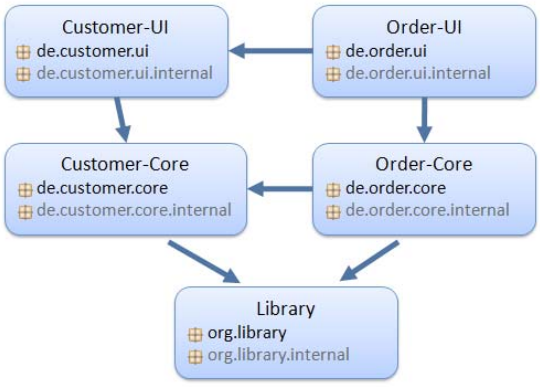
OSGi
modulare Softwareentwicklung mit Java

Newsletter

August 2008

Einleitung

Softwaresysteme werden größer und komplexer, Änderungen sollen schnell und ohne Nebenwirkungen umgesetzt werden, und Programmteile sollen wiederverwendet werden. Aus diesen und anderen Anforderungen folgt die Notwendigkeit, große Java-Systeme zu modularisieren. Obwohl diese Notwendigkeit bereits 1972 von David Parnas erkannt und definiert wurde, bietet Java von Hause aus keine Unterstützung für Modularisierung, Information Hiding und Kapselung auf einer grobgranularen Ebene an.
OSGi springt hier in die Bresche und bietet ein dynamisches Modulsystem für Java. OSGi erlaubt es, Module in Java zu definieren und für diese Module sowohl Sichtbarkeiten als auch Abhängigkeiten explizit zu definieren. Ein OSGi Modul besteht in der Regel aus mehreren Klassen in mehreren Packages und einer Modulbeschreibung. In dieser Modulbeschreibung werden die Abhängigkeiten zwischen Modulen definiert und Sichtbarkeiten zwischen Modulen eingeschränkt. Damit wird es in Java möglich, die Idee der Modularisierung auch auf größere Einheiten als Klassen anzuwenden.



Ein typisches Schaubild einer OSGi-Anwendung: Die Anwendungslogik ist in einzelne Module aufgeteilt, es gibt klare Abhängigkeiten zwischen den Modulen, und die Sichtbarkeit zwischen den Modulen ist eingeschränkt. So sind z.B. die Klassen aus dem Package de.customer.core.internal nicht nach außen sichtbar.

Neben den Abhängigkeiten und Sichtbarkeiten zwischen Modulen definiert OSGi einen Lebenszyklus für Module. Damit wird es möglich, einzelne Module zur Laufzeit hinzuzufügen, auszutauschen, zu aktualisieren oder ganz aus dem System zu entfernen. Aufbauend auf diesem dynamischen Verhalten definiert OSGi ein leichtgewichtiges Service-Modell und bietet die entsprechende Infrastruktur an. Hierdurch lassen sich Module schreiben, die in einer dynamischen Umgebung sauber arbeiten und damit ein Gesamtsystem bilden, welches vollständig dynamisch verwaltet werden kann.

OSGi gestern und heute

OSGi ist im Vergleich zu anderen Java Technologien relativ alt. Bereits 1999 wurde von der OSGi Alliance [0] die erste Version der OSGi-Spezifikation erarbeitet. Der damalige Fokus lag in der Bereitstellung und Verwaltung von Services in lokalen Netzen und Geräten. Dabei sollten neue Geräte in einem Netzwerk über OSGi neue Dienste für Andere zur Verfügung stellen. Interessenten eines Dienstes sollten über den Status des Dienstes zur Laufzeit informiert werden, u. a. zum Beispiel wenn ein Dienst nicht mehr verfügbar ist. Jahrelang war OSGi vor allem im Bereich Netzwerkgeräte und Embedded Devices vorzufinden. Ein prominentes Beispiel für die Verwendung der OSGi-Technologie im Embedded-Bereich ist sicherlich die BMW-5er-Limousinen, deren Entertainment-System auf Basis von OSGi funktioniert.

Im Jahr 2003 entschied das Entwicklungsteam der Eclipse IDE [1], das selbst entwickelte Komponentensystem auf OSGi umzustellen. Seit der Version 3.0 basiert damit jedes Eclipse Produkt auf OSGi. Da Eclipse mittlerweile auch in einer Vielzahl von Anwendungsgebieten außerhalb von IDEs eingesetzt wird, wird Equinox [2], die entsprechende OSGi-Implementation von Eclipse, in einer Vielzahl von Projekten und Produkten eingesetzt (die Beispiele reichen von Lotus Notes bis zur NASA). Neben Equinox existieren eine Menge weiterer OSGi-

Implementationen wie Apache Felix [3], Knopflerfish [4] und mBedded Server von Prosyst [5].

Mittlerweile ist OSGi im Java Enterprise angekommen. So benutzen große Application Server wie WebSphere [6], Weblogic [7], Glassfish [8] und JBoss [9] bereits OSGi im Kern oder werden dies in kürze tun. Durch die SpringSource Application Platform [10] gibt es in diesem Bereich einen neuen Mitstreiter, der nicht nur intern selbst auf Basis von OSGi realisiert ist, sondern es auch ermöglicht, Enterprise-Anwendungen direkt als OSGi-Module zu betreiben. Eine eigene OSGi Expert Group [11] arbeitet neue Anforderungen und Spezifikationen aus, die durch vermehrten Einsatz im Enterprise Bereich entstehen.

Erste Schritte in OSGi

Die ersten Schritte in OSGi fallen im Gegensatz zu anderen Java Enterprise Technologien relativ leicht. Zunächst muss man sich eine entsprechende OSGi-Runtime installieren. Die Benutzer der Eclipse IDE können diesen Schritt überspringen und mit der in Eclipse integrierten OSGi-Console arbeiten (siehe [12]). Aber auch, wenn man direkt mit einer OSGi-Runtime arbeiten möchte, ist die Installation in der Regel recht einfach und reduziert sich auf den Download einer Jar Datei.

In OSGi werden Anwendungen in einzelne Module, so genannte Bundles, unterteilt. Ein Bundle besteht in der Regel aus einer Reihe von Java-Klassen und einer Bundle-Beschreibung, die gemeinsam in Form eines normalen Jars ausgeliefert werden. Die Bundle Beschreibung befindet sich in der aus der Jar-Spezifikation bekannte Manifest-Datei: `META-INF/MANIFEST.MF`.

Ein einfaches HelloWorld-Bundle würde sich demnach aus folgenden beiden Artefakten zusammensetzen:

Das Manifest hätte folgende Einträge:

```
Manifest-Version: 1.0
Bundle-Name: HelloWorld
Bundle-Activator:
de.itagile.HelloActivator
Bundle-SymbolicName: HelloWorld
Bundle-Version: 1.0.0
Import-Package: org.osgi.framework
```

Eine Klasse die automatisch gerufen wird sobald das Bundle gestartet wird, würde wie folgt aussehen:

```
package de.itagile;
import org.osgi.framework.*;
public class HelloActivator imple-
ments BundleActivator {
  public void start(BundleContext
context) {
    System.out.println("Hello!");
  }
  public void stop(BundleContext
context) {
    System.out.println("Goodbye.");
  }
}
```

Wenn Sie die kompilierte Klasse zusammen mit dem Manifest in ein Jar verpacken, können Sie das Bundle in einer laufenden OSGi-Runtime installieren, starten und beenden.

Weitere Schritte mit OSGi

Das Beispiel soll zeigen, dass es relativ einfach ist, einen Einstieg in OSGi zu finden. Echte Vorteile für die eigene Entwicklung entstehen, wenn man die Konzepte von OSGi konsequent einsetzt:

Sichtbarkeit: So könnten in unserem Beispiel weitere Packages wie de.itagile.internal existieren. Nun sollte die Sichtbarkeit nach außen eingeschränkt werden und lediglich das Package de.itagile über folgenden Eintrag exportiert werden: `Export-Package: de.itagile`. Die Vorteile sind offensichtlich: In den nicht exportierten Packages kann refactored werden, ohne die Zusammenarbeit mit anderen Modulen zu gefährden und bei der Benutzung dieses Bundles sind nur die öffentlichen Packages relevant.

Versionisierung: Es kommt eine neue Version (2.0) einer Bibliothek heraus, die leider mit der im System benutzten Version (1.0) inkompatibel ist. Wenn diese Bibliothek nun in Bundles gekapselt ist, können beide Versionen gleichzeitig im System installiert werden. Die alte Version würde man wie folgt referenzieren: `Import-Package: org.library;version="[1.0.0,2.0.0)"`. Die neue Version entsprechend: `Import-Package: org.library;version="2.0.0"`

Services: Mit dem leichtgewichtigen Servicemechanismus lassen sich Implentierung und Definition von Objekten gut trennen, und so werden Module weiter entkoppelt. Die verwendete Service Registry ist insbesondere für dynamische Systeme ausgelegt. So lassen sich hiermit leichter Anwendungen entwickeln, bei denen einzelne Module zur Laufzeit ausgetauscht werden.

Aufbauend auf den vorgestellten Features von OSGi gibt es eine Reihe von weiteren Technologien und Frameworks, die den Umgang mit den vorgestellten Konzepten weiter vereinfachen. So gibt es beispielsweise die Declarative Services [13] oder die Spring Dynamic Modules [14], die es erlauben, Services deklarativ zu beschreiben.

Zusammenfassung

OSGi bietet ein dynamisches Modulsystem für Java. Hiermit können aktuelle Java-Anwendungen in einzelne Module unterteilt und diese zur Laufzeit aktualisiert oder ausgetauscht werden. Es können klare Modulgrenzen beschrieben werden, und so bei der Entwicklung Abhängigkeiten und damit Aufwände besser analysiert werden. Diese Konzepte erlauben neue Anwendungsarchitekturen, die OSGi durch ein leichtgewichtiges Servicekonzept weiter unterstützt. Da OSGi auf bestehenden Standards aufbaut, erlaubt es einen einfachen Einstieg, auch bei bestehenden Anwendungen. OSGi ist von der OSGi-Alliance ausführlich spezifiziert und wird durch eine Reihe von namhaften Firmen unterstützt. Zudem existieren mehrere Open-Source- und kommerzielle Implementierungen.

OSGi – modulare Softwareentwicklung mit Java

Newsletter vom August 2008

Eine detaillierte Einführung in OSGi finden Sie in dem kürzlich erschienenen Buch: „Die OSGi Service Platform - Eine Einführung mit Eclipse Equinox“. Das Buch vermittelt einen fundierten Überblick über die zugrunde liegenden Technologien, Begriffe und Konzepte. Tutorials demonstrieren praktisch die einzelnen Konzepte am Beispiel von Eclipse Equinox. Mehr unter: http://www.osgibook.org

Referenzen

[0] www.osgi.org
[1] www.eclipse.org
[2] www.eclipse.org/equinox
[3] http://felix.apache.org
[4] www.knopflerfish.org
[5] www.prosyst.de
[6] www.ibm.com/websphere
[7] www.bea.com/products/weblogic/server
[8] https://glassfish.dev.java.net/
[9] http://www.jboss.org/
[10]:http://www.springsource.com/products/suite/applicationplatform
[11] http://www.osgi.org/EEG/
[12]:http://www.128.ibm.com/developerworks/opensource/library/os-ecl-osgiconsole/
[13] http://www.aqute.biz/Snippets/HelloWorldComponent
[14] http://www.springframework.org/osgi

Unser Angebot zu OSGi

- Schulungen – auch inhouse
- Coaching
- Unterstützung durch erfahrene OSGi-Entwickler
- Komplette Projekte

Weitere Informationen erhalten Sie gerne unter
agile@akquinet.de

Kontakt: akquinet AG ▪ Paul-Stritter-Weg 5 ▪ 22297 Hamburg
Fon +49 (0)40 881 73 – 0 ▪ info@akquinet.de ▪ www.akquinet.de