Fallbeispiel: Generative Softwareentwicklung bei sd&m

Alexander Knecht, sd&m

software design & management AG
Thomas-Dehler-Straße 27,
81737 München
Email: knecht@sdm.de

Abstract: Generative Softwareentwicklung lebt von der Einführung eines formalen, fachlichen Modells in ein Entwicklungsprojekt. Anhand eines Fallbeispiels wird eine konkrete Vorgehensweise für typische Themen der Sprachdefinition, Datenerfassung und Generatorbau vorgestellt.

1 Einleitung

Das Spektrum an generativen Techniken und Methoden ist so vielfältig [CE00], dass sie für den Einsatz in Projekten gezielt zusammengestellt werden müssen. Im folgenden wird dazu ein konkreter Ansatz aus einem erfolgreichen sd&m-Projekt vorgestellt. Als Fallbeispiel dient eine Extranet-Anwendung, die über 800 HTML-Seiten umfasst und auf Basis von Java-Server-Pages und Servlets gebaut wurde. Davon waren über 100 unabhängige, formularbasierte Dialoge mit ähnlichen Funktionen aber stark unterschiedlicher Darstellung, sowie statische und redaktionell gepflegte Seiten. Alle Seiten folgen einem von einer Medienagentur festgelegten Styleguide. Die Anwendung ist mandantenfähig und unterstützt verschiedene Zielgruppen. Jede Zielgruppe hat ihre eigene Navigationsstruktur. Die fachliche Dialogspezifikation lag in Form von PowerPoint Folien vor. Die Projektlaufzeit betrug zehn Monate, wobei insgesamt etwa 15-20 Mitarbeiter beteiligt waren. Das System ist seit November 2001 produktiv.

2 Generative Entwicklung im Projekt

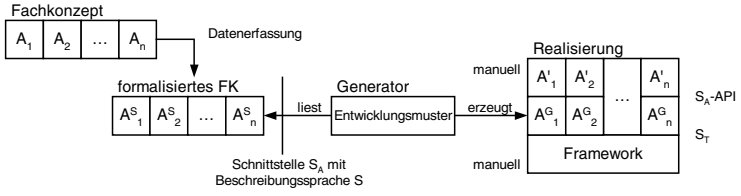

Abbildung: Generative Entwicklung im Projekt

Generative Entwicklung in einem Projekt, ist dann sinnvoll, wenn im fachlichen Konzept viele Anwenungselemente A_i vorkommen, die ähnlich realisiert werden sollen. Im Fallbeispiel werden diese Entwicklungsmuster in Generatoren isoliert, die sie dann als Generate A^G_i in die Anwendung verteilen. Dazu wird eine fachliche Schnittstelle S_A ins

Projekt eingezogen. Man definiert zum einen eine Beschreibungssprache *S*, in der die A^S_i formal erfasst werden, damit der Generator das Fachkonzept ‚lesen' kann, zum anderen eine Programmierschnittstelle S_A-*API*, über die eine Spezialisierung der A^G_i pro Element erfolgen kann. Zur Kopplung von manuellem und generiertem Kode wurde das ‚Sandwich-Pattern' verwendet. Zwei manuell erstellte Klassen in Framework und Anwendungskode klammern per Vererbung eine generierte Klasse. Virtuelle Methoden der technischen Schnittstelle S_T versorgen das Framework mit elementspezifischem Wissen und an S_A-*API* kann pro Element Ablauflogik individuell realisiert werden. Dies ermöglicht feingranulare Mustereinbindung ohne manuelles Nachbearbeiten bei wiederholter Generierung.

3 Sprachdefinition und Datenerfassung

Im Fallbeispiel werden drei Arten von Anwendungselementen auf Basis von XML formalisiert: Dialoge, fachliche Datentypen und die Menüs der Navigationsstruktur. Ein Dialog besteht aus ein oder mehreren Dialogseiten, die eine Menge von Dialogfelder enthalten. Die Dialogfelder unterteilen sich in reine Anzeigefelder für Texte und Bilder, Schaltfelder für Buttons, Datenfelder für Formulare sowie Gruppenfelder, die wiederum selbst Dialogfelder enthalten können. Zur Erfassung und Bearbeitung der XML-Dialogdefinitionen wurde ein spezieller Editor auf Basis des Vektorgrafikprogramms ‚Visio' realisiert. Jedes XML-Element wurde als Visio-Shape dargestellt; seine Attribute konnten über einen ‚Eigenschaften'-Dialog geändert werden. Es wurde ein pixelgenauer Maßstab zugrunde gelegt, in dem die Dialogfelder positioniert wurden. Über eine spezielle Symbolleiste wurden die Generatoren aufgerufen, die ihre Ausgabe direkt in die Testumgebung schrieben. Damit wurden sehr kurze ‚Edit-Compile-Test' Zyklen erreicht. Es war die Aufgabe der Anwendungsentwickler, die fachliche Spezifikation auf PowerPoint-Folien mittels des Dialogeditors zu erfassen.

Die fachlichen Datentypen wie Text, Aufzählung, Datum, Zeit und Geld sind im Framework hinterlegt. Spezielle Text- oder Aufzählungstypen, die sich nur durch bestimmte Parameter unterscheiden, wurden direkt als XML-Dokument gepflegt. Die Definition der vielen Aufzählungstypen wurde durch die Fachabteilung selbst durchgeführt, so dass der Erfassungsaufwand lediglich in der Nachbereitung der XML-Datei bestand.

Jede Zielgruppe hat seine eigene Menüstruktur. Die Menüpunkte legen Menütext, die Seitenüberschrift, einen Teil der URL sowie den aufzurufenden Dialog fest. Für die Navigation wurde auf Kundenwunsch ein Redaktionssystem auf Lotus-Notes Basis realisiert, mit dem die Menüs durch die Fachbereiche gepflegt werden.

4 Musterdefinition und Generatorbau

Zur Entwicklung der Generatoren wurden Generator-Definitionen in vier Schritten erarbeitet, aus denen die eigentlichen Generatoren wiederum erzeugt wurden. *(1) Prototyp-Erstellung*: Zuerst wird ein Prototyp der Generator-Ausgabe erstellt und in der Zielumgebung getestet. *(2) AT-Analyse*: Dabei werden im Ausgabe-Prototyp die anwendungsspezifischen Begriffe (A) identifiziert und somit von den rein technisch bedingten Textbereichen (T) getrennt. Die Notation A bzw. T ist an die „Software-Blutgruppen" 0, A, T, AT aus [Si99] angelehnt. *(3) Muster-Definition*: Hierbei werden die A- und T-Teile in

zwei Dateien ausgelagert und Redundanzen zusammengefasst. Die A-Stellen werden im T-Teil parametriert und in einem XML-Dokument zusammengefasst. Die Parameternamen aus dem T-Teil tauchen als Attributnamen im XML-Dokument wieder auf, um den Bezug zum T-Teil herzustellen. *(4) Generator-Definition*: Dazu werden die Muster-Definitionen über Pfadausdrücke mit den gewünschten XML-Knotenmengen verknüpft und die Reihenfolge der Musteranwendung bestimmt. Des weiteren wird ggf. benötigte Transformationslogik von XML-Inhalten in Generator-Ausgaben hinzugefügt und die Ausgabedateien festgelegt.

Im Fallbeispiel wurden alle Generatoren mit dem Generator-Generator *Egg* erzeugt [Kn01] und dadurch der Aufwand für den Generatorbau signifikant reduziert. Dieses Werkzeug erhält die Generator-Definition als Eingabe, die als ‚Quellkode-Stylesheet' wirkt. Im Prinzip funktionieren sie wie XSL-Stylesheets, jedoch können die Muster eins zu eins aus der Muster-Definition in Schritt (3) verwendet und die Transformationslogik in Java implementiert werden. Außerdem werden sie kompiliert und nicht interpretiert. Die Anbindung der XML-Elemente der Eingabe erfolgt wie in XSL über Pfadausdrücke. Von XSL wurde bewusst Abstand genommen, wegen unübersichtlicher Muster- und Transformationsdefinition, eingeschränkter Programmiermöglichkeiten, geringer Performanz und fehlender Whitespace Kontrolle in der Ausgabe.

5 Zusammenfassung

Zusammenfassend betrachtet hat sich der Generatoreinsatz im konkreten Projekt zum Fallbeispiel sehr bewährt. Die Vorteile der verbesserten Änderbarkeit, Qualität und verkürzten Entwicklungszeit ergaben sich aus der Isolierung der Entwicklungsmuster in Generatoren und der Einführung der formalen fachlichen Schnittstelle. Die Kommunikation wurde damit zuerst im Team und später auch mit dem Kunden präziser. Die Abhängigkeit zwischen der Anwendungs- und Frameworkentwicklung konnte durch die Einführung des Generatoren-Teams entkoppelt und gepuffert werden. So war es möglich, fachlich frühzeitig in die Breite zu gehen und gleichzeitig technische Entscheidungen spät zu treffen bzw. sie nachträglich zu ändern. Die Qualifikation der Entwickler konnte dabei im Projektverlauf noch wachsen, ohne ein Projektrisiko einzugehen. Letztlich wurde das Team früher als geplant wieder abgebaut, da vieles als Kodemuster realisiert werden konnte und somit weniger Kode manuell erstellt werden musste.

Literaturverzeichnis

[CE00] Czarnecki, K; Eisenecker, U. W.: Generative Programming - Methods, Tools, and Applications. Addison-Wesley, June 2000

[Kn01] Knecht, A.: Egg - Ein Generator Generator für Quellkode-Stylesheets. www.-generia.de/egg.pdf, Juli 2001.

[Si99] Siedersleben, J.; Albers, G.; Fuchs, P.; Weigend, J.: Schriftenreihe Informatik, Segregating the Layers of Business Information Systems. In (Donohoe, P Hrsg.) Software Architecture, 1999; S. 389-404.