Vorlesung
Software aus Komponenten

3. Komponenten-Modelle

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2007/08

OSGi - Plattform

- **Hardwareunabhängige dynamische SW-Plattform**, auf der Anwendungen und Dienste kombiniert und ausgeführt werden können
  - spezifiziert Java-basierte Laufzeitumgebung oberhalb der JVM und deren Basisdiensten
  - speicherplatzfreundliche Einbettung verschiedener Dienste in dieselbe JVM
- Entstand aus Komponentenansätzen im Bereich eingebetteter Anwendungen
  - Ziel: herstellerunabhängige generische SW-Plattform zur Steuerung und Vernetzung von Geräten aller Art
  - Anwendungen in den Bereichen Automotive, Handy, Gebäude-Management, Fernsteuerung von Hausgeräten
- Grundprinzip des **Gateways**: Plattform wird nicht auf einzelnen Geräten ausgerollt, sondern die Infrastrukturdienste werden im Zusammenspiel der Geräte, als wirklich verteiltes System, erbracht.

- Service-Anwendungen als **Bündel**
  - können dynamisch zur Laufzeit eingespielt, aktualisiert und auch wieder entfernt werden, ohne die JVM (bzw. das Grundgerät) anzuhalten.
  - können über Fernwartung administriert werden
  - Abhängigkeiten werden automatisch aufgelöst
  - intelligentes Versionsmanagement
- ermöglicht nachträgliche Auslieferung und Installation von Diensten sowie Verteilung von Informationen zur Laufzeit auf verschiedenen Endgeräten
- Heute auch als **Applikationscontainer im Enterprise-Bereich** verfügbar
  - Eclipse – Equinox-Projekt als prominenteste Entwicklung
    - Seit Eclipse 3 Basis für dessen Plugin-Konzept
  - ermöglicht fein granulare Komposition von Anwendungen
  - schlank und einfach gegenüber dem etablierten J2EE-Standard

Konzept

- Konzept ermöglicht **Einbeziehung der Clients in Dienstekonzept**
  - Fernwartung von Clients
  - Rich Client Konzepte
- Aufnahme und Standardisierung im Rahmen des Java Community Prozesses als JSR 921 – *Dynamic Component Support for Java SE* – als offizielles dynamisches Komponentenmodell für Java

- **OSGi Allianz** (früher: Open Service Gateway Initiative) – Industriekonsortium zur Koordinierung der Weiterentwicklung dieser Spezifikation
  - Derzeit über 30 Firmen aus sehr unterschiedlichen Bereichen als Vollmitglieder
    - u.a. Sun, IBM, Nokia, Motorola, Oracle, NEC, Hitachi, Red Hat, Samsung, Siemens, Telefonica, BEA, Dt. Telekom
  - spezifiziert nur APIs und Testfälle
  - Referenzimplementierung nur als „proof of concept“, nicht für den Produktiveinsatz

- Dokumente verfügbar unter der OSGi Specification License
  - enthält eine Nicht-Patent-Klausel
- Community Prozess offen nur für Mitglieder

- Leistungsimplementierungen, meist in Teilbereichen mit speziellem Anwendungsfokus, von dritter Seite
  - bestehen meist aus dem Framework und einer größeren Anzahl von vorgefertigten Service-Bündeln (Basisdienste), die selbst wieder dynamisch installiert werden können
  - kommerzielle Framework-Implementierungen
  - Open Source Framework-Implementierungen
    - Equinox (von Eclipse getrieben)
    - Oscar – Apache Felix (Community-Projekt der Apache Foundation)
    - Concierge – für mobile und eingebettete Systeme
- Im Enterprise-Bereich besonders im Zusammenspiel mit Spring eingesetzt
- Durch Verbreitung von OSGi-basierter Middleware in verschiedenen Anwendungsbereichen entsteht ein Markt für OSGi-Komponenten

Geschichte

- März 1999: Gründung der Allianz
- Mai 2000: R 1 mit Fokus auf Vernetzung von Haushaltsanwendungen
- Oktober 2001: R 2 mit Verbesserungen im Bereich Sicherheit und Fernwartung
- März 2003: Einsatz in den Bereichen Automotion und Unterhaltung
- Juni 2004: R 3 und Einsatz in Eclipse
- Okt. 2005 bis Sept. 2006 : R 4 mit den Teilen Kern-Spezifikation, Kern-Erweiterungen, Mobil-Spezifikation
- Aktuelle Version 4.1 (Mai 2007)

Leistungsmerkmale

- **Standardisiertes nicht-proprietäres SW-Komponenten Framework** für Hersteller, Diensteanbieter und Entwickler
  - Ausprägung als offener Standard führt zu fairer Balance zwischen den einzelnen Akteursgruppen
  - Zwei zueinander orthogonale : Dienste-Modell (Services) und Komponenten-Modell (Bündel)
- SW-Modell für **Koexistenz mehrerer Java-basierter Komponenten in einer einzigen JVM** mit besonderem Augenmerk auf Sicherheitsaspekte
  - minimiert die Speicheranforderungen, verbessert die Performanz und die Geschwindigkeit der Kommunikation
  - detailliertes Sicherheitskonzept erlaubt Abschirmung der Komponenten gegeneinander und gegen den Kontext (Sandkastenkonzept)
  - Adjustierung der Rechte erlaubt aber auch breite Zusammenarbeit verschiedener Komponenten

- **Verwaltungsschnittstelle für Software-Komponenten**
  - Deployment als Funktion der Komponente selbst
  - Standardisiertes Deployment-Format als Bündel
    - Zusammenfassung von Java-Paketen und Ressourcen
  - Deployment erfolgt nicht passiv in die Service-Plattform, sondern aktiv durch die Plattform selbst
    - Komponente wird „assimiliert“
  - Ermöglicht so „heißes“ Laden, Starten, Stoppen, Aktualisieren, Herausnehmen
  - Verschiedene Versionen derselben Komponente können in einer VM koexistieren
  - Deklaration von Export- und Importrelationen werden auf der Ebene von Java-Paketen über Namensäquivalenz von der Service-Plattform aufgelöst
  - Laufzeitkonfiguration wird beim Runter-/Hochfahren persistent gespeichert

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **Sichere Ausführungsumgebung**
  - Mehrebenen-Sicherheitsmodell, das Java-Konzepte mit OSGi-eigenen koppelt
  - Java Run Time: Pointer-Sicherheit, Puffer-Überläufe
  - Java Sprache: Modifier legen Sichtbarkeiten fest
    - damit können Bündel gegeneinander abgeschirmt werden
    - weiterer Modifier für Zugriff innerhalb eines Bündels
  - administrative Rechte auf Bündelebene (minimal bundle content exposure)
    - Zugriff auf Ressourcen kann begrenzt werden
    - Alle Objekte auf dem Aufrufstack müssen über das Recht verfügen

- **Sichere Ausführungsumgebung (cont.)**
  - Rechte, einzelne Java-Pakete aus einem Bündel zu importieren oder zu exportieren (managed communication links)
    - Interaktion zwischen Bündeln ist nur bei entsprechenden Rechten möglich
  - Paket-Sharing als mögliche Angriffsquelle
    - Möglichkeit der Definition von vertrauenswürdigen Bündeln (package permission)
  - Rechte zu Publikation, Suche oder Nutzung dezidierter Dienste (service permission)
- Diese Rechteverwaltung wird vom Plattform-Betreiber zur Verfügung gestellt und kann so an verschiedene Sicherheitsszenarien angepasst werden.

- **Kooperatives Modell des gegenseitigen Findens und Nutzens der Dienste von Komponenten** innerhalb derselben OSGi-Plattform
  - Dienst = Java-Objekt, welches ein Bündel einem anderen zur Verfügung stellt
  - damit sind sehr schlanke Komponenten möglich
  - wichtig besonders im Bereich eingebetteter und mobiler Anwendungen mit seinen Ressourcenbeschränkungen
  - Vermeidung von Redundanz auf der Ebene von Basisdiensten, die sonst mehrfach in Komponenten vorhanden sind, wenn sie nicht von der Plattform zur Verfügung gestellt werden (class sharing)
  - besonderer Schwerpunkt liegt auf aussagefähigen Testfall-Sammlungen, mit denen die dynamische Zusammenarbeit verschiedener Bündel getestet werden kann.

- **Flexible Fernwartungs-Architektur:** Erlaubt Verwaltung mehrerer tausend Service Plattformen von einem einzigen Management-Bereich aus
  - genaues Lebenszyklusmodell erlaubt sehr feingranulare Verwaltung der Komponenten über enge oder weite Policies
  - Definition einer **Fernwartungs-API**: Wartungsaufgaben können auf einer höheren Abstraktionsebene formuliert werden
  - Bindung an verschiedene existierende Protokolle
    - erlaubt die Ausführung der Wartungsaufgaben mit einem für die Geräte optimalen Wartungsregime unter Berücksichtigung von Gerätespezifika
    - variable Verfügbarkeit und geringe Bandbreite im Mobilbereich
    - permanente Verfügbarkeit und hohe Bandbreite bei Hausdienstleistungen
  - Conditional Permission Admin Service erlaubt zusammen mit der Unterstützung digitaler Signaturen dynamische Wartungsregeln, mit der größere Farmen von Softwareinstallationen einfach gewartet werden können

- **COTS – Kommerzielle Komponenten „aus dem Regal“**
  - OSGi als Basis für einen sich rasch entwickelnden Komponentenmarkt
    - So gut wie alle allgemein gebräuchlichen Protokolle, die es gibt, liegen heute bereits als OSGi-Bündel vor.
  - Fein granulares Konzept erlaubt die Entwicklung von Bündeln mit hoher funktionaler Bindung – ein Bündel, eine Funktion
  - Trennung von Spezifikation und Implementierung erlaubt, verschiedene Implementierungen mit verschiedenen Leistungsparametern und für verschiedene Geräteplattformen von verschiedenen Anbietern durch einen Betreiber plattform- und anwendungsspezifisch zusammenzustellen.
    - OSGi Service Plattform – wenn einmal aufgespielt – standardisiert die Verwaltung dieser Komponenten
  - Zertifizierung von Plattformen und Bündeln ist (nur) für Mitglieder der Allianz möglich
    - Zusicherung der plattformunabhängigen Lauffähigkeit von Bündeln

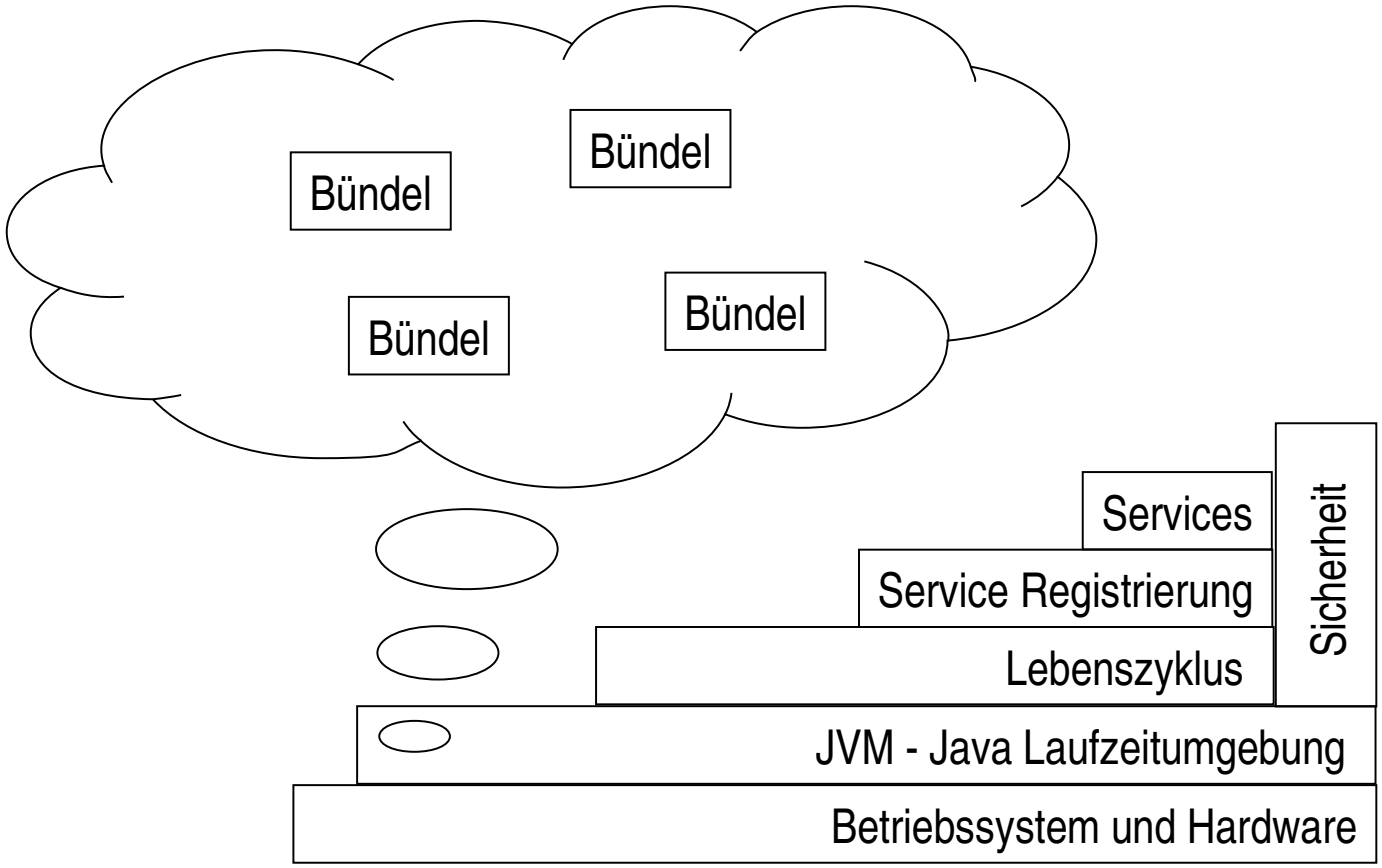
- Standardisierte optionale Dienste und Werkzeuge als generische Lösungen durch Verwendung vorgefertigter Komponenten als **Basisdienste**
  - Logging, Konfiguration, HTTP, XML, Netzwerk, IO, Ereignisbehandlung und -verwaltung, Geräte-Erkennung und Laden von Treibern (PnP), Nutzer-Authentifizierung und Autorisierung, usw.

Anwendung im Middleware-Bereich

- Interkonnektivität innerhalb einer Schicht zusammen mit Anbindung des Kontexts an darunter liegende SW-Schichten bestimmen die Leistungsfähigkeit einer Komponenten-Architektur im Middleware-Bereich
  - Besondere Bedeutung des Zusammenspiels von OSGi (Interkonnektivität) und Spring (Kontextanbindung) als zueinander orthogonale Architekturkonzepte
- Ende 2006 nahm innerhalb der OSGi-Allianz eine **Enterprise Experts Group** die Arbeit auf

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OSGi Service Plattform Architektur - Überblick



Aufgaben der OSGi-Plattform

- Installation: **installBundle**-Methode der Plattform
  - Das Bündel wird für den Einsatz in der Umgebung vorbereitet
  - Service Plattform registriert die neuen Import- und Exportrelationen (auf der Ebene von Java-Paketen)
  - Erzeugt ein **Bundle**-Objekt und initialisiert den **BundleContext** innerhalb des Bündels
- Vor dem Start müssen Abhängigkeiten aufgelöst werden
  - Bündel heißt **aufgelöst**, wenn alle Importrelationen durch Exporte anderer aufgelöster Bündel bedient werden
  - Kann Auflösung mehrerer anderer Bündel anstoßen
  - Zirkuläre Abhängigkeiten sind möglich.
- Deinstallation erfolgt virtuell – Pakete des Bündels sind so lange verfügbar, so lange aufgelöste Bündel mit diesen Abhängigkeiten existieren.
  - Neuauflösung von Abhängigkeiten über refresh des Managers sind möglich .

- Start/Stop: Aufgelöste Bündel können gestartet und angehalten werden. Dabei werden die erforderlichen Ressourcen faktisch zugeordnet bzw. entzogen.
  - Dazu erzeugt das Bündel ein **Aktivator-Objekt**, welches den Lebenszyklus über die Methoden **start** und **stop** kontrolliert.
  - Beim Start wird der Kontext an das Bündel gebunden.
  - Aktive Bündel können über den Kontext **Dienste** veröffentlichen, suchen und binden. Dienste sind einfache Java-Objekte.
  - Dienste können über ihre Schnittstelle sowie eine Metadaten-Tabelle **registriert** werden – **Service Registry**
  - Zustand wird persistent gespeichert und bei Stop/Start der Plattform aktive Komponenten automatisch hochgefahren
    - Möglichkeit der Steuerung über **start levels**

- Realisierung über den **Management-Agent** – ein normales Bündel mit Administrationsprivilegien
  - Dienst =Verwaltung der Konfiguration der Laufzeitumgebung, in der er selber läuft
  - Komplexität dieses Agenten bestimmt die Leistung der aktuellen Plattform
  - Verantwortlich für die Konsistenz der Laufzeitkonfiguration
- Aufgaben:
  - Verwaltung der Bündelauflösungen
  - persistente Verwaltung der aktiven Konfiguration
  - Verwaltung der Rechte in Bezug auf einzelne Bündel
    - Erfolgt über **Lokalisierungsinformation** (=abstrakte Identität), welche vom Manager für das Bündel bei dessen Installation vergeben wird
  - Sicherung der **Namensäquivalenz**

Namensäquivalenz und Bündelabhängigkeiten

Bündelabhängigkeiten auf der Ebene von Java-Paketen

- unterscheide Spezifikations- und Implementationspakete
- **Spezifikationspaket**: Name und Version definieren die Schnittstelle, die für alle Provider verbindlich ist.
  - Provider müssen die voll Schnittstelle implementieren
  - spätere Versionen dürfen frühere nur erweitern
    - Problem der semantischen Konsistenz bleibt trotzdem!
- **Implementationspaket**: Nichts davon gilt.
  - Inhalts- und Versionsnummern sind inkompatibel zwischen verschiedenen Providern.
- Beispiel: Paket importiert eine Spezifikationsschnittstelle und exportiert eine Implementierung dieser Schnittstelle.
- Es ist Aufgabe des Management-Bündels, diese Referenzen korrekt aufzulösen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Service-Registrierung

- Einheitliche Schnittstelle, um Änderungen der Konfiguration an alle registrierten Service-Objekte zu propagieren
  - diese müssen insbesondere Event-Kanäle zu der geänderten Komponente aktualisieren.
  - in-memory-Register, um Zugriffszeiten zu minimieren.
  - „Kleber, der die Plattform zusammenhält“
- Registrierte Objekte heißen **Dienste**. Zugeordnet ist immer ein Schnittstellenname und eine Liste von Eigenschaften.
  - Deklarative Dienste – Bündelklassen werden erst geladen und Objekt erzeugt, wenn der Dienst in Anspruch genommen wird.
- Funktionalität:
  - Objekte einzelner Bündel registrieren
  - Service-Register nach passenden Objekten durchsuchen
  - Nachrichten über Registrierung und Deregistrierung von Objekten verschicken

Laufzeitumgebung

- Laufzeitumgebung basiert auf Einzel-JVM-Umgebung, wobei sogar J2ME Profile zum Einsatz kommen können
- Plattform verwaltet physische Bündelinhalte
- Auffinden über die Lokalisierungsinformation des Bündels, welche das Management-Bündel plattformspezifisch vergeben hat (Namens-Dienst)
- Bündelinhalt wird von dieser Quelle bezogen
  - URL: Bündelinhalt wird über einen Inputstream bezogen
  - lokale Klasse: Bündelinhalt wird über Classloader bezogen.
  - Plattform nutzt für jedes Bündel einen eigenen Classloader
- Strenge Trennung zwischen Schnittstelle und Implementierung, um Abhängigkeiten sauber auf Spezifikationsebene verwalten zu können.

Basisdienste

- Basisdienste wurden im Rahmen von OSGi 4 (Core Specification and Service Compendium) standardisiert
  - eine Spezifikation – mehrere Implementierungen
- Verfügbarkeit optional, abhängig vom Anwendungsgebiet
- Framework Services – Dienste zur Verwaltung der Plattform selbst
  - Permission Admin, Conditional Permission Admin, Package Admin, Start Level, URL Handler
- System Services – horizontale Dienste, die von vielen Bündels benötigt werden
  - Log Service, Configuration Admin, Event Admin, Device Access, User Admin, Deployment Admin, Monitoring usw.
- Protocol Services – Abbildung eines externen Protokolls auf einen Service
  - Http Service: dynamischer Servlet-Starter
  - UPnP Service: neuer Standard in der Unterhaltungselektronik, wird auf die Service-Registrierung abgebildet

- Miscellaneous Services
  - Wire Admin: Zusammenbinden dezidierter Dienste entsprechend einem vorgegebenen Abo-Schema statt allgemeiner Kompatibilität
  - XML Parser
- Programming Services – Unterstützung zur Behandlung der Komplexität des Programmiermodells
  - Service Tracker – verfolgt Änderungen an den Diensten
  - Declarative Services – Definition einer Service Component Runtime, die Servicedefinitionen aus Bündeln liest und diese an der Stelle des Bündels selbst registriert.
- Weitere Basisdienste in Entwicklung
  - Remote Management – Abstraktionsschicht oberhalb der verschiedenen Fernwartungsprotokolle
  - Web Services – Implementierung der Basisfunktionalitäten
  - Application Services – Einbettung in das J2EE-Serverkonzept
  - Connectivity – besonders im Zusammenhang mit mobilen Endgeräten
  - Distribution – Erweiterung auf verteilte JVM‘s

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorteile der Plattform

- Deutliche Senkung der Entwicklungskosten und Einarbeitungszeiten möglich.
- Flexiblere Möglichkeit, um Geräte für verschiedene Märkte anzupassen
- Deutliche Verringerung der Probleme im Deployment Prozess
- Fernwartungsmöglichkeiten
- Einheitliche Behandlung eingebetteter Geräte und deren Einbettung in die Umgebung