Oberflächenmodernisierung mit MaTriX

Uwe Erdmenger
pro et con Innovative Informatikanwendungen GmbH, Dittesstraße 15, 09126 Chemnitz
uwe.erdmenger@proetcon.de

Abstract

Die Modernisierung von Benutzeroberflächen ist ein wichtiges Thema in Softwaremigrationsprojekten. Dabei beschleunigt der Einsatz von Tools das Projekt und gestaltet es kostengünstiger. Gegenstand der folgenden Ausführungen ist *MaTriX* [1], ein Werkzeug für die Entwicklung und Modernisierung von Bildschirmmasken unter Beibehaltung der verarbeitenden Serverprogramme sowie der Middleware und unter Verwendung moderner Webtechnologien wie Ajax und HTML 5.

1 Motivation

Ausgangspunkt der Betrachtungen ist ein Messagebasiertes Programmsystem, in welchem Bildschirmmasken mit Serverprogrammen durch Austausch von Messages (Telegrammen), die als COBOL-Strukturen definiert sind, kommunizieren. Für die Modernisierung solcher Benutzeroberflächen gibt es verschiedene Ansätze. Eine Möglichkeit ist die Maskenmigration unter Nutzung von Werkzeugen [1]. Bei einer Neuentwicklung kommen häufig Java-EE-Techniken (JSP, JSF, ...) zum Einsatz [2]. Dabei werden die Masken z.B. als JSPs oder Facelets neu erstellt, wobei auf umfangreiche Bibliotheken an Oberflächenelementen zurückgegriffen werden kann. Dieser Ansatz bringt jedoch auch eine Reihe von Nachteilen mit sich: Zusätzlich zur Middleware wird nun noch ein Application Server (z.B. JBoss) benötigt, welcher Administrationsaufwand und Serverlast verursacht. Diese wird noch erhöht durch die Notwendigkeit einer Schnittstelle, welche den Zugriff auf einzelne Felder der COBOL-Message von Java aus ermöglicht und dazu Kenntnisse des strukturellen Aufbaus jeder Message benötigt. Darüber hinaus müssen die Entwickler, die das Anwendungssystem warten und weiterentwickeln, die notwendigen Java-EE-Kenntnisse erwerben, wozu nicht immer Bereitschaft besteht.

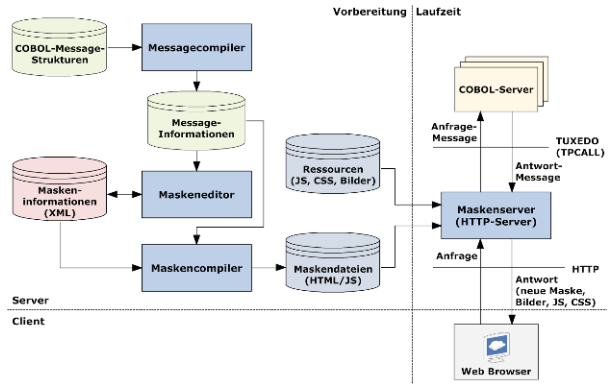
Ein Ziel der Entwicklung von MaTriX war es, möglichst *wenig Rechenlast auf dem Server* zu erzeugen. Die Auswertung der COBOL-Messages sollte erst auf dem Client erfolgen, die Realisierung aber dennoch *browserbasiert* sein und *keine zusätzliche clientseitige Installation* einer Software erfordern. Eine weitere Forderung war der *Einsatz moderner Web-Technologien* und die einfache Anpassbarkeit an neue technische Möglichkeiten, die sich z.B. durch HTML 5 bieten.

Die Entwicklung der neuen Oberfläche sollte einfach sein und keine Einarbeitung in eine neue Programmier- oder Auszeichnungssprache erfordern. Voraussetzung dafür war die Bereitstellung eines *grafischen Editors* (Maskeneditor), mit dem die Masken neu erstellt und bearbeitet werden können.

[1] MaTriX wird aus Mitteln der Europäischen Union und des Freistaates Sachsen gefördert.

2 Architektur

Die folgende Grafik verdeutlicht die prinzipielle Architektur von MaTriX:



MaTriX setzt sich aus vier zentralen Bestandteilen zusammen: Messagecompiler, Maskeneditor, Maskencompiler sowie einem Maskenserver. Diese Komponenten sowie ihr Zusammenwirken sollen im Weiteren erläutert werden.

Messagecompiler: Die Messages stellen die Schnittstelle zwischen den neu zu erstellenden Masken und den vorhandenen Serverprogrammen dar. Sie liegen als COBOL-Strukturen vor. Damit die anderen Komponenten auf die einzelnen Felder einer Message zugreifen können, ist eine Auswertung der COBOL-Struktur erforderlich. Es werden Informationen zu Namen, Verschachtelung und Typ der Felder ermittelt sowie deren Länge und der Offset berechnet und in einem definierten Format in Files abgelegt. Für die erforderliche syntaktische Analyse kommt ein von *pro et con* mit firmeneigenen Metawerkzeugen [3] entwickeltes COBOL-Frontend zum Einsatz.

Maskeneditor: Zum Erstellung und Bearbeiten von Masken wurde ein grafischer Editor entwickelt. Dadurch erübrigt sich das Erlernen einer speziellen Auszeichnungssprache (XML). Er greift auf die abgelegten Message-Informationen zu und gestattet damit eine dialogbasierte Bezugnahme auf Message-Felder. Darüber hinaus bietet er eine Vielzahl von Oberflächenelementen, eine flexible Möglichkeit für deren Anordnung in einem Grid-Layout, die Möglichkeit der Definition und Verwendung von Teilmasken und Maskentemplates usw. Er wurde als Web-Anwendung unter Verwendung von *jQuery* realisiert, so dass für ihn keine lokale Installation erforderlich ist.

Speicherung der Maskeninformationen: Der Editor legt die Maskeninformationen in XML-Files ab. Zu diesem Zweck wurde eine XML-Sprache definiert, welche leicht erweitert und an neue Anforderungen angepasst werden kann. Es existieren Elemente für Textanzeige, Eingabe (Input-Felder, Checkboxen, Selectboxen, Comboboxen, Kalender für Datumswerte uvm.), Aktionen (Links, But-

tons, Tastatureingaben), Layout, ... Ein besonderes Element für die häufig benötigten Tabellen wurde auf Basis der Funktionalität des jQuery-Plugins *DataTables* entwickelt. Es ist über einen Editordialog vielseitig konfigurierbar und erleichtert die Anzeige großer Datenmengen.

Maskencompiler: Da in dieser gespeicherten Form die Maskendaten nicht direkt vom Browser verarbeitet werden können, ist eine Übersetzung erforderlich. Sie erfolgt automatisch immer direkt beim Speichern des XML-Files auf dem Serversystem. Der Vorteil dieses Vorgehens ist eine Verlagerung der Kompilierung in die vorbereitende Phase, wodurch Server und Client zur Laufzeit entlastet werden. Für die Erstellung des Maskencompilers wurde die Programmiersprache *Perl* gewählt, da sie für die Verarbeitung von Text sehr geeignet ist und auch über performante Bibliotheken für den Umgang mit XML verfügt. Das Ergebnis der Übersetzung ist nicht direkt HTML, sondern ein JavaScript-File, welches bei Abarbeitung zur Laufzeit den notwendigen DOM-Tree aufbaut und anzeigt. Auf diese Weise können die Informationen aus der Message, die ja erst zur Laufzeit aktuell vorliegt, leichter eingebunden werden. Die in der XML-Definition `<input value="#{FD-MESSAGE.INOUT.BENUTZ}"/>` eines Eingabefeldes enthaltene Referenz auf das Messagefeld `BENUTZ` wird beispielsweise übersetzt in `inp_13.value = getMessageAlnum(182, 8);`, wobei 182 der berechnete Offset des Feldes in der Message und 8 seine Länge ist.

Maskenserver: Diese Komponente arbeitet als HTTP-Server und stellt zur Laufzeit die generierten Maskendaten, aber auch andere Ressourcen wie z.B. CSS-Files oder Bilder bereit. Gleichzeitig arbeitet sie als Client für die Middleware, indem sie die vom Browser empfangenen Messages an diese weitergibt, die Antwort abwartet und diese dann wieder zurück an den Browser schickt. Dabei wird der Message-Inhalt nicht verändert, so dass keine Kenntniss über ihren Aufbau erforderlich ist. Prinzipiell kann jeder Webserver entsprechend erweitert werden, wobei jedoch auf eine effiziente Anbindung der Middleware zu achten ist, um keine Performance-Lecks zu erzeugen.

Laufzeitbibliothek: Eine JavaScript-Bibliothek übernimmt die clientseitige Ablaufsteuerung. Dazu gehören z.B. das Senden und Empfangen der Messages, deren Auswertung, das ggf. notwendige Nachladen neuer Masken und Übertragen der Message-Daten in die Maske. Die gesamte Kommunikation wird über Ajax realisiert. Dadurch ist es möglich, Informationen auf dem Client (im Browser) zwischenzuspeichern und nur die sich ändernden Informationen neu zu übertragen. Somit ergibt sich die folgende Ablaufsteuerung: Nach dem Start von MaTriX wird eine initiale Message gelesen, ausgewertet und dann die dazu passende (Start-)Maske nachgeladen und inkl. der aktuellen Messagedaten angezeigt. Nach erfolgten Eingaben in der Maske wird mit diesen Daten eine neue Anfrage-Message erstellt und an den Server gesendet, welcher über die Middleware eine Antwort-Message berechnet und zurück sendet. Der gesamte Ablauf wiederholt sich zyklisch solange die MaTriX-Anwendung aktiv ist.

3 Erfahrungen und Ausblick

MaTriX wurde bereits für die Entwicklung von Masken in einem produktiven Umfeld eingesetzt. Serverseitig kommen bei dieser Anwendung COBOL-Programme unter der Steuerung der Middleware *Tuxedo* zum Einsatz. In einem Pilotprojekt wurden eine Reihe bereits vorhandener Masken mit dem neuen System reimplementiert, um den Anwendungsentwicklern und auch den Anwendern eine Möglichkeit des Vergleichs zu geben. Die ersten Erfahrungen bestätigen die Eignung des gewählten Ansatzes. Insbesondere die neuen Gestaltungsmöglichkeiten, wie z.B. scrollbare Tabellen und die damit verbundene Annäherung der Maskenanzeige an das von Web-Anwendungen her bekannte Design wurden positiv aufgenommen. Gleichzeitig konnten die späteren Anwender damit frühzeitig in das Projekt eingebunden werden und Einfluss auf dessen Ausgestaltung nehmen. Im Ergebnis dessen wurde z.B. eine History eingebaut, welche es gestattet, unter Aktualisierung des Inhaltes auf die 20 zuletzt besuchten Masken zurückzugehen. Dieses Verhalten war mit dem bisherigen System nur sehr umständlich mit Hilfe einer Zwischenspeicherung von Messages in der Datenbank realisierbar.

Die in o.g. Projekt verwandten Masken wurden mit dem Maskeneditor neu erstellt. Dieses Verfahren ist jedoch bei einer größeren Anzahl von zu migrierenden „Alt-Masken“ nicht effektiv. Daher sollen in einem nächsten Schritt die Möglichkeiten einer (teil-)automatischen Konvertierung untersucht werden. Dabei sind kundenspezifisch verschiedene Maskenformate auszuwerten und in die XML-Sprache zu migrieren. Für Masken, die in SCREEN COBOL vorliegen, kann zu diesem Zweck mit *ScreenConv* ein bereits existentes anderes Werkzeug von *pro et con* zum Einsatz kommen. Eine besondere Herausforderung ist die Ablösung der bisherigen Positionierung mit festen X/Y-Koordinaten durch das Grid-Layout. Dabei wird ein hoher Automatisierungsgrad angestrebt, um einen Übergang zu MaTriX so reibungslos wie möglich zu gestalten.

Literaturverzeichnis

[1] Erdmenger, U.; Kaiser,U.; Loos, A.; Uhlig, D.:
Methoden u. Werkzeuge für die Software Migration.
In: Gimnich, R.; Kaiser, U.; Quante, J.; Winter, A. (Eds.): 10th Workshop Software-Reengineering (WSR 2008), 5.-7. May 2008, Bad Honnef.
Lecture Notes in Informatics, (LNI)-Proceedings, Volume P-126, S. 83-97

[2] Bodhuin, T. et al.: Migrating COBOL systems to the Web by using the MVC design pattern.
In: Proceedings Ninth Working Conference on Reverse Engineering WCRE 2002,
ISSN 1095-1350, S. 329–338

[3] Erdmenger, U.: Der Parsergenerator BTRACC2.
11. Workshop Software-Reengineering (WSR 2009), Bad Honnef 4.-6. Mai 2009
In: GI-Softwaretechnik-Trends, Band 29, Heft 2, ISSN 0720-8928, S. 34 und 35