Aus Alt mach Neu

Markus Eisele
msg systems ag
Ismaning

Schlüsselworte:

Java EE 5, Java EE 6, J2EE, Migration, GlassFish, WebLogic

Einleitung

Die Enterprise Java Edition 5 ist schon seit einigen Jahren auf dem Markt und auch für den Nachfolger mit der Versionsnummer 6 ist die Referenzimplementierung bereits verfügbar. Die Zeit von Applikationsservern, welche die Java 2 Enterprise Edition unterstützen ist abgelaufen. Dazu gehören bei Oracle der OC4J genauso wie der Bea WebLogic Plattform (7.x – 10.2). Spätestens mit dem Erscheinen der Java Enterprise Edition 6 kompatiblen Version des WebLogic werden die alten Produktlinien End-Of-Life (EOL) im Extended Support. Auch wenn dies konkret für den WebLogic 8.1 noch bis zum März 2011 für 9.x bis November 2013 und den 10.2 sogar März 2015 bedeutet, sollte man sich aktuell schon die Frage stellen, was mit den etablierten Anwendungen älterer Bauart dann passieren soll. Sie alle müssen auf den Prüfstand und es muss entschieden werden ob und wie diese migriert werden sollen. Neben rein technischen Aspekten spielen hier noch weitere Dinge eine entscheidende Rolle. Den einzig richtigen Weg gibt es nicht. Allerdings kann anhand von Beispielen vergleichsweise einfach gezeigt werden, was die Optionen sind und welche Wege sich ergeben können. Die Ansätze hierzu sind vielfältig. Der Artikel soll einen Überblick über Migrationspfade und Strategien geben. Unter welchen Umständen ist es möglich und sinnvoll alte Konzepte zu übernehmen? Welche neuen Technologien sind verpflichtend? Wie sieht die richtige Entscheidung zur Migration aus?

Strategisches

Allen Fragen voran steht die nach dem Sinn. Warum sollten alte Anwendungen überhaupt angepackt werden? Vielfach ergibt sich der Bedarf aus Umständen von Produktkompatibilitäten und Support-Verträgen. So wird ein WebLogic 10.0 beispielsweise nicht mehr für das aktuelle SUSE Enterprise Linux 11 supportet. Ist man also gezwungen ein Update der Hardware durchzuführen und kommt nicht um das aktuelle Betriebssystem herum steht man schon vor der Entscheidung. Somit ist der erste Schritt jeder Überlegung auch die Betrachtung der aktuell in Betrieb befindlichen Komponenten mit den jeweiligen Versionen unter Berücksichtigung der vereinbarten Supportumfänge. Sind nicht alle Anwendungen betroffen, oder kann keine generelle unternehmensweite Entscheidung für eine zukünftige Zielplattform im Hinblick auf Produkt oder Technologie getroffen werden, kann nur zu einer separaten Betrachtung der einzelnen Migrationskandidaten geraten werden.
Grundsätzlich sind bei einer Migration auf eine neue Enterprise Java Version drei verschiedene Szenarien denkbar:

- Beibehalten des bisherigen Standards
- Mix zwischen alt und neu
- Komplette Migration auf neuen Standard

Behält man den bisherigen Standard bei, vertraut man auf die sogenannten Kompatibilitäsmodi der jeweiligen Produkte. Ein aktueller WebLogic 11g (10.3.3.0) ist beispielsweise durchaus in der Lage Container Managed Persistance (CMP) Entity Beans auszuführen, auch wenn diese Technologie in der

Java EE 5 durch die Java Persistance API (JPA) abgelöst wurde. Im einfachsten Fall sind dann nur ein paar Deployment Descriptoren auszutauschen und die Anwendung auf dem neuen Server zu deployen. Klar muss allerdings sein, dass man sich damit lediglich ein paar weitere Jahre Laufzeit sichert. Dieses Szenario kommt also maximal für Anwendungen in Betracht, welche eine fest definierte Laufzeit haben und für die keinerlei Weiterentwicklung geplant ist.

Ein Mix von Standards ist vielfach notwendig, wenn es sich um große Wartungsprojekte handelt oder ein Produktwechsel im Spiel ist. Soll eine Anwendung nicht nur technologisch auf den neuesten Stand gebracht, sondern auch beispielsweise vom GlassFish auf den WebLogic umziehen ist auch der einfache Kompatibilitätsmodus vielfach keine Option mehr, da in der J2EE noch vergleichsweise viel Raum für herstellerspezifische Erweiterungen vorhanden war. Ein einfaches Ändern der Deploymentdeskriptoren bleibt zwar auch hier theoretisch möglich, scheitert in der Praxis allerdings an zu vielen Stellen. Bei großen Projekten hat der Mix von Alt und Neu vor allem vor dem Hintergrund des Wartungsbudgets Vorteile. Hier sollte also problemorientiert vorgegangen werden, was konkret bedeutet, dass zumindest die Persistenzschicht aktualisiert wird und der Rest mehrheitlich unangetastet bleibt.

Eine komplette Umstellung auf alle beteiligten Java EE Standards kann schnell mehrstellige Personentage Aufwand verursachen. Neben Anpassungen in der Business Logik (CMP auf JPA oder EJB 2.x auf EJB 3.0) sind auch die Anwendungsoberflächen komplett umzubauen (JSP auf JSF). Vielfach ist hier auch die Unternehmens-IT Impulsgeber und die Projekte müssen den Technologiesprung mitmachen. Dennoch bleibt auch hier der Rat zum Pragmatismus. Nur in wenigen Ausnahmefällen lohnt sich der Kraftakt wirklich.

Analog des folgenden Entscheidungsdiagramms kann also schnell die grundsätzliche Richtung der Migrationsbemühungen festgelegt werden. In der überwiegenden Mehrheit der Fälle dürfte es auf einen Mix von Alt und Neu herauslaufen. Lediglich in besonders performancekritischen Projekten oder bei Projekten welche sich aktiv in Weiterentwicklung befinden lohnt sich eine komplette Modernisierung.

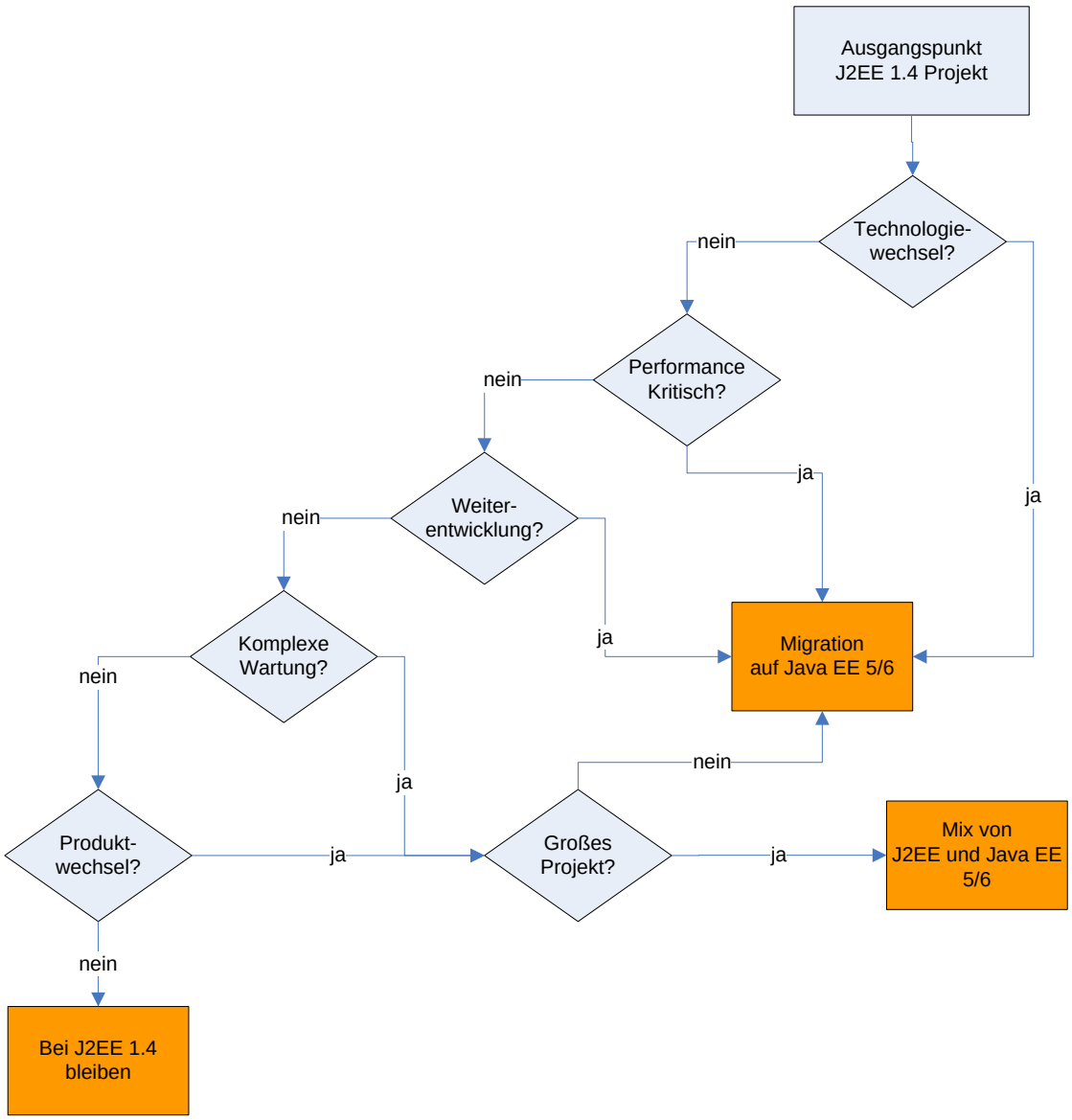
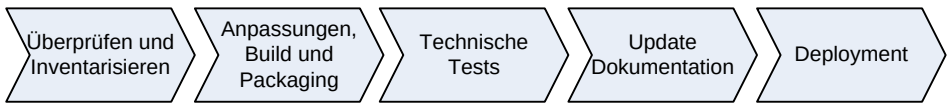


Abb. 1: Entscheidungsfindung

Grundsätzliches Vorgehen

Steht das Szenario fest, muss man sich über das Vorgehen Gedanken machen. Der hier vorgestellte fünf Phasen Plan umfasst die wichtigsten Aktivitäten für die jeweiligen Szenarien. In Abhängigkeit vom Szenario sind allerdings pro Phase entsprechend mehr Dinge zu tun.



+

Abb. 2: Migrationsschritte

Begonnen wird mit dem konkreten Überprüfen und Inventarisieren der Anwendung. Dabei sollten alle relevanten Themen im Fokus stehen. Eine J2EE Anwendung besteht aus unterschiedlichen Anteilen. Neben Konfigurationsdateien und Frameworks auch aus den Java Sourcen der Anwendung. Zumeist findet auch ein umgebungsspezifischer Build statt. Manchmal werden Funktionen der jeweiligen Applikationsserver Plattform verwendet, welche in einer neuen Version ausgetauscht werden müssen, um die Ablauffähigkeit zu garantieren. Alle genannten Anteile müssen für die Migration in geeigneter Weise angepasst werden. In dieser Migrationsphase sollte daher eine komplette Liste aller eingesetzten Frameworks und Komponenten erstellt werden. Dazu gehört auch die Untersuchung des Sourcecodes auf evtl. Inkompatibilitäten auf Ebene der JDK Version. Die Praxis zeigt, dass hier an nur wenigen Stellen eingegriffen werden muss. So sind mit dem JDK 1.5 beispielsweise neue Klassen hinzugekommen (java.net.Proxy). Auch die Einführung von typsicheren Enumerationen sorgt für Probleme. Der Variablenname enum darf nun nicht mehr auftauchen. Eine detaillierte Liste wird aber von Oracle bereitgestellt. Ebenfalls problematisch sind sogenannte „deprecated features“. Dies bezeichnet Funktionalitäten, welche beim Wechsel von einer Produktversion auf die nächste weggefallen sind. Für den Oracle WebLogic Server findet sich diese Informationen klassischerweise leider nur zwischen den Zeilen in den jeweiligen „What’s New“ Dokumenten. Der GlassFish v3 bringt ein wenig Ordnung mit einer eigenen API-Doc Seite. Am Ende dieser Phase steht eine konkrete Liste aller Dinge, welche für das Konkrete Migrationsvorhaben zu ändern sind.
Diese Liste stellt auch automatisch die ToDo Liste für die konkreten Anpassungen dar. Im gleichen Zuge ist natürlich auch der Build der Anwendung anzupassen. Neben evtl. neuen Server Bibliotheken sind auch die Framework Abhängigkeiten anzupassen. Egal um welche Art von Migration es sich handelt, lohnt sich eine Adaption der aktuellen Build Werkzeuge. Der Autor ist bekennender Maven Anhänger und rät daher zur konsequenten Umstellung aller alten ANT Builds. Berücksichtigt werden müssen beim Anpassen des Builds darüber hinaus noch entsprechende Staging Konzepte oder Vorgaben für die Auslieferung.
Tests stellen einen elementaren Bestandteil jeder Migration. Je nach Szenario kann hier von vereinfachten, technischen Tests bis hin zum kompletten fachlichen Regressionstest alles vorkommen. Ein absolutes Minimum stellt dabei die korrekte Lauffähigkeit der JUnit Tests dar. Diese sind daher in dieser Phase auch konkret anzupassen. Auch empfehlenswert ist die Durchführung von Last- und Performance Tests. Hierbei werden nicht selten produktionskritische Probleme aufgedeckt, welche noch im Rahmen der Migration zu beheben sind. Alle weiteren Testabstufungen bis hin zum vollumfänglichen Regressionstest befriedigen das unterschiedliche Sicherheitsbedürfnis der Verantwortlichen und können daher auch nicht generisch in diesem Rahmen definiert werden.
Gerne vergessen wird auch das Update der Dokumentation. Auch hier kann zwischen einem simplen Update der Release-Notes bei der Verwendung des J2EE Kompatibilitätsmodus bis hin zur kompletten Überarbeitung der Projektdokumentation (Design, Architektur, Betrieb, etc.) alles notwendig werden. Diese Phase liegt bewusst vor dem tatsächlichen Deployment um ihr entsprechende Bedeutung zuzumessen.
Deployment im Sinne dieses Phasenmodells stellt die erfolgreiche Übergabe der Anwendung in den Linienbetrieb dar. Wie dies konkret geschieht und auf welche Umgebung(en) ist stark geprägt von den jeweiligen Unternehmensvorgaben.

Probleme und Risiken

Bei jeder Migration existieren Gefahren bzw. Stolpersteine. Auch wenn man diese nie vollständig im Vornerein benennen kann, sollte man sich auf die Wichtigsten einstellen. Ihr Vorhandensein und die Auswirkungen auf die konkrete Migration sind direkt abhängig von der Komplexität des Projekts und sie treten daher auch in unterschiedlicher Schwere zu Tage.
Größtes Problemfeld stellen die herstellerspezifischen Erweiterungen dar. Sind sie im Zuge einer Migration ohne Produktwechsel vielfach noch beherrschbar, stellen diese anderenfalls ein großes Risiko für die Migration dar. Gerade im Umfeld der J2EE ließ der Standard noch vergleichsweise

viele Freiheiten für die Hersteller bei der Implementierung unterschiedlicher Spezifikationen. Somit ist es nur wahrscheinlich, dass die besonderen Funktionen auch genutzt wurden. Was auf dem WebLogic neueren Datums dann auch vielfach funktioniert gibt es dann beispielsweise auf dem brandaktuellen GlassFish überhaupt nicht. Verdächtige sind hier nur mit Spezialwissen zu identifizieren. Für eine Anwendung, welche im WebLogic Server läuft könnten das beispielsweise die Folgenden sein:

• WebLogic eigene JSP Taglibraries
• Verwendete com.bea.* oder weblogic.* Packages
• Verwendung der Split development Structure

Aber bereits mit einfachen Open Source Frameworks können Probleme auftreten. So sind beispielsweise Apache Struts Versionen vor 1.0 nicht mit dem JDK 1.5 lauffähig. Konkret steht hier eine Abhängigkeit zu einer Funktionalität aus dem JDK 1.3 im Weg welche erst mit neueren Frameworksversionen behoben wurde. Daher ist es unumgänglich alle im Projekt vorhandenen Frameworks auf ihre Kompatibilität mit der Zielplattform genauer zu untersuchen.

Die größten Risiken bergen allerdings hausgemachte Themen. Allen voran steht eine schlechte Codequalität. Was unübersichtlich und wenig nach OO Prinzipien designed wurde birgt dabei die größten Probleme. Diese Situation kann nur noch durch das Fehlen jeglicher automatisierter Tests verschlimmert werden. Trifft man auf einen solchen Problemfall bleibt ungeachtet aller vorgenannter Überlegungen zumeist nur die Entscheidung zwischen so weitermachen wie bisher oder komplett neu machen. Ähnlich dramatisch kann sich eine schlechte Architektur auswirken. Das Fehlen von Designpattern.

- Schlechte Architektur
- Schlechte oder keine Dokumentation

Zusammenfassung

Der Aufwand einer Migration hängt grundlegend von der Architektur der Anwendung ab. Dabei spielen viele unterschiedlichen Kriterien eine Rolle. Schwerpunkt von gemischten oder kompletten Java EE 5/6 Migrationen bildet sicherlich die Persistenzschicht. Ist diese bereits in der Vergangenheit durch die Verwendung von Java Design Patterns (Session Facade, Data Transfer Object usw.) entsprechend gekapselt worden ist jede Migration beherrschbar. Dennoch bleiben Risiken welche durch stringente Überwachung minimiert werden können.

Die Seitenzahl wird von uns eingefügt!

Bitte fügen Sie Ihre Kontaktadresse hinzu.

Kontaktadresse:

Markus Eisele
msg systems ag
Robert-Bürkle-Str.1
D-85737 Ismaning

Telefon:	+49 (0) 89-96 101 0
Fax:	+49 (0) 89-96 101 2315
E-Mail	markus.eisele@msg-systems.com
Internet:	www.msg-systens.com
Internet:	blog.eisele.net