Puzzle ITC GmbH
Eigerplatz 4
CH-3007 Bern
Telefon: +41 31 370 22 00
Telefax: +41 31 370 22 01
www.puzzle.ch

Erfahrungsbericht einer Java EE Migration

Von Diminic Brügger, Puzzle ITC GmbH – 04.11.2013

Für einen Kunden durfte Puzzle ITC eine Java Anwendung auf JBoss EAP 6 portieren. Dominic Brügger zeigt im folgenden Beitrag auf, wie die Applikation schrittweise von Drittanbieter-Bibliotheken gelöst und zu einer Standard Java EE 6 Anwendung migriert wird.

Zu Beginn stellt sich die Frage, ob eine solche Migration sinnvoll und wirtschaftlich ist. Die Antwort hängt von den Rahmenbedingungen des Projektes ab und kann nicht generell auf andere Fälle übertragen werden. In unserem Fall haben zwei wesentliche Faktoren die Entscheidung für eine Migration begünstigt:

1.) Der Auftraggeber hat sich strategisch für JBoss EAP 6 als Plattform für Java Geschäftsapplikationen entschieden. Bereits sind mehrere Anwendungen auf dieser Umgebung produktiv im Einsatz. Mit der Migration wird die Software Architektur und das Deployment an die bestehenden Anwendungen angeglichen. Mittelfristig verspricht man sich dadurch eine Reduktion der Betriebs- und Wartungskosten.

2.) Mit der Migration werden gleichzeitig Verbesserungen an der Softwarearchitektur umgesetzt. Dieses Thema werden wir im nächsten Abschnitt erörtern.

Vereinfachend kommt hinzu, dass das Benutzerinterface aus einem Swing Thin Client besteht und somit von der Migration ausgeschlossen werden kann. Die Umsetzung der Migration haben wir schliesslich in drei aufeinander aufbauenden Schritten geplant. Diese Strategie stellt sicher, dass die Applikation zu jedem Zeitpunkt lauffähig bleibt und die einzelnen Schritte isoliert getestet werden können.

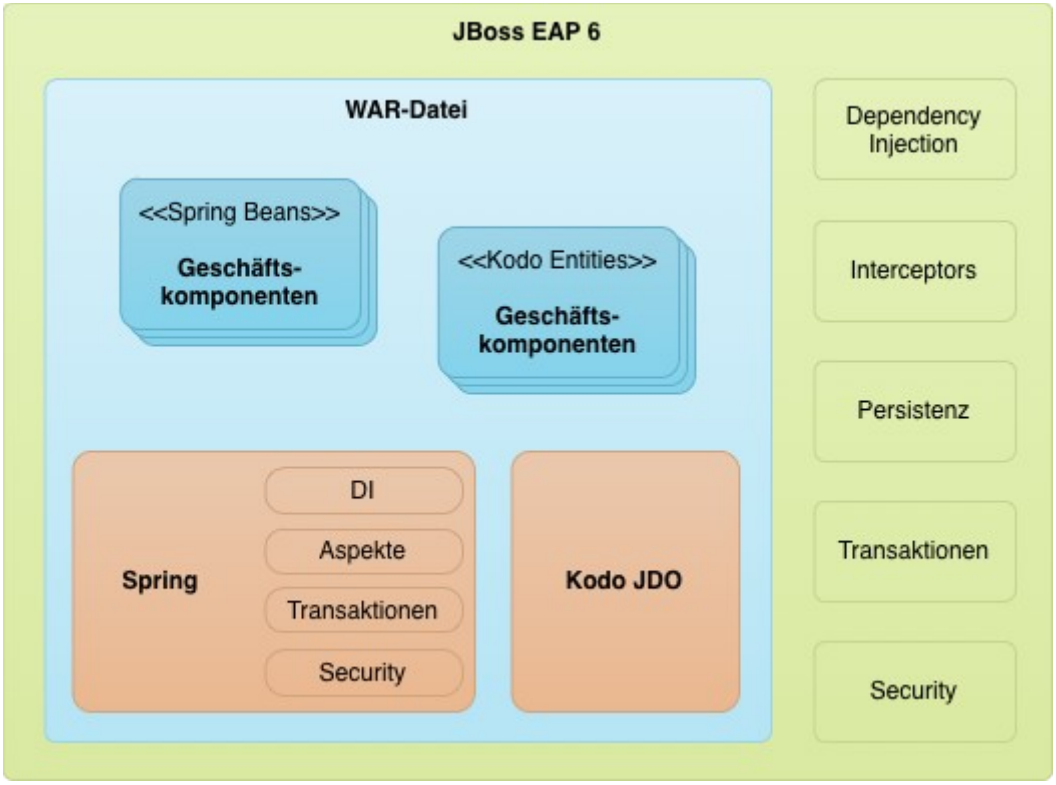
1. Schritt: Anpassen der Architektur

Im ersten Migrationsschritt passen wir die bestehende Applikation so an, dass sie optimal auf einem Java EE 6 Container deployed werden kann. Ausgangslage ist eine Softwarearchitektur bestehend aus einem Thin Client, einem Schnittstellenmodul und einem Servermodul. Das Schnittstellenmodul stellt dem Thin Client eine REST-Schnittstelle zur Verfügung. Die Ausführung der

eigentlichen Geschäftslogik wird über eine RMI-Schnittstelle an das Servermodul delegiert. Das Deployment der Schnittstellenschicht auf einem dedizierten Server bringt weder technische noch fachliche Vorteile und wird bei der neuen Architektur weggelassen. Die REST-Schnittstelle integrieren wir in das Servermodul und ersetzen die RMI-Schnittstelle durch direkte Service-Aufrufe. Das zuvor als Standalone Java-Anwendung konzipierte Servermodul packen wir neu als WAR-Datei.

Mit dem Zusammenlegen der serverseitigen Module überarbeiten wir ebenfalls das Konfigurationsmanagement. Die gesamte Applikation wird neu auf dem Applicationserver durch System Properties konfiguriert. Gegenüber den bisherigen verteilten Konfigurationsdateien ist dies ein wesentlicher Vorteil.

Nach dem ersten Migrationsschritt kann die Applikation bereits auf JBoss EAP 6 ausgeführt und getestet werden. Die Geschäftskomponenten basieren jedoch weiterhin auf dem alten Technologiestack und schöpfen das Potential des Applicationservers nicht aus. Als nächsten Migrationsschritt werden wir deshalb die Persistenzschicht auf den JPA Standard migrieren.



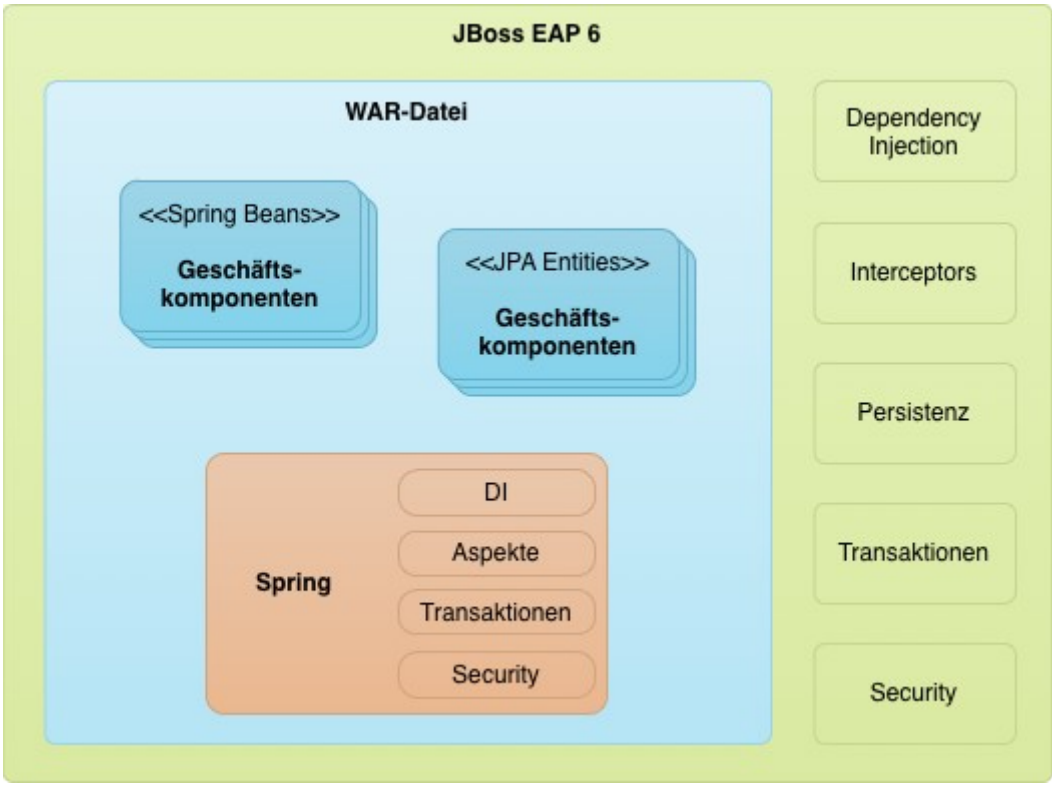
Nach dem 1. Schritt kann die Anwendung auf JBoss EAP 6 ausgeführt werden. Wichtige Dienste wie Dependency Injection, Persistenz und Transaktionen werden aber weiterhin von Spring und Kodo bezogen.

2. Schritt: Migration Persistenzschicht

In der bestehenden Applikation ist der Datenzugriff mit Kodo JDO umgesetzt. Im zweiten Migrationsschritt lösen wir dieses proprietäre objektrelationale Framework durch den JPA Standard ab und können somit den Persistenzdienst vom Java EE Container nutzen.

Als Erstes müssen wir die Mappingdefinitionen umschreiben, das heisst die Definitionen aus den Kodo Konfigurationsdateien als JPA Annotationen abbilden. Das stellt sich als grössere Herausforderung dar, da die Mappings komplex sind und das bestehende Datenbankdesign nicht immer optimal für JPA ausgelegt ist. Nach den Mappingdefinitionen migrieren wir sämtliche Abfragen. Kodo Queries schreiben wir zu JPQL Queries um. Ein grosser Teil der Abfragen besteht aus nativen SQL Queries, welche mit wenigen Anpassungen übernommen werden können. Das Testing und das anschliessend nötige Performancetuning entpuppen sich als weitaus grösster Aufwand. Es muss auf jeden Fall sichergestellt werden, dass sich das Laufzeitverhalten der Applikation nicht verändert.

Nach diesem Umbau verwenden wir mit JPA das Persistenzframework des Applicationservers. Die JPA EntityManagerFactory und die Transaktionen werden aber immer noch vom Spring Framework und nicht vom Container verwaltet. Diese Baustelle gehen wir im nächsten und letzten Migrationsschritt an.

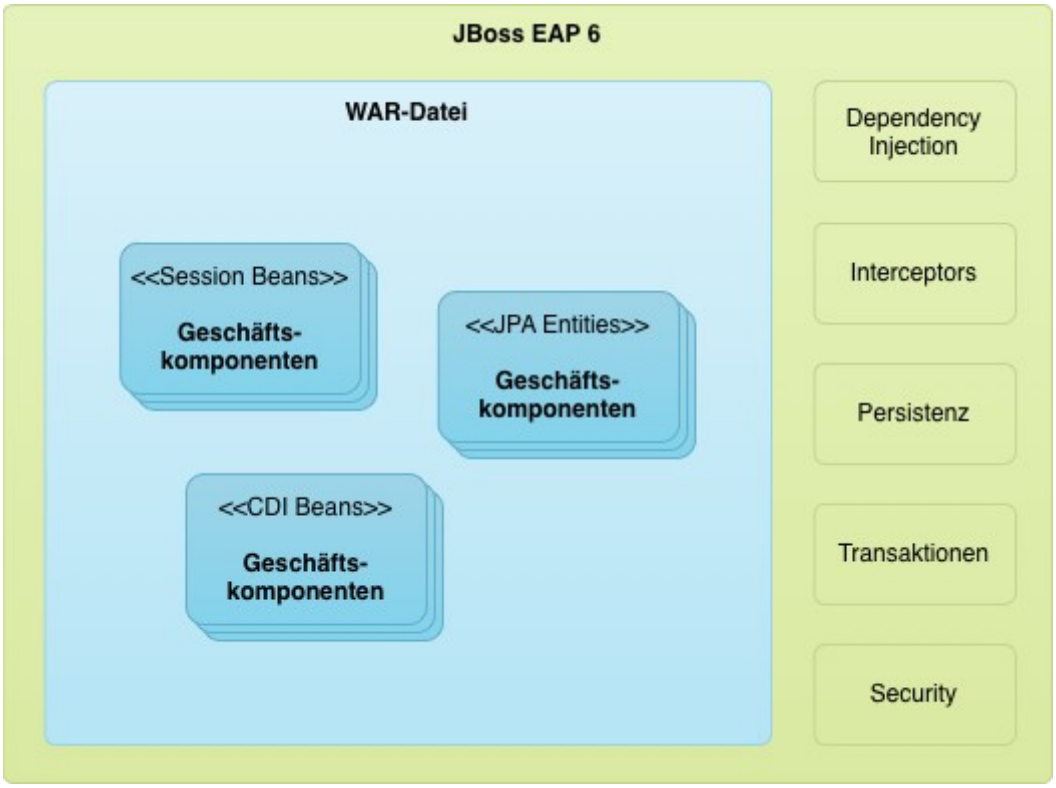


Nach dem 2. Schritt ist die Persistenzschicht mit dem JPA Standard implementiert. Kodo JDO wurde ausgebaut.

3. Schritt: Spring durch EJB und CDI ersetzen

Die bestehende Anwendung verwendet das Spring Framework für Dependency Injection und Transaktionsmanagement. Im letzten Migrationsschritt lösen wir Spring Beans und Aspekte durch das Java EE Komponentenmodell ab. Das geht relativ geradlinig vonstatten, indem wir Spring Beans entweder durch Stateless Session Beans oder durch CDI Beans ersetzen. Grob gesagt werden jene Beans, die mit dem Spring Transaktionsaspekt ausgestattet waren, zu Session Beans migriert. Diese sind als EJBs standardmässig transaktional (Container Managed Transaction). Alle anderen Spring Beans ersetzen wir durch CDI Beans. Bei dieser Arbeit tauchen als einzige kleine Überraschung zirkuläre Abhängigkeiten zwischen Spring Beans auf. Spring scheint in diesem Fall toleranter zu sein als der Java EE Container, weshalb diese Zyklen noch bereinigt werden müssen.

Erfreulicherweise können einige Hintergrundtasks, welche zuvor auf Spring Scheduling und Quartz basierten, sehr einfach mit dem EJB Timer Service umgesetzt werden. Ein Vorteil dabei ist, dass die Task Threads vom Application-server verwaltet werden und sich somit besser kontrollieren und überwachen lassen. Am Ende dieses Migrationsschrittes kann das Spring Framework entfernt werden. Die Serverseite der Anwendung basiert nun vollständig auf Java EE Standardtechnologien.



Nach dem 3. und letzten Schritt basieren sämtliche Geschäftskomponenten auf den Java EE APIs. Es werden keine Drittanbieter-Bibliothenken mehr benötigt.

Eine Erfolgsgeschichte

Im Grossen und Ganzen dürfen wir mit dem Resultat zufrieden sein. Die meisten Migrationsschritte konnten zügig und ohne Überraschungen umgesetzt werden. Die grösste Herausforderung war die Migration der Persistenzschicht. Das JPA-Mapping für das bestehende Datenbankmodell zu definieren war schwieriger als erwartet. Als wesentliche Verbesserung sehen wir das Deployment und die Konfiguration der Anwendung. Zuvor war dies ein fehleranfälliger Prozess mit einem Installerprogramm, welcher auf mehreren Servern ausgeführt werden musste. Neu liefern wir dem Betrieb eine WAR-Datei und ein dokumentiertes Set von System Properties. Um die Verteilung der Applikation kümmert sich JBoss EAP 6.

Durch die Verwendung von Standards gewähren wir unserem Kunden einen hohen Investitionsschutz. Java EE ist die bewährte Plattform für komplexe Geschäftsanwendungen.

Kontakt:

Puzzle ITC GmbH **www.puzzle.ch**

André Kunz **kunz@puzzle.ch**

Eigerplatz 4

3007 Bern **031 370 22 00**