

# **Vorlesung Software aus Komponenten**

## **3. Komponentenmodelle**

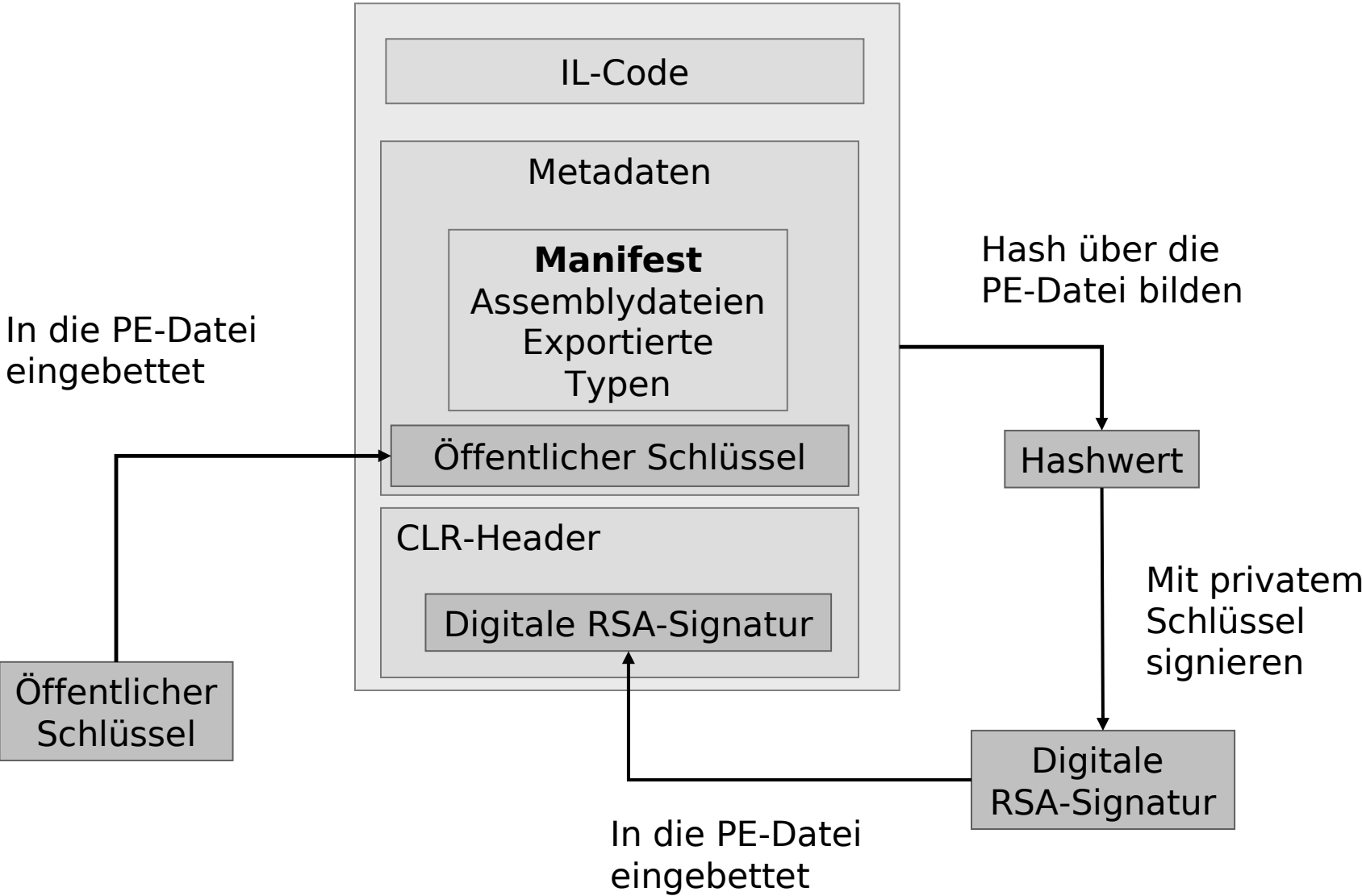
Prof. Dr. Hans-Gert Gräbe  
Wintersemester 2012/13

### Komponenten als Packungseinheiten - Assemblies

- atomare, selbstbeschreibende Einheit, inklusive Metadaten
  - Portable Executable (PE-Datei)
- Metadatenstruktur wird **Manifest** genannt
- "single file assembly" - Manifest ist Teil des eigentlichen Codes - oder
- "multi file assembly" - Manifest ist eigenständige Einheit
- ein Manifest enthält ...
  - Identität der Assembly (Name, Version, ...)
  - die Namen aller Dateien in der Assembly
  - kodierte Hashwerte aller Dateien im Assembly
  - Details der vorhandenen Klassen, Methoden und Eigenschaften
  - Namen und Hashwerte aller referenzierten Assemblies
  - Sicherheitseinstellungen

- Assemblies können *private*, *shared* oder *global* sein.
  - Private Assemblies im gemeinsamen Verzeichnis. Annahme der Versionskompatibilität
- Shared Assemblies verwenden *Strong Names* zur eindeutigen Identifizierung
  - zusammengesetzt aus Dateinamen (ohne Erweiterung), Versionsnummer, Kultur, Token für einen öffentlichen Schlüssel
  - festdefinierte Struktur:  
Name\_Versionsnummer\_Kultur\_Schlüsseltoken
- globale Assemblies werden auf dem Rechner mit dem Werkzeug *gacutil* im globalen Assembly-Zwischenspeicher (Global Assembly Cache (GAC)) gespeichert.
  - Vorteil: Assembly kann global genutzt werden
  - Nachteil: Keine einfache Installation mehr möglich
- Signierung = Möglichkeit zur Sicherstellung der Unversehrtheit des Codes

# 3.7. Das .NET-Konzept Assemblies



### Weitergabe von Assemblies

- Kopieren von Dateien in Zielverzeichnis, z.B. per Batch
  - alle abhängigen Verweise und Typen sind in Assembly enthalten
  - referenzierte Assemblies im Anwendungsverzeichnis
- Konventionelle Installation möglich (cab, msi ...)
  - Verknüpfungen auf Desktop, Startleiste ...
  - Einbindung in Softwareverwaltung von Windows
  - zukünftig eventuell Automation von Verknüpfung
- Private Assemblies
  - werden in Anwendungsverzeichnis installiert
  - werden **nur** von jeweiliger Anwendung benutzt
  - es werden nur Typen gebunden, die für die Anwendung passen

### 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 2012/13

### 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.