Zwischen Skylla und Charybdis: Automatische vs. manuelle Content-Migration eines grossen Informationsportals

Dr. Andreas Zamperoni, Jörg Hohwiller, sd&m AG

1. Ausgangssituation

Anfang 2004 gewann sd&m den Auftrag, das zentrale Informationsportal des Bundes, www.bund.de, von einer proprietären Anbieterlösung auf ein Standard-Contentmanagement-System (CMS) zu migrieren. Während dies eine komplette Neuentwicklung der Portalsoftware sowie der Software zur individuellen Anpassung CMS-Komponenten bedeutete, war eine der Schlüsselanforderungen, alle aktuellen Seiten und Dokumente inhaltlich und bzgl. der Darstellung strukturell und layout-technisch identisch in das neue System zu überführen.

www.bund.de bietet umfassende, kommentierte Wegweiser zu allen Dienstleistungsangeboten der Bundesbehörden, sowie viele Informationen und Verweise auf dieselben auf Landes- und Kommunalebene. Darüber hinaus verfügt das Portal über eigene und importierte Informationsangebote, wie z.B. Stellenangebote der Bundesagentur für Arbeit im Öffentlichen Sektor oder geografische Such- und Anzeigemöglichkeiten für Behördenadressen. Insgesamt umfasst das Portal ca. 20.000 – 30.000 Seiten, die über verschiedene Navigations- und Suchalternativen angesteuert werden können. Die genaue Anzahl der Seiten lässt sich jedoch nicht genau bestimmen, da eine signifikante Anzahl Seiten das Ergebnis dynamischer oder statischer Suchanfragen ist.

Aufgrund der vertraglichen Rahmenbedingungen standen für die Content-Migration ausschließlich zwei Datenquellen zur Verfügung: 1. das Live-Portal, wie es im Internet sichtbar war und 2. ein Dump der zugrundeliegenden Microsoft SQL-Datenbank des Altanbieters.

Zielformat der Migration war ein XML-Datenmodell, das durch den Importer des neuen CMS verarbeitet werden konnte.

2. Manuelle vs. Automatische Migration

Weder für den Kunden noch für sd&m war es zum Ausschreibungszeitpunkt abzuschätzen, welchen objektiven Aufwand es kosten würde, den Gesamtcontent des Portals in das neue System zu migrieren. Als Worst-Case-Szenario hatte der Kunde die manuelle Migration (Neueingabe durch Schreibkräfte) eines Großteils der Inhalte einkalkuliert, was ihm jedoch jegliche Ressourcen für redaktionelle Betreuung und Weiterentwicklung der Angebote genommen hätte. Darüber hinaus erfordert eine manuelle Migration des Contents die Bereitstellung eines geeigneten Redaktions- und Preview-Systems, sie hätte also erst zu einem relativ späten Projektzeitpunkt stattfinden können, bzw. hätte den Launchtermin erheblich verzögert.

Voraussetzung für einen hohen Automatisierungsgrad bei der Content-Migration ist ein qualititativ hochwertiges DB-Schema der Datenquelle, das vom Organisationsparadigma her dem des Zielschemas gleicht.

Leider war die Ausgangssituation in diesem Fall völlig gegenteilig, was auf den Charakter des Altsystems – eine proprietäre, „gewachsene“ und aus anderen Projekten wiederverwendete Individuallösung – zurückzuführen war. So hatte die Ausgangs-DB folgende, eine Migration erschwerende Charakterisitka:

- Keine Primärschlüssel, keine referentielle Integrität
- Im Gegensatz zum CMS-Standard keine Zuordnung von DB-Tabellen zu Dokumenttypen, sondern von Tabellen zu Propertytypen (Properties = Inhalts- und Meta-Attribute der Dokumente)
- „De-normalisierte“ Tabellen, d.h., Tabellen mit Dokumenten verschiedener Dokumenttypen
- Datensätze einer Tabelle mit Schlüsseln unterschiedlicher Syntax und Semantik, teilweise in der gleichen Spalte
- Keine uns als Nachfolgeanbieter zur Verfügung stehende zusätzliche Dokumentation, Informationen oder gar Datenmodelle

Trotz dieser widrigen Ausgangssituation erschien den Projektbeteiligten aufgrund der großen Datenmenge der Versuch einer so umfassend wie möglichen automatischen Content-Migration sinnvoll und erstrebenswert. Die gegenübergestellten Budgetschätzungen für eine manuelle Migration (mehrere hundert Bearbeitertage (BT)) und automatischer Migration (ca. 60 BT + 40 BT Nacharbeiten) untermauerten dies eindeutig. Dafür waren aber zunächst ein komplettes Reverse-Engineering der Alt-DB und eine Entscheidung über zu verwendende Werkzeugunterstützung notwendig.

3. Vorgehen

Der erste Ansatz, den Content mit einer Data-Warehouse-Software aus der Alt-DB zu extrahieren, konnte aus (Lizenz-)Kostengründen nicht ernsthaft betrachtet werden. Ebenso wurde nach Prüfung verworfen dann verworfen, für die Migration einen OR-Mapper (Hibernate, Castor, etc.) einzusetzen. Die Abbildungsmöglichkeiten der untersuchten OR-Mapper

erwiesen sich aufgrund der (mangelnden) Qualität des Schemas der Alt-DB (s.o.) als nicht mächtig genug.
Danach wurde der Versuch unternommen, die komplette Conetnt-DB „1-zu-1“ als XML zu exportieren (bei MS-SQL mittels XML Unterstützung möglich) und die Weiterverarbeitung mit XSLT durchzuführen. Neben der schlechten Performance (Speicher und Rechenzeit) von XSLT scheiterte dies aufgrund der Beschränktheit von XSLT: die Komplexität der benötigten Transformation war einfach zu hoch für XSLT.
Letztlich wurde ein Brute Force Ansatz mithilfe von Java gewählt. Mit direkter JDBC-Programmierung wurde eine minimale Zugriffsschicht geschaffen um die Dokumente als Ganzes aus der Alt-DB zu extrahieren. Eine darauf aufsetzende Software enthielt die hartcodierten Transformationen für jeden einzelnen Dokumenttyp. Aufgrund des Wegwerf-Charakters der Software war dieser Ansatz vertretbar, denn eine spätere oder häufige Wartung dieser Software wäre nahezu unmöglich gewesen.
Die Software legte die Dokumente direkt als BLOBs in Dateien (ca. 2 GB) ab und verwendete die IDs als Dateinamen. Die einzelnen Dokumente wurden dann pro Dokumenttyp komplett in den Hauptspeicher geladen und in das Schema des Zielsystems überführt. Das Gesamtergebnis wurde als eine XML-Datei (37 MB, ohne Bilder, Downloads, etc.) für den Content sowie eine weiter XML-Datei für die Benutzer- und Gruppendaten auf die Festplatte geschrieben.

4. Besondere Herausforderungen

Insgesamt gab es bei der Migration viele kleine und große Probleme zu lösen. Hier ein kurzer Auszug:

- Textfragmente werden im Ziel-CMS als XML-Richtexte nach einer festen DTD gespeichert. Im Alt-System lagen sie als beliebige Strings vor, waren weder wohlgeformt noch HTML-konform. Ein großer Teil der Textfragmente waren im Altsystem als MS Office abgelegt und enthielten MS Office-spezifisches XML. Oft fehlten sogar schließende spitze Klammern bei Tags. Mit JTidy und vielen manuell erstellten – und Schritt für Schritt weiterentwickelten – Ersetzungsregeln konnten diese Content-Fragmente zum großen Teil bereinigt und dann automatisch weiterverarbeitet werden.
- Verweise von Inhalten untereinander waren auf viele unterschiedliche Weisen abgebildet! Z.B. hatte jedes Dokument zwei IDs – und es galt, die impliziten (= nicht dokumentierten) Regeln zu erkennen, welche der beiden IDs im Einzelfall zu verwenden war.
- In Textfragmenten waren (1) Links mit Javascript, (2) interne Verweise als externe URLs (z.B. www.bund.de/Service/English-.6118-2.htm), (3) interne Verweise nach 12 verschiedenen Notationsregeln, (4) Textinformationen wie „kein externer Zugriff möglich“, (5) Links auf das Lokale Dateisystem („file:|C:\\Dokumente und Einstellungen\...), und weitere Verweisarten vorhanden.
- Die Ziele der Navigationsknoten wurden zum größten Teil als statische URL-Strings in der DB gespeichert. Dabei waren diverse interne Parameter mit in die URL codiert, die den Aufbau und die Wiedergabe von Teilen der Seite dynamisch nach internen (= nicht dokumentierten) Regeln steuerten. In diesem Fall musste neben der Content-DB auch die Wiedergabe der Seite im Internet zu Rate gezogen werden.
- Die Inhalte im Alt-System wurden nicht strukturiert abgelegt und redaktionell benannt, sondern im Redaktionssystem ausschließlich über Suchtrefferlisten erreicht. Im Zielsystem wird jeder Inhalt im Repository mit einem redaktionellen Namen versehen und in einer Ordnerstruktur (wie in einem Dateisystem) abgelegt. Diese Ordnerstruktur musste synthetisch, aufgrund der inhaltlichen Merkmale der Alt-Dokumente gebildet werden - eine Gratwanderung zwischen Längenbegrenzungen, Lesbarkeit und Eindeutigkeit.
- Die eigene Software zum Extrahieren und Transformieren der Daten aus der Alt-DB bis in die Zwischenablage im Dateisystem dauerte weniger als 30 Minuten. Der reine Import der Daten in ein leeres CMS-Repository mit dem Importer des vorgegebenen CMS-Produktes dauerte danach zwei volle Tage! Viele Testdurchläufe waren dadurch nicht möglich. Aufgrund der starken Verlinkung der Inhalte untereinander war es nahezu unmöglich, die Daten in „Inseln“ zu zerlegen, um sie dann getrennt importiert zu können. Bei der automatischen Publikation der Inhalte durch den Importer kam es trotz Einstellung des maximal zulässigen Heaps (1,8 GB) zu OutOfMemory-Fehlern. Ein Bug im CMS-Produkt führte bei der anschließenden Nachpublikation zu Inkonsistenzen, die aufwendig manuell oder durch weitere Migrationsskripte repariert werden mußten.

5. Fazit

Wir haben in unseren CMS-Projekten festgestellt (und übertragen dies auch auf andere Projekte mit Datenmigrationsanteil): die Datenmigration ist in der Regel eine einmalige Sache; Eleganz ist hier fehl am Platz – vor allem, wenn es um „echte“ Legacydaten geht. Mit „echten Legacydaten“ bezeichnen wir Daten, deren Bedeutung und Vernetzung – Semantik – zwar erhalten bleiben soll, die sich in der Beschreibung, Organisation, Struktur und Verlinkung (Syntax) stark verändern. Nach unserer Erfahrung ist die Effizienz eines erfahrenen Java-Programmierers an dieser Stelle nicht zu schlagen.
Vorher abschätzen und zu vergleichen sind die Aufwände für manuelle vs. automatische (+ Nachbesserungen!) Migration: zumeist existiert eine klare und einfach zu berechnende Trennlinie für die Anzahl der Dokumente, ab der sich die automatische Migration lohnt – und diese ist meist höher als man denkt. Wir haben die Grenze im Verlauf unserer Arbeiten um den

Faktor 10, von 3-5 auf 30-50 Dokumente pro Dokumenttyp erhöht.
Im Idealfall ist die Datenmigration nach Abschluss der fachlichen (Neu-)Entwicklung der Contentstruktur für das Zielsystem angesiedelt. Dann steht das Zielsystem in vollen Umfang zur Verfügung. In vielen Projekten ist aus Zeitgründen eine Parallelisierung von fachlicher Entwicklung und Migration der Daten notwendig. Dies erfordert aber eine enge Abstimmung der fachlichen Teams mit dem Migrationsteam und eine aufwendigere (teurere) kontinuierliche Anpassung der Migrationsskripte. Obwohl daher eigentlich zu vermeiden, bietet diese Parallelisierung auch einen großen Vorteil, der bei der Planung des Migrationszeitpunktes eine ausschlaggebende Rolle spielen kann: bei möglichst früher Migration kann zum Integrationstest schon die echte Datenbasis für die Testfälle genutzt werden. Dies verbessert die Qualität der Integrationstests erheblich und, wie in unserem Falle, erlaubt es sogar, den Integrationstest und den Abnahmetest (fast) gänzlich zu vereinen.
In jedem Fall ist es sehr wichtig, sehr gut mit dem Kunden zu kommunizieren, welches Ergebnis aus dem automatischen Teil der Migration zu erwarten ist. Im Gegensatz zu bestimmten fachlichen Teilen eines Software-Systems vermag der Kunde für die Datenmigration als abstrakte, software-technische Transformation oftmals nicht abzuschätzen, wie hoch ihre Komplexität letztlich ist. Der Anteil der automatischen Migration steuert jedoch zwei Aktivitäten, die in der Regel durch den Kunden beigestellt werden müssen: (1) der Anteil der manuellen Migration und (2) die Doppelpflege (in Wechselwirkung mit der Delta-Migrationsstrategie), d.h., welche Dokumente er nur im alten, nur im neuen System oder doppelt pflegen muss.

Dr. Andreas Zamperoni ist promovierter Informatiker (Software Engineering) und seit 2002 bei sd&m Frankfurt als kundenverantwortlicher Projektmanager und Projektleiter, vor allem im Bereich Content-Management-Systeme und Portale (www.t-online.de, www.telekom.de, www.bund.de), tätig.

Jörg Hohwiller ist Diplom-Informatiker und seit 2002 bei sd&m Frankfurt, vor allem im Bereich Content-Management-Systeme und Portale (www.t-online.de, www.bund.de), tätig.