

Vorlesung Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

Weitere Features

- Verbindung von managed und unmanaged code über Interop-Technik
 - Einbindung traditioneller COM-Programme über .NET-Kapsel an der CIL vorbei möglich
 - unmanaged code oft an performanzkritischen Stellen
 - Bedeutung relativiert durch die Erfahrung, dass es oft effizienter ist, die Algorithmen zu verbessern statt die Implementierung
- Komplexes Sicherheitskonzept
 - Authentizität des Herstellers
 - Schutz der Programme vor Veränderung
 - codebasiertes und nutzerbasiertes Sicherheitsmodell zur Beschreibung der Vertrauenswürdigkeit des Codes

Weitere Features

- Attributbasiertes Programmieren
 - Dependency Injection zur Übernahme von Diensten der Umgebung (Assembly) in das Programm
 - wurde von Java als Annotationen z.B. in EJB 3.0 übernommen
- Basisklassen-Bibliothek deckt alle wichtigen Grundfunktionalitäten ab
 - z.B. Textformatierung, Email-Versand, Codegenerierung
 - Basisklassen zur Anbindung an den Web Service Standard als Grundlage für verteilte Anwendungen
- Integration der Laufzeitumgebung in Windows Server seit 2003
 - explizit in Konkurrenz zu J2EE
 - Kernbestandteil von Windows Vista
 - abgespeckte Version der Laufzeitumgebung für Kompaktgeräte

Weitere Features

- Bindung verschiedener Programmiersprachen an dieselbe Plattform erlaubt Erstellung einer Applikationen unter Verwendung verschiedener Hochsprachen
 - wird durch sprachunabhängige Editoren wie Eclipse oder Visual Studio .NET
 - Gemeinsame Verwendung von Bibliotheken
 - Wartbarkeit derartiger Projekte ist deutlich komplexer als solcher, die nur in einer Hochsprache geschrieben sind.
 - Ändert sich mglw. mit dem Einsatz generativer Techniken und modellgetriebener Software-Entwicklungsansätze
- Hoher Anklang der Plattform vor allem in marktengen und akademischen Bereichen, wo Programmiermöglichkeiten voll ausgereizt werden.

Bestandteile des Frameworks

- Common Intermediate Language / Common Language Runtime
 - Zwischensprachencode wird in Maschinencode übersetzt und ausgeführt
 - kleinste Schnittmenge für Sprachen, die von CLR unterstützt werden
- Base Class Library
 - Bibliotheken mit wichtigen Basisfunktionen werden erst auf der Ebene der CIL eingebunden und sind so hochsprachenunabhängig verfügbar
- Common Language Specification / Common Type System
 - Vereinigung der Sprachkonzepte, die im .NET-Framework unterstützt werden, um Datenkompatibilität auf Hochsprachenebene zu sichern
- Assemblies
 - Komponentenkonzept zum Kapseln von Software-Bausteinen
 - mit Metadaten, Integritätscheck, Versionierung
 - Konzept des Managements von Abhängigkeiten zu anderen Assemblies
 - Global einheitliches Namenssystem (GAC)
 - werkzeuggestütztes Verwaltungssystem für Assemblies
 - Administration über Konfigurationsdateien (XML)

Vorlesung Software aus Komponenten

4. Moderne Komponentenkonzepte

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

Allgemeine Konzepte

- Genauere Trennung und Spezifikation von
 - Entwicklungsmodell
 - Interaktionsmodell
 - Plattformmodell
 - Packaging and Deployment
- Fokus auf serviceorientierte Architekturen
 - stärkere Trennung von **Geschäftslogik** innerhalb der Komponenten und **Interaktionslogik** im Framework
- Fokus auf eng gekoppeltes Zusammenspiel der Interaktionslogik auf einer gemeinsamen BS-Basis (Applikationsserver, Client-PC)
 - Spring-Framework
 - OSGi – Open Service Gateway Initiative
 - CCM – Corba Component Model
- Im Vergleich zu J2EE leicht-gewichtigere Container und Entwicklungsstrukturen (Lean Software Movement)

Separation of Concerns

- **Concerns** = Spezifische Anforderung oder Gesichtspunkte, die in einem Software-System behandelt werden müssen, um die übergreifenden Systemziele zu erreichen (Laddad, 2003).
- Problem: Mehrdimensionalität dieser Anforderungen lässt sich in modularem Entwurfsparadigma nur bedingt abbilden
- Trennung bereits auf der Ebene der Anforderungserhebung in
 - **Fachanforderungen** (*core concerns*), die als Komponenten ausgebildet und früh in die finale Applikation integriert werden
 - **Querschnittsanforderungen** (Logging, Sicherheit, Transaktionskonzepte, *cross cutting concerns*), deren Integration möglichst lange hinausgezögert wird
- In modernen Komponentenmodellen werden Querschnittsanforderungen der Fachapplikation als Basisdienste aus der Plattform über einen klar ausgeprägten **Kontext** (Laufzeitumgebung) injiziert.
 - Ansätze: Dependency Injection, Inversion of Control
 - Verallgemeinerung: Aspektorientierte Programmierung

Aspekte

Dependency Injection

- Klassisch: Objekt ist selbst zuständig, seine Abhängigkeiten (Zuordnung von benötigten Objekten und Ressourcen) zu erzeugen und zu verwalten.
- EJB 2: Steuerung liegt ausschließlich beim Framework, Anforderungen werden über Deployment-Deskriptoren spezifiziert
- Neu: Objekte werden vom Framework (im Kontext) erzeugt und verwaltet und von dort auf der Ebene der Anwendung (POJO) verfügbar gemacht – Verallgemeinerung des Factory-Pattern
 - Dezentralisierung durch Annotationen – Verschiebung von der Werkzeug- auf die Sprachebene

Inversion of Control

- Anwendung (POJO) verhält sich gegenüber dem Kontext wie ein Client gegenüber dem Server

Aspekte

Aspektorientierte Programmierung

- Kontext wird nicht nur als Datenhintergrund und Lebenszyklus-Verwaltung verstanden, sondern als Laufzeitumgebung, in der die Fachlogik des Anwendungscodes abläuft
- Definition von Joinpoints im Anwendungscode, an denen Aspekte eingeflochten werden können
 - Joinpoints (Compilezeit) und Joinpoint shadows (Laufzeit)
- Definition von Regeln, nach denen an solchen Joinpoints wirkliche Pointcuts ansetzen
- Advices – Art der möglichen Eingriffe: before, after, around
- Vorteil der Aspektorientierung ist die logische und physische Trennung der Semantik von Querschnittsanforderungen in den Komponenten von deren fachlicher Realisierung (Aspekt)
- Entwicklung eines Sprachstandards für derartige Funktionalität.
- Steckt noch in den Kinderschuhen – heute meist nur in Form expliziter Sprachkonstrukte wie in EJB 3 verfügbar.

Das Spring Framework

- Leichtgewichtiges Open Source Applikationsframework für die Java-Plattform, um Entwicklungen innerhalb der J2EE zu vereinfachen
 - Alternative zu den Sun J2EE Blueprints als Architekturempfehlung für J2EE-Anwendungen, dem ein strenges Schichtenkonzept (Web-Schicht, Business-Schicht auf EJB-Basis, Daten-Schicht) zu Grunde liegt.
- Verwirklicht beispielhaft das Zusammenspiel zwischen POJO's auf der Anwendungsebene und verschiedenen konditionierbaren Framework-Kontexten (Entwicklung, Testumgebung, Produktivumgebung) durch weitere Einbettung der Kontexte
- Auf der Basis existieren weitere Projekte, welche die Anbindung von Spring an verschiedene häufig anzutreffende Entwicklungsumgebungen und -anforderungen realisieren.

Ansatz und Community

- Oktober 2002: Grundlagen werden von Rod Johnson im Rahmen seiner Buchreihe „Expert One-On-One J2EE Design and Development“ entwickelt, der auch den Sourcecode des Framework-Gerüsts als freien Download zur Verfügung stellt.
- Juni 2003: erstes Release, unter der Apache 2.0 Lizenz
- März 2004: Spring 1.0 – seitdem fand das Framework sehr schnelle Verbreitung. Entwicklung wird von SpringSource koordiniert
 - <http://www.springsource.org>
- Gemeinsame Entwicklungsbemühungen um einen leichtgewichtigen Nachfolger der J2EE
 - Bibliotheken stehen unter der Apache 2.0 Lizenz frei zur Verfügung
 - *Spring Community Forums* als themenspezifische Kommunikationsplattformen
 - *SpringSource Enterprise Bundle Repository* als eine Sammlung von Basiskomponenten

Mission Statement

We believe that:

- J2EE should be easier to use
- It is best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero.
- JavaBeans offer a great way of configuring applications.
- OO design is more important than any implementation technology, such as J2EE.
- Checked exceptions are overused in Java. A platform shouldn't force you to catch exceptions you're unlikely to be able to recover from.
- Testability is essential, and a platform such as Spring should help make your code easier to test.

We aim that:

- Spring should be a pleasure to use
- Your application code should not depend on Spring APIs
- Spring should not compete with good existing solutions, but should foster integration. (For example, JDO, Toplink, and Hibernate are great O/R mapping solutions. We don't need to develop another one.)

(Quelle <http://www.springsource.org/about>)

Geschichte

- Oktober 2006: Version 2.0
- November 2007: Version 2.5
- September 2009: SpringSource wird von VMWare übernommen
- Dezember 2009: Version 3.0
 - Neue *Spring Expression Language*, mit der Ausdrücke zur Laufzeit ausgewertet werden können. So lassen sich dynamische Elemente des Systems implementieren, ohne komplexe Techniken wie eine Programmiersprache nutzen zu müssen.
 - Konfiguration mit Annotationen und Meta-Annotationen
 - REST-Unterstützung (Representational State Transfer) im MVC-Framework. Erlaubt den standardisierten Austausch von Zustandsinformationen zwischen verschiedenen Komponenten
 - *Abhängigkeitsmanagement* aus der Plattform ausgelagert, da neue Konzepte und Werkzeuge wie Maven, ivy oder OSGi existieren, die nicht Spring spezifisch sind

4.2. Spring Konzeption

- Komponenteneinsatz auch auf der Ebene der Applikationsentwicklung
 - Applikationsdienst wird im Zusammenwirken mehrerer Komponenten (diese werden **Applikationsobjekte** genannt) erbracht, die **innerhalb** der Applikation als Kontext konfiguriert werden.
 - dabei Rückkehr zu plain old Java Objekten (POJO)
- Schichtenmodell mit Standardisierung auf verschiedenen Abstraktionsebenen in eigenen Teilprojekten
 - Konfiguration und Basisarchitektur: Spring Core
 - Kommunikation und Datenaustausch: Spring Integration
 - Prozessmodellierung: Spring Web Flow
 - Benutzerschnittstellen (UI)-Entwicklung: Spring Faces, Spring Web
 - Verteilung und Performance: Spring JEE
 - Datenbankschicht: Spring ORM, Spring DAO
 - Querschnittsfunktionen: Spring AOP

- Komponenten **und** Kontext (als leichtgewichtiger Container) sind Gegenstand der Entwicklung
 - Lösung der Applikation von der inhärenten Bindung an einen J2EE-Server
 - Komponente trifft keine unnötigen Annahmen über den Kontext
 - Applikationsobjekte werden zur Laufzeit von außen konfiguriert.
 - Konfiguration der Komponente transparent für den Anwender, er hat Zugriff auf die Dienstschnittstelle, nicht aber auf die Zuordnung der Ressourcen. Entsprechende Konfigurationsmethoden sind nur auf der Implementierungsklasse bekannt.
 - Explizites Deployment wie bei EJB wird damit von vornherein vermieden.
- Nutzung von Dependency Injection (Inversion of Control) und Annotationen zur Kommunikation zwischen Komponente und Kontext

- Spring bietet speziellen Support für typische Laufzeitumgebungen an
 - Spring MVC – Framework für Webanwendungen
 - Spring Web Flow – Framework für Abläufe auf einer Web-Site
 - Spring Security (vormals Acegi) – Framework für Sicherheit
 - Spring Rich Client – Framework zur Verteilung der Dienst-erbringung auf Server und Client
- Unterstützung aspektorientierte Ansätze
 - Idee: Basisdienste (cross cutting concerns) werden vom Kontext angeboten und über entsprechende deklarative Schnittstellen (Interzeptoren) in die Komponente eingebunden.
 - Spring bietet eigene Annotationen u.a. im Bereich Transaktionen und Bindung an die Persistenzschicht.
- Fazit: Ein und dasselbe Komponentenmodell kann sowohl für grob granulare Fassadenkomponenten (etwa EJB) wie auch für fein granulare Applikationsobjekte verwendet werden.