Softwarearchitektur für Geschäftsanwendungen auf Basis von OSGi

Dehla Sokenou
GEBIT Solutions, Koenigsallee 75b, D-14193 Berlin
dehla.sokenou[at]gebit.de

1 Motivation

Werden wiederholt kundenspezifische Lösungen für fachlich ähnliche Anforderungen entwickelt, bieten domänenspezifische Plattformen große Produktivitäts- und Qualitätsvorteile. Eine tragfähige Komponentenarchitektur muss die gemeinsame Fachlichkeit im Plattform-Layer kapseln und Individualisierungen flexibel und wartungsfreundlich koppeln.

Ein typisches Beispiel für dieses Problem sind Kassensysteme (POS-Systeme). An diese werden folgende Anforderungen gestellt: Aus Softwareherstellersicht spielt Wiederverwendung und Wartbarkeit eine große Rolle. Aus Kundensicht sind in der Regel eine Reihe individueller Lösungen zu realisieren: Implementierung kundenspezifischer Workflows, variable Anpassung an verschiedene Umgebungen, z.B. wenn unterschiedliche Hardware eingesetzt wird, insbesondere auch die Integration desselben Treibers in unterschiedlichen Versionen für unterschiedliche Geräte desselben Herstellers, Einhaltung gesetzlicher Vorgaben und Länderspezifika, z.B. Fiskalisierung.

Es entsteht also ein Spannungsfeld zwischen der Standardisierung einerseits und der Individualentwicklung andererseits. In der bestehenden POS-Plattform wurden zwar auch bisher Technologien eingesetzt, um diese Anforderungen zu erfüllen, es handelte sich jedoch nicht um einen einheitlichen Ansatz, was zunehmend zu einer Verschlechterung der Wartbarkeit führte und die Weiterentwicklung erschwerte.

2 Lösungsansatz und Vorgehen

Um diese Anforderungen möglichst gut zu erfüllen, wurden verschiedene Ansätze für Java evaluiert, u.a. Context and Dependency Injection (CDI) der Java EE Plattform, Dependency Injection Frameworks wie Spring und die Komponentenplattform OSGi.

Als einzige Alternative bietet OSGi die Möglichkeit, Komponenten nicht nur statisch zu erzeugen und einzubinden, sondern zudem dynamisch zu aktivieren und deaktivieren und Komponenten erst zur Laufzeit einzubinden. Da diese Fähigkeit für die bestehende POS-Plattform unabdingbar war, wurde OSGi als Komponentenplattform gewählt.

Im Wesentlichen erfolgte die Umstellung der POS-Plattform in drei Schritten. Zunächst wurden die bestehenden POS-Komponenten in OSGi-Bundles umgewandelt. Nach diesem Schritt hatte man zunächst wieder ein lauffähiges System, was allerdings noch nicht die Vorteile von OSGi im vollem Umfang nutzen konnte, da viele der so erstellten OSGi-Bundles noch zu viele Abhängigkeiten hatten oder nicht optimal geschnitten waren, um in allen Fällen von der Fähigkeit der dynamischen Aktivierung zu profitieren. Zudem war noch Code vorhanden, der zuvor dazu diente, die richtigen Services anzusteuern bzw. die richtigen Komponenten zu laden. Die Anpassung und Bereinigung dieser Codestellen wurde in einem zweiten Schritt durchgeführt. Im dritten Schritt wurde der bestehende Code analysiert und, sofern notwendig oder sinnvoll, wurden einzelne POS-Komponenten auf mehrere OSGi-Bundles verteilt oder anders geschnitten.

3 Erfahrungen und Ergebnisse

Die POS-Plattform ist bereits bei mehreren Kunden im Einsatz. Allerdings basieren erst neuere Versionen auf OSGi. Trotz allem konnten während der aktuellen Entwicklung und dem Einsatz beim Kunden bereits einige Ergebnisse der Umstellung beobachtet werden.

Im Zuge der Umstellung wurden insbesondere folgende Aspekte betrachtet und bewertet:

Entwicklung: Komponenten werden wie bisher unabhängig voneinander entwickelt. Aufwand entsteht bei der Konfiguration des kunden- und länderspezifischen Systems, allerdings ist dieser Aufwand geringer als zuvor, da viele Arbeiten, z. B. das Auflösen referenzierter Services, bereits durch die OSGi-Plattform übernommen werden.

Debugging, Fehleranalyse: OSGi bietet zwar eine OSGi-Konsole, die zur Analyse von z.B. fehlerhaft nicht aktivierten Komponenten verwendet werden kann, allerdings ist diese Analyse meist sehr mühsam und im Fall, dass die OSGi-Plattform aufgrund fehlerhafter Konfiguration gar nicht erst hochgefahren wird, deutlich erschwert, da dann auch die OSGi-Konsole nicht zur Verfügung steht.

Performance: die dynamische Aktivierung von Komponenten hat keinen Einfluss auf die Einhaltung der geforderten Performanceanforderung, allerdings war ein erhöhter Optimierungsbedarf im Vergleich zu vorher festzustellen, um dieses Ziel zu erreichen.

Wartbarkeit: durch viele kleine, lose gekoppelter Komponenten, die jeweils einen Aspekt kapseln, ist die POS-Plattform deutlich einfacher zu erweitern und zu pflegen.

Testbarkeit: hier mussten einige Anpassungen an den eingesetzten Testframeworks vorgenommen werden, um diese OSGi-fähig zu machen. Der Testaufwand ist nicht höher, denn auch zuvor mussten alle Konstellationen getestet werden.

Zusammenfassend lässt sich aussagen, dass der Einsatz von OSGi in Geschäftsanwendungen lohnt, die eine dynamische Konfiguration des Systems zur Laufzeit verlangen. Andere Alternativen bieten zwar statische Konfiguration zur Compile- oder Startup-Zeit an; erfordert das zu implementierende System eine dynamische Konfiguration, gibt es zum Einsatz von OSGi im Augenblick keine Alternative.