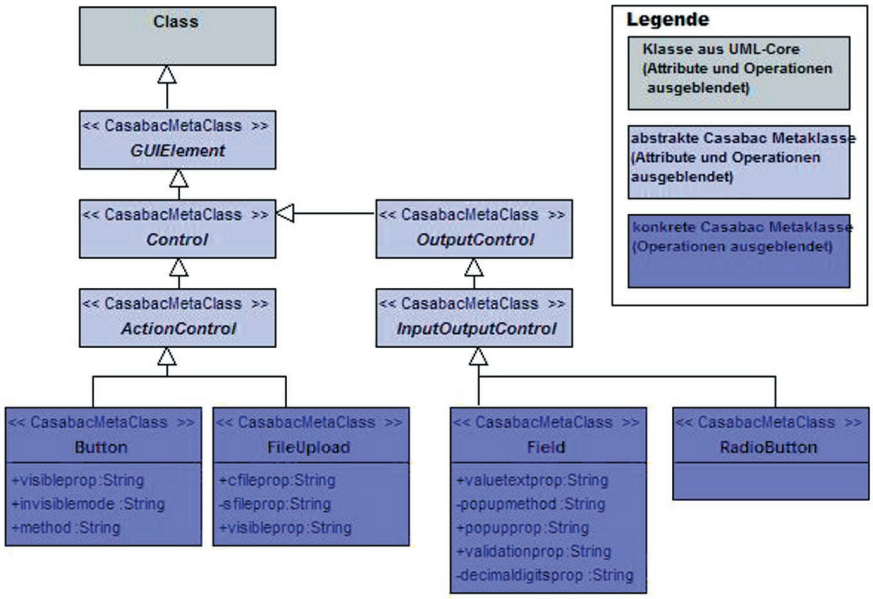

Abb. 7: Metamodell der Casabac-Dialoge

Validierung dieser Zuordnung von zentraler Bedeutung. Da wir unterschiedliche Tools kombinieren und keine durchgängige Bearbeitung aller Modellinformationen in einem Werkzeug verwenden, müssen Fehler bei der Zuordnung vor der Generierung überprüft und gemeldet werden. Fehlermeldungen sollten „sprechend" sein, damit falsche Zuordnungen schnell identifiziert werden können. So kann bereits vor der Generierung sichergestellt werden, dass es keine falschen Referenzen gibt, die zu einer fehlerhaften Generierung und damit zu nicht lauffähigen Programmen führen. Letztlich sichern wir so die „referenzielle Integrität".

Die notwendige Generator-Technologie

Ein MDSD-Generator, mit dem man den skizzierten Lösungsansatz beschreiten will, sollte also folgende Eigenschaften haben:

- ein flexibles, programmierbares Metamodell,
- eine Möglichkeit zur parallelen Instanzierung des Metamodells aus unterschiedlichen Modellquellen,
- eine Unterstützung zur Validierung von Modellierungsrichtlinien,
- eine Transformation auf Basis des instanzierten Metamodells.

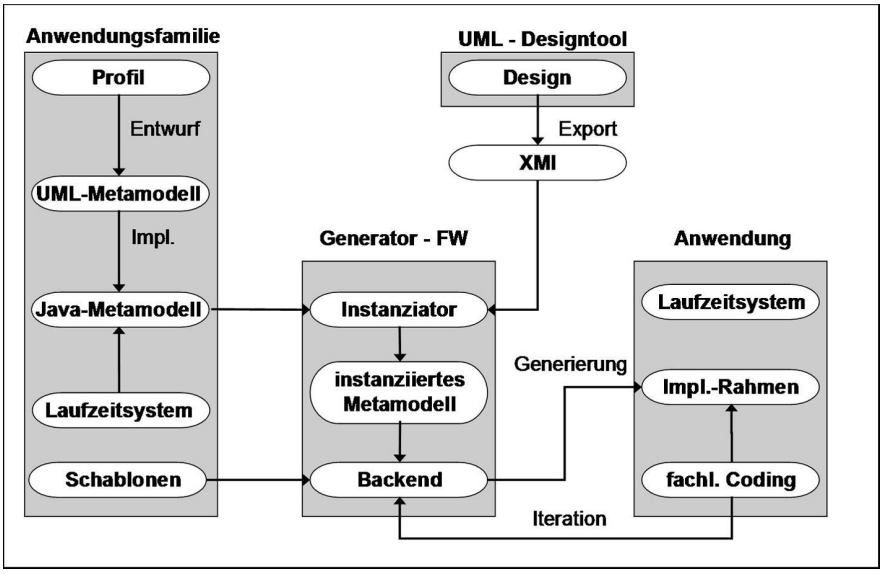

Abb. 8: MDSD-Generator-Framework

Das flexible, programmierbare Metamodell benötigen wir, um dem Generator die Sprachelemente (siehe Abb. 6 und 7) bekannt zu machen und um die Transformation von der konkreten Syntax trennen zu können.

Die parallele Instanziierung des Metamodells aus unterschiedlichen Modellquellen verwenden wir, um sowohl das UML-Modell mit den dynamischen Aspekten wie auch die GUI-Beschreibung mit den statischen Aspekten verarbeiten zu können. Um die Zuordnung prüfen zu können, setzen wir die Validierung der Modellierungsrichtlinien – in unserem Fall die Referenzen auf Modellelemente – ein. Die Transformation muss dann letztlich auf das instanzierte Metamodell zugreifen können.

Das von uns eingesetzte open Generator Framework unterstützt diese Eigenschaften, ist als Opensource verfügbar und auf Source Forge zu finden. Eine detaillierte Darstellung der Arbeitsweise mit dem open Generator Framework für die Verbindung von UML-Modell und Casabac-GUI zeigt Abbildung 9. Wir verwenden dort einen Kompositum-Instanziator, der die Instanzierung eines Metamodells aus mehreren Modellquellen unterstützt – in unserem Fall aus der XMI-Datei, die die UML-Information enthält, und der Casabac-XML-Datei für die Masken.

Mit diesem Handwerkszeug sind wir in der Lage, die Informationen aus beiden Tools – UML-Werkzeug und Casabac GUI Builder – einzulesen und über Generator-Schablonen in ausführbaren Programmcode zu transformieren. Diese Werkzeugkombination erlaubt es uns nun, sowohl die grafische Darstellung und Struktur unserer HTML-Dialoge wie auch die Ablaufsteuerung in einer effektiven Form zu beschreiben und die Präsentationsschicht einer Anwendung vollständig daraus zu generieren.

Fazit

Was haben wir jetzt erreicht? Wir sind in der Lage, die jeweiligen Vorteile entsprechender Spezialwerkzeuge zur Beschreibung unterschiedlicher Aspekte einer Anwendung nutzen und miteinander kombinieren zu können. Dabei entstehen keine Einschränkungen für die Generierung, da durch die Entkopplung von Transformation und konkreter Syntax die Generator-Schablonen nichts von den Modellquellen „wissen" müssen. Dadurch wird eine problembezogene, gezieltere Werkzeug-

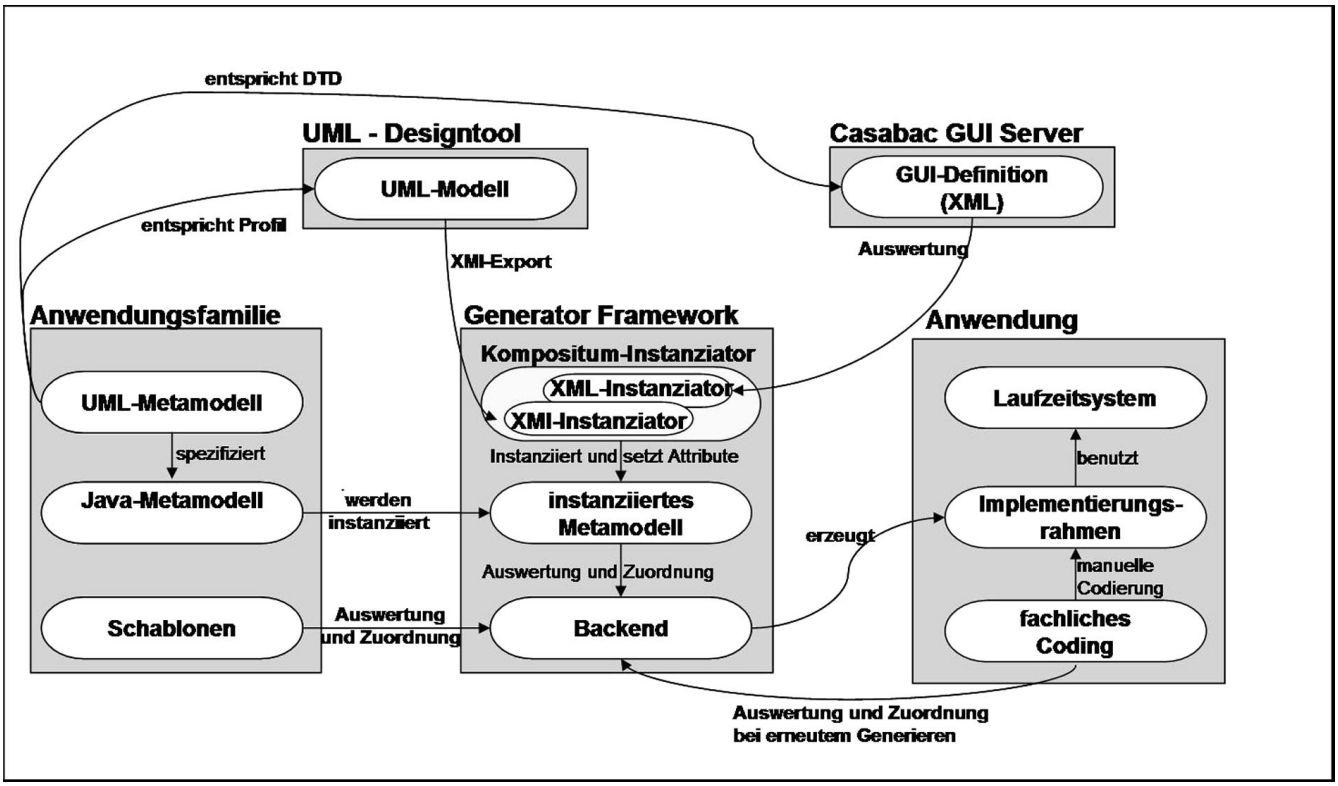

Abb. 9: Arbeitsweise mit dem open Generator Framework für die Verbindung von UML-Modell und Casabac-GUI

unterstützung ermöglicht, die die modellgetriebene, generative Softwareentwicklung von der Beschränkung auf UML als Modellierungssprache befreit. Auch für Migrationsprojekte ist dieser Ansatz eine Chance, vorhandene Informationsquellen wie bestehenden Quellcode oder Datenbankschemata als Modellquellen bei der Generierung mit berücksichtigen und verwerten zu können.

Aber auch die Nachteile wollen wir nicht verschweigen. Denn die fehlende Durchgängigkeit in der Werkzeugunterstützung führt zum einen zu weniger Komfort in der Bedienung, zum anderen zu einer erhöhten Fehleranfälligkeit bei der Referenzierung. Unserer Erfahrung nach überwiegen jedoch die Vorteile, wenn man die Nachteile durch eine intensive Validierung ausgleicht.

Links

Open Generator Framework, siehe: http://sourceforge.net/projects/architecturware
Informationen zu Casabac, siehe: http://www.casabac.de
Weitere MDSD/MDA-Publikationen, siehe:http://www.itemis.de/publikationen.php
MDA Informationen der OMG, siehe: http://www.omg.org/mda
Eine sehr lesenswerte Einordnung des Themas: http://www.voelter.de/data/articles/MDSD.pdf