Migration auf Knopfdruck – Macht das überhaupt Sinn?

Markus Klenke
TEAM GmbH
Paderborn

Schlüsselworte

Migration, Modellgetriebene Softwaremigration, ADF, Forms

Einleitung

Nach der Bereitstellung eines Forms-Life-Time Supports von Oracle stellt sich nun nicht mehr die Frage, wie lange die bestehende Applikation gewartet werden kann, sondern wieweit neue Anforderungen mit der nicht mehr weiterentwickelten Plattform realisierbar sind. Um die Langlebigkeit der Applikation zu gewährleisten ist es daher immer noch förderlich, die Applikation in eine neue Umgebung zu migrieren. Dieses Unterfangen bringt allerdings einen großen Kostenfaktor mit sich. Um diesen zu minimieren, bestreben viele Unternehmen eine automatische 1:1 Migration in die neue Architektur. Es besteht jedoch die Gefahr, dass bei komplexen Applikationen eine Zielapplikation generiert wird, welche zwar die Anforderungen der Altapplikation im Sinne der Funktionalitäten abbilden könnte, jedoch keine wartbare Applikation für die Nachwelt darstellt, da sie nicht den Konventionen und Ideologien der neuen Technologie entspricht. Wir wollen im Folgenden einige Ideen aufzeigen, wie eine teilautomatisierte Migration mit manuellen Zwischenschritten und einem Abstraktionsansatz die oben genannten Probleme der Migration auf ein Minimum reduzieren kann. Als exemplarisches Beispiel wird Oracle Forms als Quellsystem sowie das Oracle Application Development Framework als Zielsystem betrachtet, da sich diese Systeme in der Sprache sowie der Denkweise der Implementierungen sehr stark voneinander unterscheiden.

Tausend gute Gründe zu migrieren

Die Entscheidung für eine Migration der Quellsoftware kann aus unterschiedlichen, sich nicht ausschließenden Gründen gefällt werden. In fast allen Szenarien werden Wiederverwendbarkeit, Wartbarkeit und User Experience als Hauptintention für die Überführung der Applikation in eine neue Architektur gesehen. Zusätzlich ist, wie in der Einleitung beschrieben, auch die Weiterentwicklungsmöglichkeit ein starker Migrationstreiber, um weiterhin Akzeptanz beim Endbenutzer zu erfahren und durch moderne Features neue Endbenutzer hinzuzugewinnen.. Begutachtet man diese Anforderungen im Detail, kann man feststellen, dass die Anforderungen an verschiedene Bereiche der Migration gestellt werden. So ist die Anforderung der Wartbarkeit und Weiterentwicklung einer Applikation an die Zielarchitektur gerichtet, wohingegen die Anforderung der Wiederverwendbarkeit auch starke Verknüpfungspunkte zur Altapplikation aufweist. Auch die User Experience ist stark von der Flexibilität der Anwender der Altapplikation abhängig. Soll die Applikation sich genauso anfühlen wie die Altapplikation, so ist ebenfalls eine starke Abhängigkeit zum Altsystem zu erkennen.

Neben diesen Gründen kann aber auch die Ausrichtung in die neue Architektur ein Auslöser für die Migration sein. So kann durch den Übergang zu einer moderneren Architektur Funktionalität, welche im Altsystem durch Work-Arounds gelöst wurde, sauber neu aufgesetzt werden und Konzepte, die in dem alten System nicht realisierbar waren, zur Applikation hinzugefügt werden. Durch die nun

hinzugewonnene neue Sicht auf die Applikation wird es zusätzlich einfacher neues Personal für die Weiterentwicklung der Applikation zu finden, welches wiederum eine Investition für die Zukunftssicherheit der Anwendung darstellt.

„We break for nobody“

Eine mögliche Migrationsstrategie ist es, ein 1:1 Mapping der Komponenten und Funktionalitäten bereitzustellen, es muss also für jedes Feature und jedes Element aus dem Quellsystem ein Ziel in der neuen Architektur bereitgestellt werden. Vor allem für granulare Komponenten wie simple Oberflächenelemente oder Elemente zur Datenakquise stellt sich ein Mapping relativ einfach dar, falls beide Systeme für ähnliche Konzepte gedacht sind. In unserem Beispiel kann ein FormModul und der Business Components Layer aus ADF in Betracht gezogen werden. Viele Elemente verhalten sich funktional sowie visuell sehr ähnlich und können daher auch von Plattform-Neulingen sehr schnell verwendet werden.

Auch source code kann für sich gesehen durch einfache Abbildungen vom Quell- zum Zielsystem transformiert werden. Dies wird natürlich ebenfalls bei bestehender Architekturnähe der beiden Systeme vereinfacht. So können Attribute auf Attribute sowie Funktionen auf Funktionen abgebildet werden (ähnlich einem Compiler). Um die Funktionalität des Codes weiter aufrecht zu erhalten muss allerdings gewährleistet werden, dass alle plattformspezifischen Funktionalitäten des Quellsystems in der modernisierten Applikation nachgebaut werden müssen. Bei einer endlichen Anzahl an Funktionen ist dies automatisiert möglich, jedoch bedarf es eines sehr hohen manuellen Aufwandes die Funktionalitäten im Ganzen auszubauen.

Ein Ei gleicht nicht dem anderen

Die beschriebenen Punkte deuten also allesamt darauf hin, dass eine automatisierte Migration von Forms nach ADF relativ einfach realisierbar ist. Warum tun sich also so viele Projekte schwer mit einer Migration? Der einfache Grund ist: Forms und ADF sind grundverschieden, auch wenn sie von außen betrachtet die gleiche Grundfunktionalität abdecken, nämlich effektive und einfache Interaktion mit einem Datenbestand.

Der Unterschied macht sich schon von der architekturellen Sicht bemerkbar. Während Forms monolithisch geprägt ist, also eine sehr enge Verzahnung zwischen Oberfläche und Datenbank anstrebt, ist die Realisierung in ADF ein gegenläufiger: Die Oberfläche einer Applikation soll nur zur Anzeige und Interaktionsmöglichkeit des Benutzers dienen, ist also in der Theorie strikt von jeglicher Logik zu trennen. Auch die Business-Service Schicht der beiden Plattformen unterscheidet sich vehement.

Noch schwerwiegender fällt allerdings folgender Punkt ins Gewicht: Die Denkweise, in der auf beiden Systemen entwickelt wird ist eine unterschiedliche. Während Forms-Applikationen sehr dialoglastig und kompakt entwickelt werden, ist ADF sehr prozessorientiert und jede Applikation kann als Komponente in einem höher liegenden Prozess gesehen werden.

Gerade diese Unterschiede machen sich bei einer Migration von Forms zu ADF sofort bemerkbar, daher ist ein Trade-Off zwischen Wünschen an die Zielplattform und Wiederverwendbarkeit der Quellapplikation unausweichlich; Migrationen, die sich zu sehr auf eine Migrationsintention stützen,

werden im am Abschluss einer Migration keine ADF Applikation erhalten, welche den anderen der Intentionen genügt.

Wird der Fokus zu sehr auf die Konservierung der Altapplikation gelegt, erhält man eine Applikation, die im Kern noch eine Forms Applikation ist, aber weder die Vorteile der effizienten Datenanbindung von Forms genießt, noch die Vorteile einer prozessorientierten Darstellung und Benutzerführung von ADF nutzt. Verfällt man zu sehr in die Modernisierungsintention erreicht man zwar eine verständliche und effiziente ADF Applikation, kann jedoch kaum Elemente aus der Altapplikation übernehmen. Versucht man alle Quellartefakte zu überführen und ein effektives Mapping zu ADF aufzustellen, so ist am Ende des Tages kaum mehr eine Automatisierung zu erwarten, weil für jeglichen Schritt und jede Komponenten eine eigene Transformationsregel geschrieben werden müsste.

Es ist daher wichtig eine Gradwanderung zwischen Automatisierung, Konservierung und Modernisierung zu gehen (siehe Abb. 1: Warndreieck der Migration), um das bestmögliche Ergebnis einer Migration zu erhalten. Und genau dies wird durch eine semiautomatisierte, modellbasierte Migration ermöglicht.

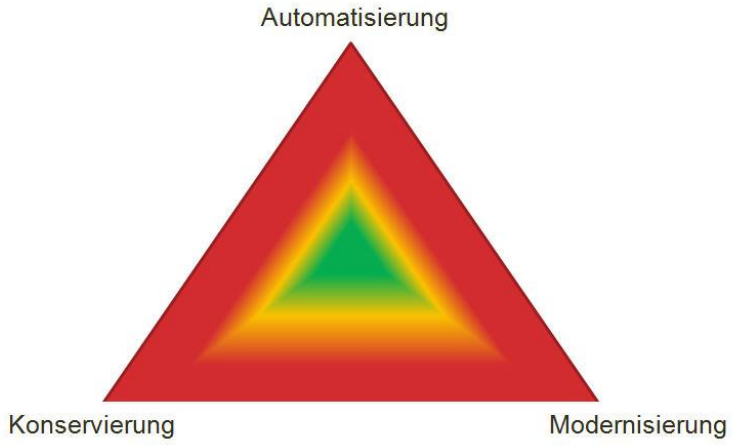

Abb. 1: Warndreieck der Migration

Die Migration unter der Motorhaube

Um die beschriebene Gradwanderung zu unterstützen bietet es sich an, die Migration von vorn herein gut durchzuplanen und viele verschiedene Szenarien aufzusetzen, die dann in einer Entscheidungsrunde bewertet werden können. Zum einen gehört dazu natürlich eine vollständige Bestandsaufnahme der bestehenden Applikation bereitzustellen. Diese Informationen müssen zu jedem Zeitpunkt der Migration zugreifbar gemacht werden. Zum anderen bietet es sich in diesem Schritt an, die Quellapplikation so gut es geht zu analysieren, ungenutzte Bereiche ausfindig zu machen und diese zu eliminieren, Cluster von fachlich zusammenhängenden Sourcen zu bilden so wie hauptsächlich benutzte Arbeitsabläufe zu extrahieren. Dieser Schritt ist bereits eine effektive Unterstützung um die Applikation vor Beginn der eigentlichen Migration zu entschlacken und die Gefahr auf fehlerhaft generierten Elementen auf der Zielseite zu minimieren. Dieser Schritt kann in

vielen Bereichen automatisiert unterstützt werden, bedarf allerdings zu jeder Zeit eine manuelle Einsicht, um die Korrektheit der gefundenen Komponenten zu gewährleisten und somit das Fundament einer erfolgreichen Migration.

Viele der Quellapplikationen sind schon lange in Produktion, so dass sich über die Zeit sehr viele individuelle Anpassungen im Code und am Framework finden lassen. Eine automatisierte 1:1 Migration würde diese Komponenten außer Acht lassen. Somit ist es wichtig, dass sich die Migration nicht mit Code allein, sondern viel mehr mit Konzepten und Features aus der Quellapplikation befassen sollte. Es ist nicht der migrierte Code, der im späteren Verlauf den gewünschten Effekt bringt, es ist vielmehr das Verständnis und die Abbildung von Mustern der Applikation. Um diesen Schritt zu gehen und um der Zielarchitektur eine Chance zu geben, auf eine andere Weise diese Features zu implementieren, ist es notwendig und hilfreich sich von der ursprünglichen Plattform zu entfernen und die wichtigsten Bereiche der Applikation abstrakt zu beschreiben. Durch diese Abstraktion besteht wiederum eine Möglichkeit, sich einen sehr guten Einblick geben zu lassen, wie sich die Applikation in ihrem Klarzustand verhalten sollte. Änderungen und Justierungen lassen sich auf dieser Ebene sehr handlich bearbeiten, ohne Gefahr zu laufen, dass die Quellapplikation nach den Änderungen nicht mehr lauffähig ist.

Durch die Bereitstellung einer Referenzarchitektur kann nun geprüft werden, welche Wünsche an die Zielapplikation in Abhängigkeit von Altsystem, Ressourcen und Plattform möglich sind. Auch hier ist ein manueller Anteil Pflicht, um eine genaue Abschätzung des weiteren Migrationsvorgehens bereitstellen zu können oder einfach um neue Features in die Applikation mit einzubinden.

Zum Vergleich: Ein Beispiel.

Um den Vorteil einer modellgetriebenen, semi-automatischen Migration noch einmal etwas näher zu beleuchten, sollen nun an einem abstrakten Beispiel die unterschiedlichen Schritte einer Migration beschrieben und erläutert werden.

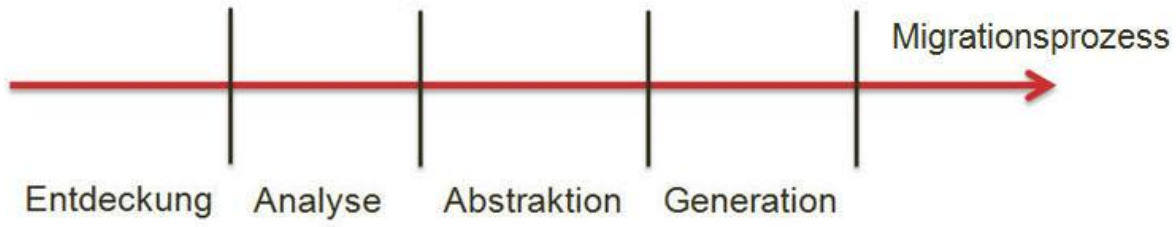

Abb. 2:Globale Schritte einer Migration

Im Entdeckungsschritt unterscheiden sich beide Verfahren kaum voneinander. In beiden Varianten wird vollautomatisch eine Menge von Quellsourcen geladen und analysiert. Zusätzlich können im modellgetriebenen Ansatz direkt Verknüpfungen zwischen einzelnen Elementen, wie beispielsweise einer Funktion und ihren Aufrufen gemacht werden. Da diese Informationen durch die Plattform und die Sprachspezifika der Quellplattform vorgegeben sind, kann auch dieser Schritt vollautomatisch erfolgen.

In einer vollautomatisierten Migration besteht der Analyseschritt aus Anwendungen von Regeln auf die Sourcen und den Code, welche aus einem fest definierten Regelkatalog stammen. Dies entspricht ebenfalls dem Ansatz der teilautomatisierten Migration, mit dem Unterschied, dass nach einem Durchlauf der Mustererkennung manuell nachgeschaut werden kann, wie vollständig die

Mustererkennung war und ob noch weitere Muster hinzugefügt werden müssen, um einen erfolgsversprechende Migration durchzuführen. Erst wenn ein Signal für die Zufriedenheit der Analyse gegeben wird, wird das Modell für den Abstraktionsschritt freigegeben.

Im Abstraktionsschritt kommt die eigentliche Stärke des modellgetriebenen Ansatzes zur Geltung. Durch die abstrakte Darstellungsweise kann man nun, vollkommen freigelöst von den Plattformen oder schon mit Sicht auf die Zielplattform, die Applikation restrukturieren, erweitern und überprüfen, ob und wie sich die gewünschten Komponenten in einer Komposition verhalten. Es ist an dieser Stelle nicht unbedingt nötig eine Abstrakte Repräsentation (im Sinne einer lauffähigen Applikation) zu erstellen, als mehr eine Grundlage für den Transformationsschritt zu erstellen. Durch die ständige Verbindung am initial geladenen Modells können nun die Strukturen, und Templates aus dem abstrakten Modell an den Transformator übergeben werden und die Inhalte der Elemente werden ähnlich zum vollautomatischen Migrationsprozess (wenn gewünscht) aus der Quellapplikation bereitgestellt.

Im Generationsschritt können nun die Zielapplikationselemente generiert werden. Durch die vorhergegangene Abstraktion können nun zusätzlich zu den benötigten „Low-Level“ Komponenten, sprich den technischen Komponenten der Zielapplikation auch strukturelle und architektonische Elemente wie Libraries, Applikationscluster, Navigationen und viele Querschnittsfunktionalitäten direkt aus dem Modell generiert werden.

What you see is what you get? Schauen wir mal etwas weiter.

Abschließend sollen an diesem Beispiel noch einmal die Ergebnisse der unterschiedlichen Migrationstypen verglichen werden. Eine vollautomatisierte Migration scheint was den initialen Aufwand anbelangt klar im Vorteil zu sein. Man drückt einen Knopf und nach kurzer Zeit ist die Zielapplikation fertig. Man läuft jedoch Gefahr, dass die Applikation nach erstem Blick nicht das liefert, was man erwartet hat. Unzusammenhängende oder falsch verknüpfte Dialoge, unnötige Kopien von einzelnen Komponenten, keinerlei Restrukturierungen in einer neuen Umgebung. Dies spiegelt noch einmal wieder, dass am Ende einer vollautomatisierten Migration zu sehr hoher Wahrscheinlichkeit keine “echte“ ADF Applikation erstellt wird.

Die modellgetriebene semiautomatisierte Migration bietet gerade zu Beginn einer Migration einen höheren Schwellenwert der Applikationsverständnis. Man arbeitet auf abstrakten Komponenten, bekommt nicht sofort eine endgültige Resonanz, wie sich die Applikation in der Zielarchitektur verhält, durch die vielen manuellen Schritte lässt sich der Weg zu einer Applikation wie man sie sich erwartet hat allerdings sehr gut ebnen. Jeder Entwickler, der sich in der Zielumgebung wohl fühlt wird Ihnen für die Entscheidung des semiautomatisierten Prozesses danken und auch der Einstieg für Entwickler des Quellsystems ist durch die Konzeption nahe an einer Referenzarchitektur einfacher.

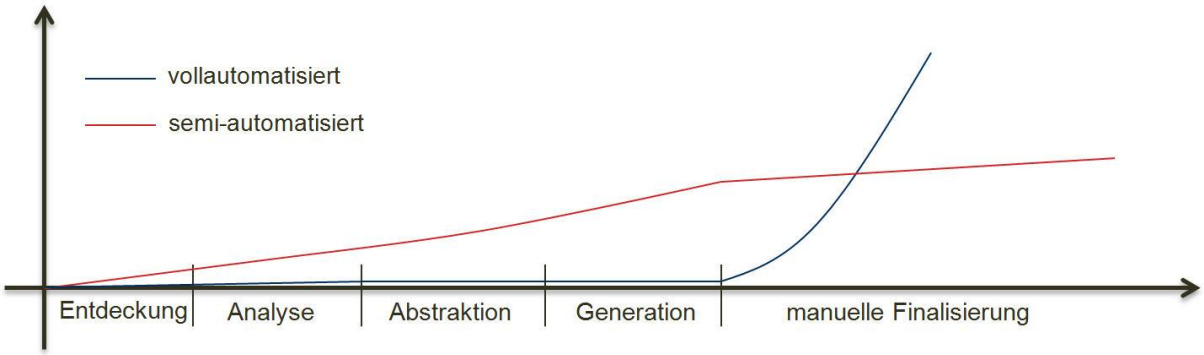

Abb. 3: Zeitaufwände der unterschiedlichen Migrationspfade

Durch die diversen Zwischenstopps kann bereits in den ersten Phasen der Migration an wiederverwendbaren Komponenten, mögliche Benutzerprozesse oder Code-Libraries gearbeitet werden, welche später in der Generation genutzt werden können. Und genau diese Features sorgen dafür, dass am Ende einer semi-automatisierten Migration wirklich eine Applikation erzeugt wird, die von einer manuell erstellten ADF Applikation kaum mehr zu unterscheiden ist, und das sollte schließlich das Ziel eines jeden Automatismus sein.

Kontaktadresse:

Markus Klenke

TEAM GmbH

Herrmann-Löns-Str 88

D-33104 Paderborn

Telefon:	+49 (0) 5254 / 8008 - 55
Fax:	+49 (0) 5254 / 8008 - 19
E-Mail	mke@team-pb.de
Internet:	www.team-pb.de