Ein Nachdruck aus JavaSPEKTRUM SYSTEMS Special

Schicht für Schicht

Generative Softwareentwicklung mit inkrementellen Vorgehensmodellen

Wolfgang Neuhaus

Iterative und inkrementelle Vorgehensmodelle der Softwareentwicklung lösen nun auch in großen Unternehmen das Wasserfallmodell ab. Das iterative und inkrementelle Vorgehen begünstigt Rapid Prototyping. Leider kommt es dabei häufig zu Wegwerf-Prototypen. Hier bietet die generative Softwareentwicklung einen Ausweg. Quellcode-Generatoren befreien den Entwickler von der Last stupider Routinetätigkeiten, Werkzeuge zur computerunterstützten Softwareentwicklung gewährleisten die Konsistenz zwischen Modell und Codierung. Mit diesen Hilfsmitteln können Softwareentwicklungsunternehmen iterative und inkrementelle Vorgehensmodelle sinnvoll einsetzen, um somit von diesem „Prinzip der kleinen Schritte" zu profitieren.

Iterative und inkrementelle Vorgehensmodelle wie RUP (Rational Unified Process, [Kru99]) und OEP (Object Engineering Process, [Oes00]) haben seit geraumer Zeit Einzug gehalten in die tägliche Arbeit von Softwareentwicklungsunternehmen und IT-Abteilungen großer Wirtschaftsunternehmen. Sie steigern messbar die Quote der erfolgreichen Projektdurchführungen gegenüber herkömmlichen Modellen wie dem Wasserfallmodell oder dem V-Modell.

Da iterative und inkrementelle Vorgehensmodelle geprägt sind durch kurze Abstimmungszyklen, in denen konkrete Projektergebnisse mit dem Kunden diskutiert und abgestimmt werden, gewinnt das Rapid Prototyping eine besondere Bedeutung. Gerade hier stehen aber die Softwareentwicklungsunternehmen vor einem Dilemma: Um in kurzer Zeit Prototypen erstellen zu können, die als Diskussions- und Abstimmungsgrundlage dienen sollen, werden häufig – gerade in den frühen Phasen – Ansätze verwendet, die zu Wegwerf-Prototypen führen.

Bei der Verwendung von Wegwerf-Prototypen entstehen jedoch zwei zentrale Nachteile: Zum einen werden Arbeitsergebnisse nicht weiterverwendet und somit Zusatzaufwand generiert, zum anderen wird Know-how in zusätzlichen Technologien aufgebaut, das ausschließlich zur Erstellung von Wegwerf-Prototypen verwendet wird. Ein gutes Beispiel ist die Verwendung von Delphi, um – kurzfristig betrachtet – schnell und komfortabel Benutzeroberflächen visualisieren zu können, anschließend die Umsetzung jedoch mit Java-Swing-Oberflächen vorzunehmen.

Andere Unternehmen sehen einen Ausweg in der Copy&Paste-Lösung, die eigentlich keine ist. Denn hier werden bestehende GUIs ähnlich gelagerter Projekte kopiert und etwas angepasst, wodurch ebenfalls wieder ein Ergebnis entsteht, das nicht als Grundstein für die spätere Realisierung verwendet werden kann. Allerdings kostet die Realisierung von Prototypen in der Zielarchitektur viel Zeit. Zeit, die man in der Abstimmungsphase mit dem Kunden häufig nicht hat.

Hier bietet die generative Softwareentwicklung einen Ausweg. Quellcode-Generatoren, die auf Basis von Modellinformationen standardisierte Komponenten der Zielarchitektur erzeugen, befreien den Entwickler von der Last stupider Routinetätigkeiten. Case-Tools, die Round Trip Engineering unterstützen, gewährleisten die Konsistenz zwischen Modell und Codierung. Ausgestattet mit diesen Hilfsmitteln, gelingt es Softwareentwicklungsunternehmen den Anforderungen durch den Einsatz iterativer und inkrementeller Vorgehensmodelle gerecht zu werden. Prototypen werden nicht weggeworfen, sondern stellen echte Inkremente auf dem Weg zur Zielerreichung dar.

In Ansätzen werden diese Mechanismen durch einige Case-Tools bereits unterstützt, um bspw. auf Basis von Mustern Code zu generieren. Besonders ausgeprägt ist diese Funktionalität im Bereich der Enterprise-JavaBeans (EJB). Als Beispiel sei hier TogetherJ genannt.

Der Artikel beschreibt die organisatorischen und technischen Voraussetzungen, die für eine Einführung einer generativen Softwareentwicklung gegeben sein müssen. Es wird eine Vorgehensweise zur Einführung generativer Softwareentwicklungsmethoden vorgeschlagen und anhand eines Beispiels konkretisiert. Die daraus entstehenden Vorteile und Möglichkeiten werden abschließend aufgezeigt.

Die organisatorischen Voraussetzungen und Rahmenbedingungen

Generative Softwareentwicklung lässt sich nicht „auf der grünen Wiese" beginnen. Es müssen einige organisatorische und technische Voraussetzungen und auch Rahmenbedingungen erfüllt sein, um sie auch effektiv einsetzen zu können. Betrachtet man die organisatorischen Voraussetzungen und Rahmenbedingungen, so betrifft dies vor allem die

- Abstraktionsfähigkeit der Mitarbeiter;
- die Existenz einer Wiederverwendungskultur;
- den Einsatz eines iterativen und inkrementellen Vorgehensmodells;
- den projektübergreifenden Einsatz von Architekturmustern;
- und die Unterstützung auf Management-Ebene.

Insbesondere die Abstraktionsfähigkeit der Mitarbeiter ist eine Rahmenbedingung, die sich nicht in harten Zahlen oder Messgrößen beschreiben lässt. Ein gutes Indiz für eine ausreichende Ausprägung ist der Grad der Wiederverwendung bzw. die Wiederverwendungskultur in der Entwicklung. Hat diese sich bereits so-

weit ausgeprägt, dass eine Framework-Entwicklung in Gang gesetzt wurde und ein stetiger Optimierungsprozess der Framework-Komponenten stattfindet, so sollte auch die Einführung der im Folgenden beschriebenen Vorgehensweise zur Einführung generativer Softwareentwicklung gelingen.

Die Verwendung eines iterativen und inkrementellen Vorgehensmodells ist eher eine unterstützende Rahmenbedingung, denn eine echte Voraussetzung. Denn die Einführung und Optimierung einer generativen Softwareentwicklung findet am besten iterativ und inkrementell statt, wie im Folgenden deutlich wird.

Eine herausragende Bedeutung liegt sicherlich in der projektübergreifenden Verwendung von Architekturmustern. Denn selbst beim wiederholten und projektübergreifenden Einsatz derselben Frameworks führt dies nicht automatisch zu denselben Architekturen. Als Beispiel sei hier ein Persistenz-Framework genannt, dass auf Client-Seite genauso zum Einsatz kommen kann, wie auch als serverseitige Lösung in Mehrschichten-Architekturen. Nur wenn auch auf der Ebene von Architekturmustern und Architekturen eine Wiederverwendung stattfindet, lassen sich die Potentiale generativer Softwareentwicklung richtig ausschöpfen. Denn dann können im fortwährenden Optimierungsprozess sowohl die Framework-Basis wie auch der Generator und die ihm zugrunde liegenden Modellinformationen verbessert und die resultierenden Synergieeffekte voll genutzt werden.

Abschließend soll auch noch der Blick auf die notwendige Management-Unterstützung gerichtet werden. Ähnlich wie die Einführung einer Wiederverwendungskultur und die Entwicklung und ständige Optimierung von Frameworks, erfordert auch die Einführung der im Folgenden vorgestellten Vorgehensweise bei der generativen Softwareentwicklung einen Initialaufwand, der den konkreten Projektaufwand bezogen auf ein einziges Projekt übersteigt. Die Entscheidung für die Einführung einer solchen Vorgehensweise sollte also auf einer Entscheidungsebene fallen, die für den entsprechenden Rückhalt in der Startphase sorgt. Ist diese allerdings erfolgreich durchgeführt worden, so wird sich die Frage nach dem Rückhalt nicht mehr stellen, denn dann kommen die Vorteile klar zum Ausdruck.

Die technischen Voraussetzungen

Aus den aufgeführten organisatorischen Voraussetzungen und Rahmenbedingungen leiten sich bereits einige technische Voraussetzungen ab. Die wichtigsten Voraussetzungen für eine erfolgreiche Einführung sind mit

- ▼ klar definierten Entwurfsmustern,
- ▼ einer stabilen technischen Architektur über mehrere Projekte und
- ▼ dem Einsatz von Frameworks

benannt. Klar definierte Entwurfsmuster sind die Voraussetzung, um stabile Generator-Vorlagen etablieren zu können. Auch stellen sie die Grundlage für die Modellierungsrichtlinien dar, die für den anschließenden Generierungsschritt berücksichtigt werden müssen. Nur wenn demjenigen, der das technische Design übernimmt, zu jedem Zeitpunkt bei der Modellierung bewusst ist, dass er definierte Entwurfsmuster verwenden muss, um über den Generator anschließend Implementierungsanteile generieren zu können, lässt sich die erhoffte Produktivitätssteigerung erreichen.

Die technische Architektur, die die verwendeten Architekturmuster erfüllt, sollte über mehrere Projekte stabil bleiben. Ansonsten besteht die Gefahr, dass

```
...
<dest:methode>
<!--id-->
<xsl:attribute name="id"><xsl:value-of select="@id"/>
</xsl:attribute>
 // set the connection to other side of association
 public void internalConnection(
   <xsl:value-of select="type/name"/>
   p<xsl:value-of select="uppername"/>) {
    if( (g<xsl:value-of select="uppername"/> == null) ||
        !g<xsl:value-of select="uppername"/>>.contains(
          p<xsl:value-of select="uppername"/>)) {
           g<xsl:value-of select="uppername"/>.add(
             p<xsl:value-of select="uppername"/>);
    }
 }
</dest:methode>
...
```

Listing 1: Quelle – Generator-Vorlage

der Aufwand, den man in die Erstellung der Generatorvorlagen gesteckt hat, sich nicht auszahlt.

Die letzte technische Voraussetzung – der Einsatz von Frameworks – mag vielleicht zunächst verwundern. Denn natürlich lassen sich auch Generatorvorlagen erstellen, die Quellcode erzeugen, der unmittelbar auf Standardbibliotheken und dem Sprachumfang der eingesetzten Programmiersprache basiert. Allerdings funktioniert dies nur begrenzt, denn man handelt sich einen gravierenden Nachteil ein: Ohne den Einsatz von Frameworks werden die Generatorvorlagen schnell sehr komplex. Das führt dazu, dass Anpassungen an den Vorlagen aufwändig und fehleranfällig werden. Weiterhin führt es dazu, dass selbstverständlich auch die Anpassungen und Ergänzungen des erzeugten Quellcodes schwieriger werden, da der Umfang dieses Quellcodes ohne den Einsatz von Frameworks erheblich größer wird.

Eine Vorgehensweise zur generativen Softwareentwicklung

Kommen wir nun zur konkreten Vorgehensweise, die zur Einführung und Optimierung generativer Softwareentwicklungsmethoden in diesem Artikel vorgestellt werden soll. Diese Vorgehensweise wird im Anschluss durch ein Beispiel konkretisiert. Wichtig ist jedoch die Feststellung, dass es sich um eine Vorgehensweise handelt, die unabhängig von Zielarchitekturen, Programmiersprachen, Modellformaten oder Generator-Tools angewendet werden kann. Auch die im Vorfeld beschriebenen Voraussetzungen sind nicht an diese konkreten Dinge gebunden, sondern auf einer allgemeineren Ebene definiert und zu verstehen.

Die im Folgenden beschriebene Vorgehensweise enthält einen kontinuierlichen Prozess zur Qualitätssicherung und -optimierung der initial erarbeiteten Generator-Vorlagen. Dieser kontinuierliche Prozess findet – wie später zu sehen ist – selbst iterativ und inkrementell statt.

Um aber überhaupt starten zu können, bedarf es besonderer Schritte, um die Einführung generativer Softwareentwicklungsmethoden durchzuführen. Hier sind die folgenden Schritte definiert:

1) Identifikation der Zielarchitektur

Der erste Schritt liegt in der Identifikation der Zielarchitektur. Mit diesem Begriff sind nicht nur die Architekturmuster gemeint, die bei der Generierung berücksichtigt werden sollen, sondern auch die Programmiersprache, die konkreten Frameworks, ggf. eine spezielle relationale Datenbank etc. In der Regel lässt sich dieser Schritt auf der Basis bereits vorhandener Projekterfahrungen durchführen. Damit ist gemeint, dass möglicherweise bereits ein oder mehrere Projekte auf Basis einer bestimmten Zielarchitektur durchgeführt wurden, jedoch nicht unter Einsatz eines Quellcode-Generators. So lassen sich dann auch nach diesem Schritt anhand bestehender Projekterfahrungen Entscheidungen über die generierbaren Komponenten treffen.

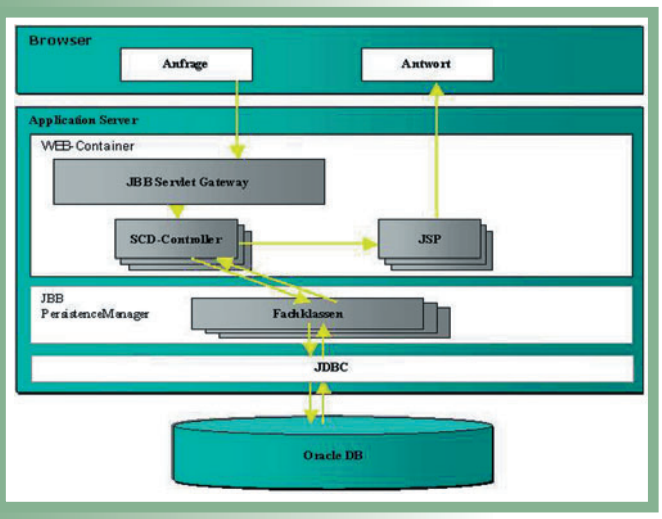

Abb. 1: Zielarchitektur

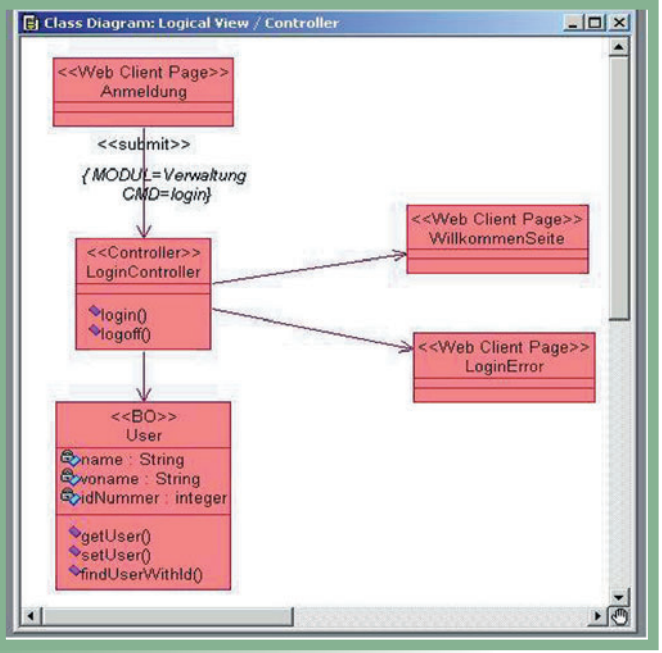

Abb. 2: UML-Diagramm

2) Identifikation der generierbaren Komponenten

Die Identifikation der generierbaren Komponenten ist sicherlich ein Schritt, der zum einen Abstraktionsvermögen und ein gutes Verständnis der Zielarchitektur erfordert, zum anderen durch Erfahrungswerte unterstützt wird. Hat man diese Erfahrungswerte noch nicht, so lassen sich einige Hinweise zur Identifikation geeigneter Komponenten finden. Geeignete Kandidaten sind beispielsweise

- Fachklassen, die Geschäftsobjekte abbilden und über einen Persistenzmechanismus in einem Datenspeicher gespeichert werden können.
- Schnittstellenklassen: Überall dort, wo definierte Interfaces realisiert werden müssen, lassen sich Generatoren einsetzen. Dies können Session-Beans sein, genauso aber auch Servlets, die ein definiertes Interface implementieren. Dies sind nur zwei Beispiele, die das große Potential gerade in diesem Bereich veranschaulichen sollen.
- Entwurfsmuster: Auch die Implementierung von definierten Entwurfsmustern ist ein Paradebeispiel für den Einsatz generativer Softwareentwicklung. Wie die Anbieter vieler Case-Tools wie TogetherJ beweisen, lassen sich Entwurfsmuster wie Queues, Pools, Singletons, EJBs etc. hervorragend über Generatoren aus Modellen erzeugen.
- Standardisierte GUI-Komponenten: Hier ist die Tabellendarstellung ein gutes Beispiel. Egal in welcher Form realisiert, stellen sich bei der Realisierung immer wieder die gleichen Aufgaben. Daher haben viele Entwickler bereits eine wiederverwendbare Tabellenansicht realisiert, die über Methoden zur Umsortierung, Aus- und Einblendung von Spalten etc. verfügt. Auch hier lassen sich Generatoren gut einsetzen, um die jeweils spezifische Ausprägung für eine Umsetzung zu generieren.

3) Auswahl geeigneter Modellformate

Dieser Schritt ist häufig der schwierigste, denn er ist eng gekoppelt an die Auswahl des Generator-Tools (nächster Schritt). Darüber hinaus liegt häufig nicht nur ein Modellformat vor, sondern je zu generierender Komponente möglicherweise unterschiedliche. Ein Beispiel dafür ist die Generierung einer 1:1 Abbildung persistenter Objekte auf eine relationale Datenbank anhand eines bestehenden Datenbank-Schemas. Hier ist zu empfehlen, diese unterschiedlichen Modellformate und -quellen möglichst in ein einheitliches Modellformat zu überführen. Dies erfordert zwar zunächst höheren Aufwand in einem konkreten Projekt, führt aber zu einer deutlich besseren Wiederverwendbarkeit der Generator-Vorlagen für weitere Projekte. Gute Erfahrungen existieren mit dem Einsatz einer erweiterten UML-Notation und der Verwendung des Generator-Tools UML-Bridge von Avantis (s. Beispiel im nächsten Abschnitt).

4) Auswahl des Generator-Tools

Bei der Auswahl des Generator-Tools ist die Abhängigkeit zum zuvor ausgewählten Modellformat zu berücksichtigen. Grundsätzlich lassen sich unterschiedliche Klassen von Generatoren unterscheiden

- *Generatoren in Case-Tools:* Diese Variante unterstützt in der Regel am besten das Round Trip Engineering, da der Generator in das Tool integriert ist, das zur Modellierung eingesetzt wird. Ein gutes Beispiel ist TogetherJ. Allerdings entsteht der Nachteil, dass die Anpassbarkeit häufig eingeschränkt ist, da der Fokus des Anbieters auf der Bereitstellung eines guten und wettbewerbsfähigen Case-Tools liegt und weniger auf der eines Generators.
- *Unabhängige Generator-Tools:* bieten den Vorteil, dass der Anbieter sich hier auf die Bereitstellung eines guten Generators konzentriert. Außerdem bieten diese Tools in der Regel eine deutlich breitere Palette an Möglichkeiten zum Import und zur Anpassung von Modellformaten. Darüber hinaus ist auch der Vorlagenmechanismus häufig allgemeingültiger und flexibler gewählt. Ein gutes Beispiel für ein solches unabhängiges Generator-Tool ist die UML-Bridge von Avantis, die es in der Foundation Edition kostenlos gibt [Ava].
- *Eigene Realisierungen:* Natürlich lässt sich ein Generator auch selbst entwickeln und auf die eigenen Bedürfnisse zuschneiden. Diese Variante hat sicherlich den Vorteil, dass unterschiedliche Modellformate und -quellen leichter zu integrieren sind und sich die Lösung ständig an die eigenen, sich wandelnden Anforderungen anpassen lässt. Nachteilig ist hingegen der Zusatzaufwand, der für die Erstellung des Generators benötigt wird. Dieser steigt insbesondere dann unverhältnismäßig an, wenn der Generator auch Anforderungen aus dem Round Trip Engineering erfüllen soll. Denn dann muss zumindest eine Berücksichtigung von durchgeführten Änderungen an generierten Quellen bei wiederholten Generierungsschritten erfolgen. Kann man auf diese Funktionalität verzichten, weil beispielsweise der Generator ausschließlich für das Rapid Prototyping eingesetzt werden soll, so erscheint eine eigene Realisierung vertretbar.

Sind diese vier initialen Schritte erfolgreich abgeschlossen worden, so beginnt die iterative und inkrementelle Entwicklung der Generator-Vorlagen. Darunter sind die folgenden Schritte zusammengefasst:

1) Erstellung einer Beispielrealisierung

Sofern für die Zielarchitektur aus vorangegangenen Realisierungen noch keine Beispielrealisierung vorliegt, sollte der erste Schritt immer in der Erstellung einer solchen Realisierung bestehen. Erst mit einer funktionierenden Beispielrealisierung hat man die Zielarchitektur verifiziert.

2) Anpassung/Ergänzung der Modellinformationen

Im zweiten Schritt gilt es festzustellen, welche Modellinformationen für die zu generierenden Quellen benötigt werden. Dazu sind die individuellen Anteile der Beispielrealisierung zu identifizieren. Ist dies erfolgt, so müssen die Modellinformationen um die noch fehlenden Teile ergänzt werden. Diese Vorgehensweise gilt für die Neuerstellung einer Vorlage genauso wie für die Erweiterung bestehender Vorlagen. Somit wachsen die Vorlagen bei jeder Iteration um ein Inkrement.

3) Erstellung/Erweiterung der Generator-Vorlagen

Als dritter Schritt steht die Erstellung bzw. Erweiterung der Generator-Vorlagen an. Dazu können Beispielrealisierungen genutzt werden, in denen die individuellen Anteile durch die dynamischen Zugriffe auf die Modellinformationen ersetzt werden. Listing 1 veranschaulicht dies.

4) Qualitätssicherung der Vorlagen

Die Qualitätssicherung der Vorlagen lässt sich besonders einfach durchführen, wenn es bereits bestehende Beispielrealisierungen gibt, die sich im Projekteinsatz bewährt haben. Dann ist zur Qualitätssicherung lediglich der Vergleich der Beispielrealisierung mit dem aus dem Generator erzeugten Quellcode erforderlich. Dies lässt sich mittels gängiger diff-Tools automatisieren. Treten Differenzen auf, so sind grundsätzlich drei potentielle Fehlerquellen zu betrachten: die Generator-Vorlage, der Generator selbst oder das Modell.

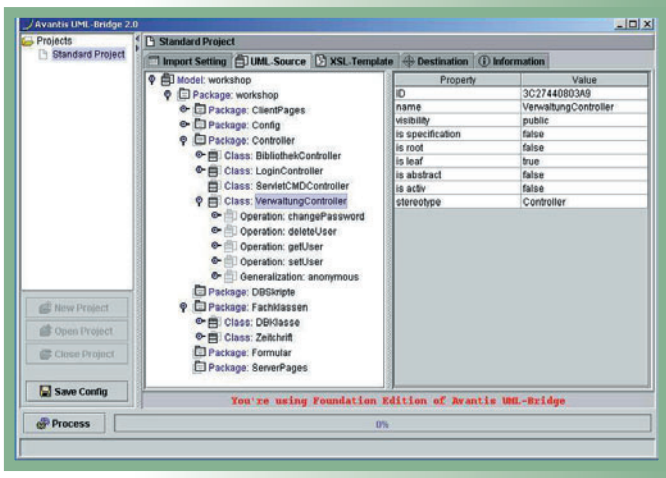

Abb. 3: UML-Bridge, Sourceansicht

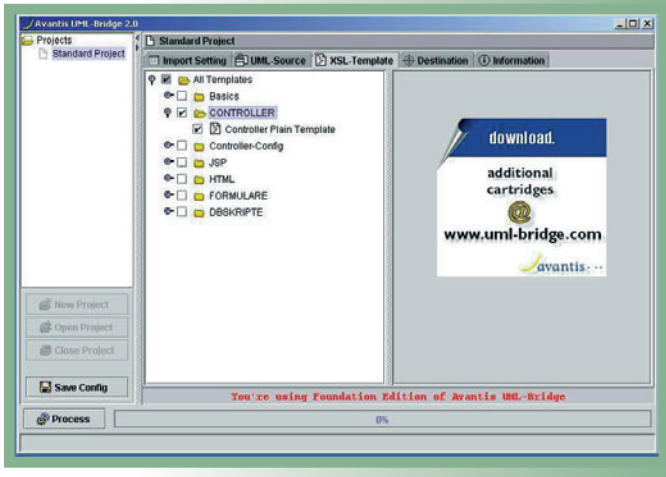

Abb. 4: UML-Bridge, Template-Ansicht

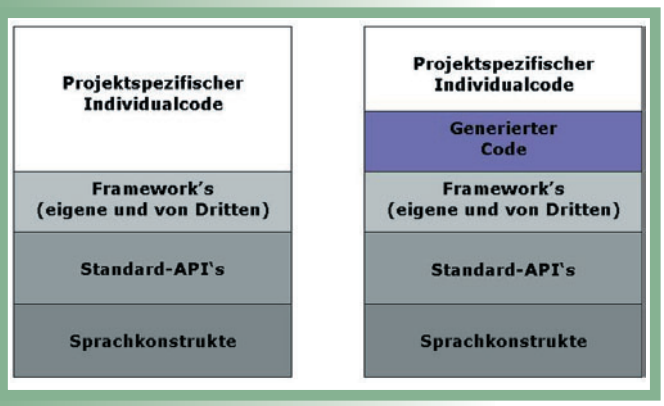

Abb. 5: Formalisierung der Codierung

5) Projekteinsatz

Der nun folgende Projekteinsatz soll vor allem zeigen, ob die gewählte Komponente tatsächlich ein geeigneter Kandidat für den Einsatz eines Generators ist. Hier sollen Erfahrungswerte entstehen, die die Grundlage für die folgende Schwachstellenanalyse darstellen.

6) Analyse der Schwachstellen

Bei der Schwachstellenanalyse stellen sich folgende Fragen, die bei einer Bewertung nützlich sind:

- **i)** Ist der Aufwand zur Pflege der Modellinformationen vertretbar oder wird dadurch das Modell bereits annähernd so komplex wie die Realisierung?
- **ii)** Werden die generierten Quellen im Projekt im Anschluss an die Generierung individualisiert?
- **iii)** Ist dies nicht der Fall, gibt es ggf. Performance-Gründe oder Architektur-Randbedingungen, die trotzdem für eine Generierung sprechen?
- **iv)** Lässt sich die Generierung der Komponente ggf. durch die Bereitstellung einer generischen Framework-Komponente ersetzen, die zur Laufzeit durch die Modellinformationen konfiguriert wird? Bei dieser Frage sind insbesondere Performance-Aspekte, Typsicherheit und die Komplexität der notwendigen Konfigurationen zu berücksichtigen.

Durch die Beantwortung dieser Fragen werden Optimierungspotentiale sowohl für die eingesetzten Frameworks wie auch für die Generator-Vorlagen erkennbar. Diese sollten dann in der nächsten Iteration berücksichtigt werden, die dann wieder mit der Anpassung/Erweiterung der Beispielrealisierung beginnt (Schritt 1).

Ein Beispiel aus der Praxis

Das folgende Beispiel soll die Vorgehensweise anhand konkreter Anforderungen aus der Praxis verdeutlichen. Die beschriebenen Voraussetzungen für die Einführung generativer Mechanismen waren gegeben. Es ging darum, Anteile einer bereits in mehrfachem Projekteinsatz bewährten Architektur für dreischichtige Webapplikationen zu generieren. Abbildung 1 veranschaulicht die bestehende Architektur in groben Zügen. Damit war bereits der erste Schritt – die Identifikation der Zielarchitektur – erfolgt. Im zweiten Schritt wurden die folgenden Komponenten für die Generierung ausgewählt:

1) Fachklassen auf Basis des JBB Persistence Managers,
2) DDL-Scripts zur Erzeugung der Tabellen,
3) Konfigurationsdateien im XML-Format für den Brockhaus JBB Configuration Manager,
4) Controller-Klassen für die Servlet-Schnittstelle über das JBB Servlet Gateway,
5) JSPs für die standardisierbaren Dialoge zum Einfügen, Ändern, Bearbeiten, Suchen, Anzeigen und Löschen von Objekten.

Die genannten Framework-Komponenten, auf deren Basis die generierten Quellen arbeiten, entstammen dem Java-Framework JBB (Java Building Blocks) der Brockhaus AG, das bereits in diversen Projekten zum Einsatz gekommen ist [JBB]. Als Modellformat wurde die UML-Notation ausgewählt. Diese wurde zu diesem Zweck um Stereotypen erweitert, um die Spezifika der Zielarchitektur besser abbilden zu können. Ein Ausschnitt eines solchen Diagramms ist in Abbildung 2 zu sehen.

Die Erweiterung der UML-Notation zur besseren Modellierung von Webapplikationen mit anschließender Generierung ist aktuell auch Gegenstand einer Diplomarbeit, die von der Brockhaus AG betreut wird und dieses Thema noch weiter vertiefen soll. Da die Modellierung in Rational Rose erfolgte, wurde ein Tool zur Generierung benötigt, das dieses Quellformat verarbeiten kann. Fündig wurden wir mit der UML-Bridge von Avantis, die in der Foundation Edition frei verfügbar ist und mit XSL-Vorlagen arbeitet (s. Abb. 3 und 4). Die initiale Erstellung der Generator-Vorlagen konnte dann innerhalb weniger Tage auf Basis der Quellen der bestehenden Projekte erfolgen.

Durch die Erfahrungen, die bereits im Vorfeld aus mehreren Projekten mit der Zielarchitektur vorhanden waren, ließen sich auch die Generator-Vorlagen in kurzer Zeit stabilisieren und verifizieren. In den darauf folgenden Iterationen wurden dann Benutzerrechtsprüfungen, Protokollausgaben und Mehrsprachigkeit in die Vorlagen integriert. Diese Funktionalitäten basierten wiederum auf den Java Building Blocks.

Vorteile

Sind die Einstiegshürden der Einführung generativer Softwareentwicklung genommen worden, so lässt sich als erster messbarer Erfolg die Reduzierung der Routinetätigkeiten bei der Codierung feststellen. Darüber hinaus werden aber weitere, qualitative Verbesserungen erreicht, die sich deutlich schwerer in harten Zahlen und Daten ausdrücken lassen. Insbesondere die Fokussierung auf eine modellgetriebene und architekturzentrierte Vorgehensweise wird durch die generative Softwareentwicklung gefördert. Ebenso lässt sich feststellen, dass die Wiederverwendung stärker in den Blickpunkt rückt. Durch die Auseinandersetzung mit den Generatorvorlagen werden Kandidaten für eine Wiederverwendung als generische konfigurierbare Komponente schneller entdeckt.

Auch das Erstellen von Prototypen ist mit Unterstützung von Generatoren deutlich effizienter und lässt sich vor allem unmittelbar auf der Zielarchitektur durchführen, sodass keine Wegwerf-Prototypen entstehen. Zudem wird ein weiterer Anteil der Codierung durch die Verwendung von Generatorvorlagen formalisiert und standardisiert, was sich in einer qualitativen Verbesserung des Codes be-

merkbar macht. Abbildung 5 veranschaulicht diesen Aspekt mit und ohne Einsatz von Generatoren. Als letztes sei noch genannt, dass auch der Wechsel von Mitarbeitern in Projekten besser unterstützt wird, da die Lesbarkeit und Einheitlichkeit des Quellcodes gesteigert wird. Zudem ist durch die modellgetriebene Entwicklung eine bessere Design-Dokumentation vorhanden.

Risiken

Allerdings sollen in diesem Beitrag auch die Risiken nicht verschwiegen werden, die die generative Softwareentwicklung mit sich bringen kann. Manche Entwickler neigen zur „Übertreibung" der Generierung und versuchen nun nahezu alles zu generieren. Dabei entstehen Vorlagen, die nur in einem Projekt genutzt werden können, weil sie zu individuell sind. Außerdem werden solche Vorlagen dann häufig auch nur für die Generierung weniger Dateien verwendet, sodass der Aufwand zur Erstellung der Vorlagen den Nutzen deutlich überschreitet.

Außerdem ist darauf zu achten, wie stark die generierten Dateien im Projektverlauf nachträglich individualisiert werden. Auch hier sollte der Anpassungsaufwand vertretbar bleiben, da sonst die Kosten/Nutzen-Relation nicht mehr stimmt.

Ein weiteres Risiko birgt die Weiterentwicklung der Bestandteile der Zielarchitektur. Hier müssen die Generatorvorlagen ständig mit diesen Bestandteilen abgeglichen und konsistent gehalten werden. Dazu eignet sich eine gemeinsame Release-Verwaltung von Framework-Komponenten und Generatorvorlagen. Nicht zuletzt sind bei einigen Entwicklern auch Akzeptanzprobleme zu erwarten, die mit der Umstellung auf eine modellgetriebene Entwicklung zu tun haben. Diesen kann man am besten begegnen, indem der Blick auf die persönlichen Vorteile für jeden einzelnen gerichtet wird. Diese liegen vor allem in der Zeiteinsparung und der Vermeidung langweiliger Routinetätigkeiten.

Zusammenfassung

Zusammenfassend lässt sich aber sagen, dass die Vorteile eindeutig die Risiken „wert sind". Gerade die höhere Stabilität der generierten Quellen führt in den Projekten zu einer deutlichen Effizienzsteigerung. Ich hoffe, dass ich mit dem Artikel einige Anregungen für die Einführung generativer Softwareentwicklung geben konnte und würde mich sehr über persönliches Feedback freuen.

Literaturverzeichnis und Links

[Ava] Avantis, UML Bridge Foundation Edition, da Avantis von der Plenum AG übernommen wurde und auf deren Seiten leider noch keine Download-Möglichkeit für die UML-Bridge vorhanden ist, müssen Interessierte direkt Kontakt mit der plenum AG aufnehmen (www.plenum.de)

[JBB] Brockhaus AG, Komponentenframework Java Building Blocks, http://www.brockhaus-ag.de/produktion/bag/WebSite/DE/ConsulTingServices/JBB/WasSindJBB/WasSindJBB.jsp

[Kru99] P. Kruchten, Der Rational Unified Process, Eine Einführung, Addison Wesley, 1999

[Oes00] B. Oesterreich, Erfolgreich mit Objektorientierung, Oldenbourg Verlag, 2000

Wolfgang Neuhaus (Dipl. Inform.) ist Chief Information Officer (CIO) bei der Brockhaus Software & Consulting AG. Seine Schwerpunktthemen sind objektorientierte Vorgehensmodelle und Framework-Architekturen.
E-Mail: wolfgang.neuhaus@brockhaus-ag.de.