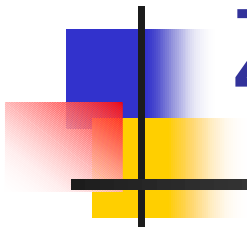
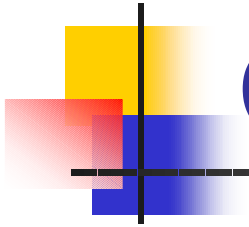


Zwischenbericht - Diplomarbeit

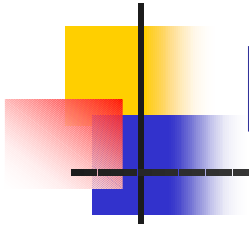
„Generierung serverseitiger Komponenten basierend auf UML Profiles“

Sean Eikenberg



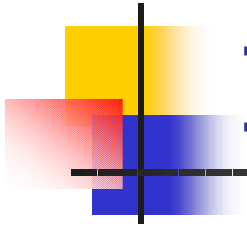
Gliederung (Aufgabenstellung)

- Anforderungen an Komponentenmodellierung (EJB)



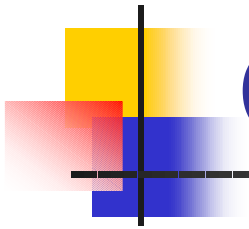
Komponentenmodellierung

- Ausgangspunkt: Modellierung komponentenbasierter Anwendungssysteme (EJB)
- Dabei Verwendung von
  - eigenen (maßgeschneiderten) Komponenten
  - vorgefertigen Komponenten („Off-The-Shelf“)
- Ziel: Anforderungen so spezifizieren, dass Generierung von EJB-Implementierungs-fragmenten basierend auf UML-Modell möglich



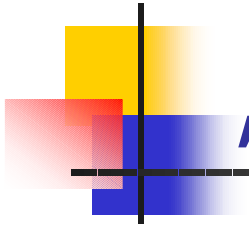
Identifizierte Anforderungen

- Modellierung d. Komponentenspezifikation:
  - Komponentenschnittstelle(n)
  - Realisierung [Implementierungsklasse(n)]
- Modellierung d. Komponentenrelationen:
  - (Vererbung)
  - Assoziationen (Aggregation, Komposition)
  - Abhängigkeiten



Gliederung (Aufgabenstellung)

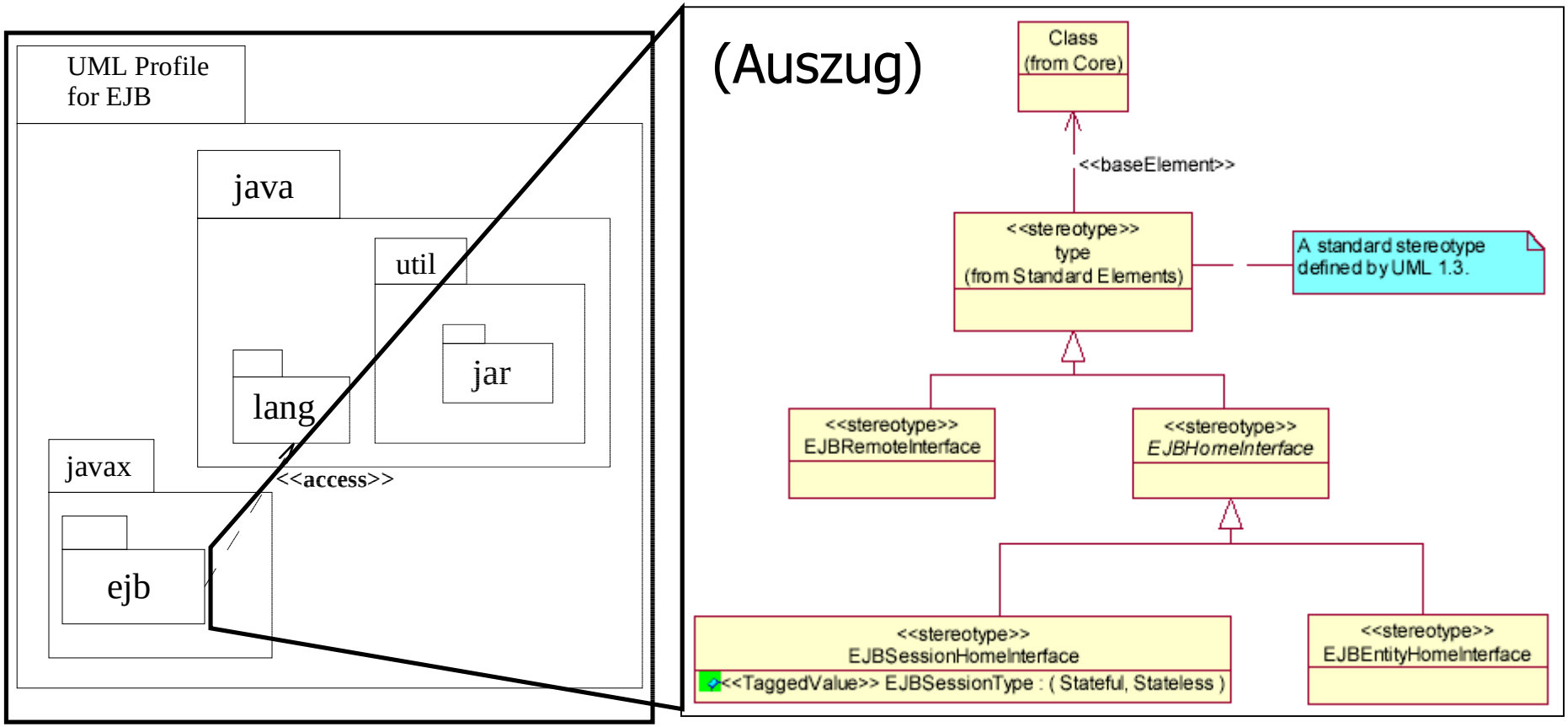
- „UML Profile for EJB“:
  - Abdeckung der Anforderungen
  - Allgemeine Optimierung / UML 1.4 – Konformität
  - Erweiterung (Relationen / EJB 2.0)

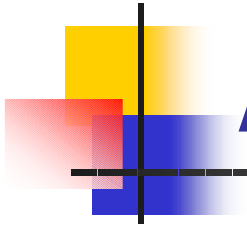


„UML Profile for EJB“

- Standardisierungsbemühung - JSR-000026 (http://www.jcp.org/jsr/detail/26.jsp)
- Ziel: Erweiterung d. UML um EJB-Semantik
- Firmenkonsortium (Rational Software Corporation)
- Basiert auf UML 1.3 / EJB 1.1
- Aktueller Stand: „Closed Public Review“

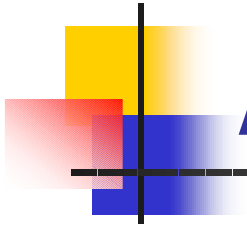
Profile-Architektur





Anforderungsabdeckung

- Profile-Konzipierung gemäß „Bottom-Up“-Ansatzes
- Modellierung der Komponentenspezifikation im Vordergrund
- Komponentenrelationen nur rudimentär: „EJB-Referenzen“

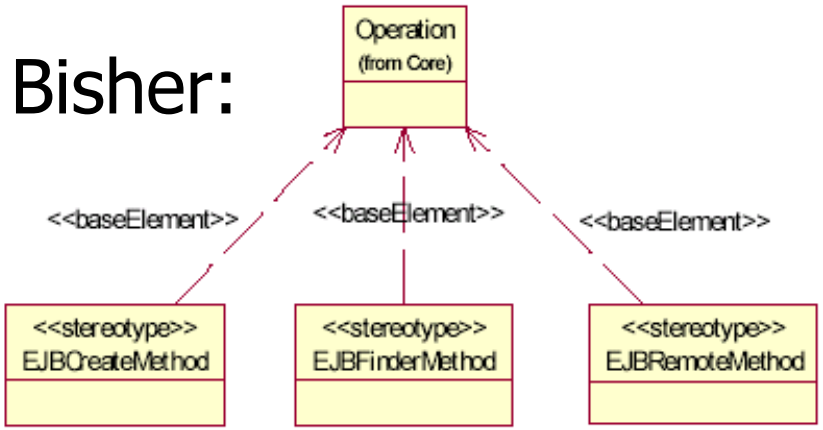


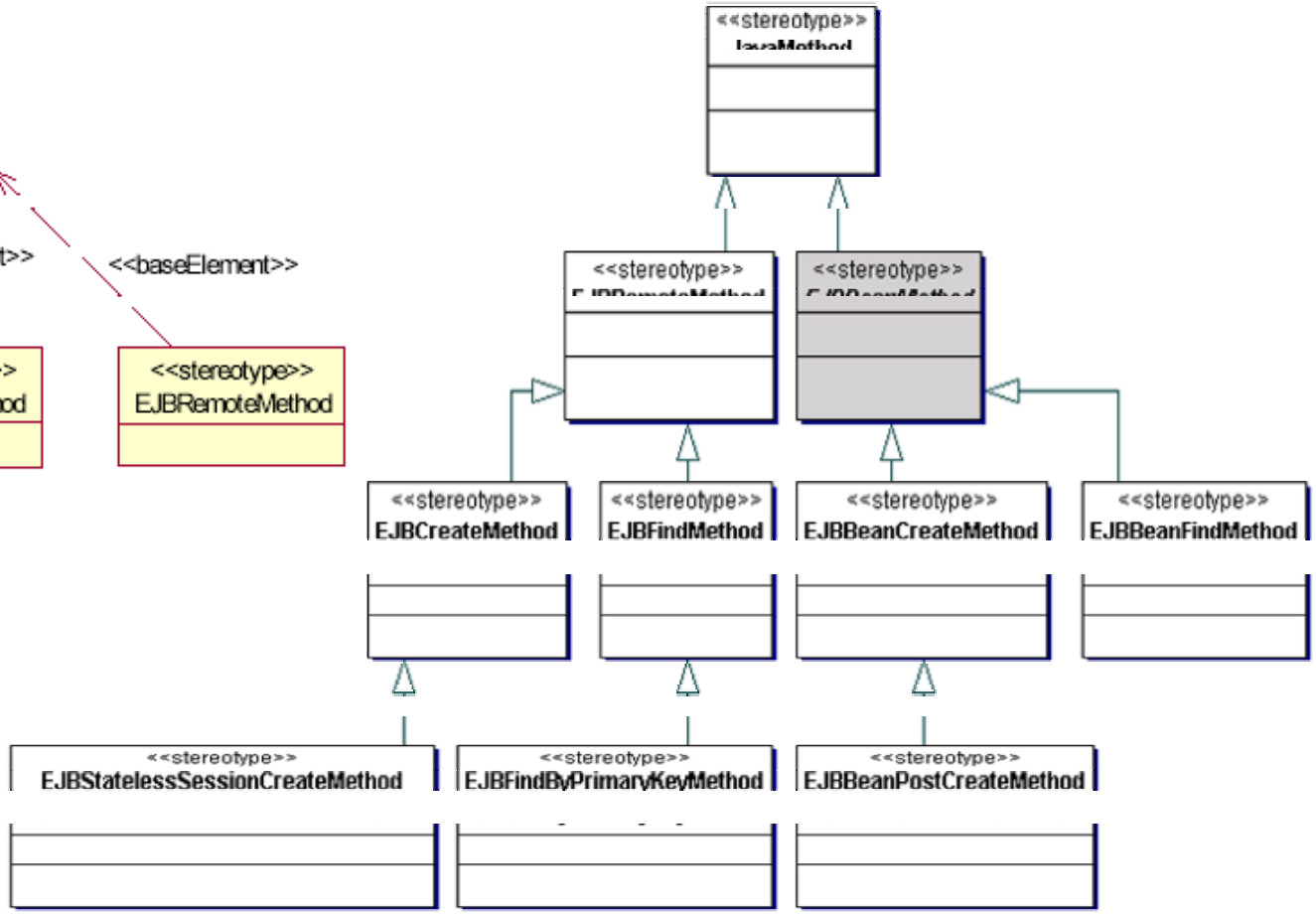
Allgemeine Optimierung & UML 1.4

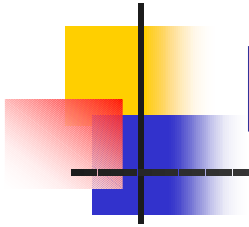
- Java-Erweiterung wird ausgeklammert!
- Syntax/Semantik-Fehler
- Redundanzminimierung
- Tag-Definitions einführen
- Constraints & Tag-Definitions an Stereotypes binden
- Veränderte Metamodellelemente berücksichtigen („Artifact“)

Optimiertes Profile (Auszug)

Bisher:

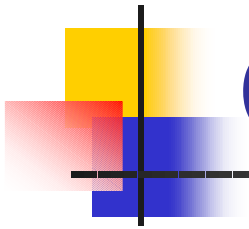






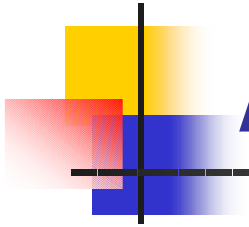
EJB 2.0 Erweiterung

- „Local Interfaces“
- „Container-Managed Persistence“
  - „Container-Managed Relationships“
  - „EJB-QL“
  - „Abstract Getter-/Setter-Methods“
- „Message-Driven Beans“
- „Select Methods“
- „Home Methods“



Gliederung (Aufgabenstellung)

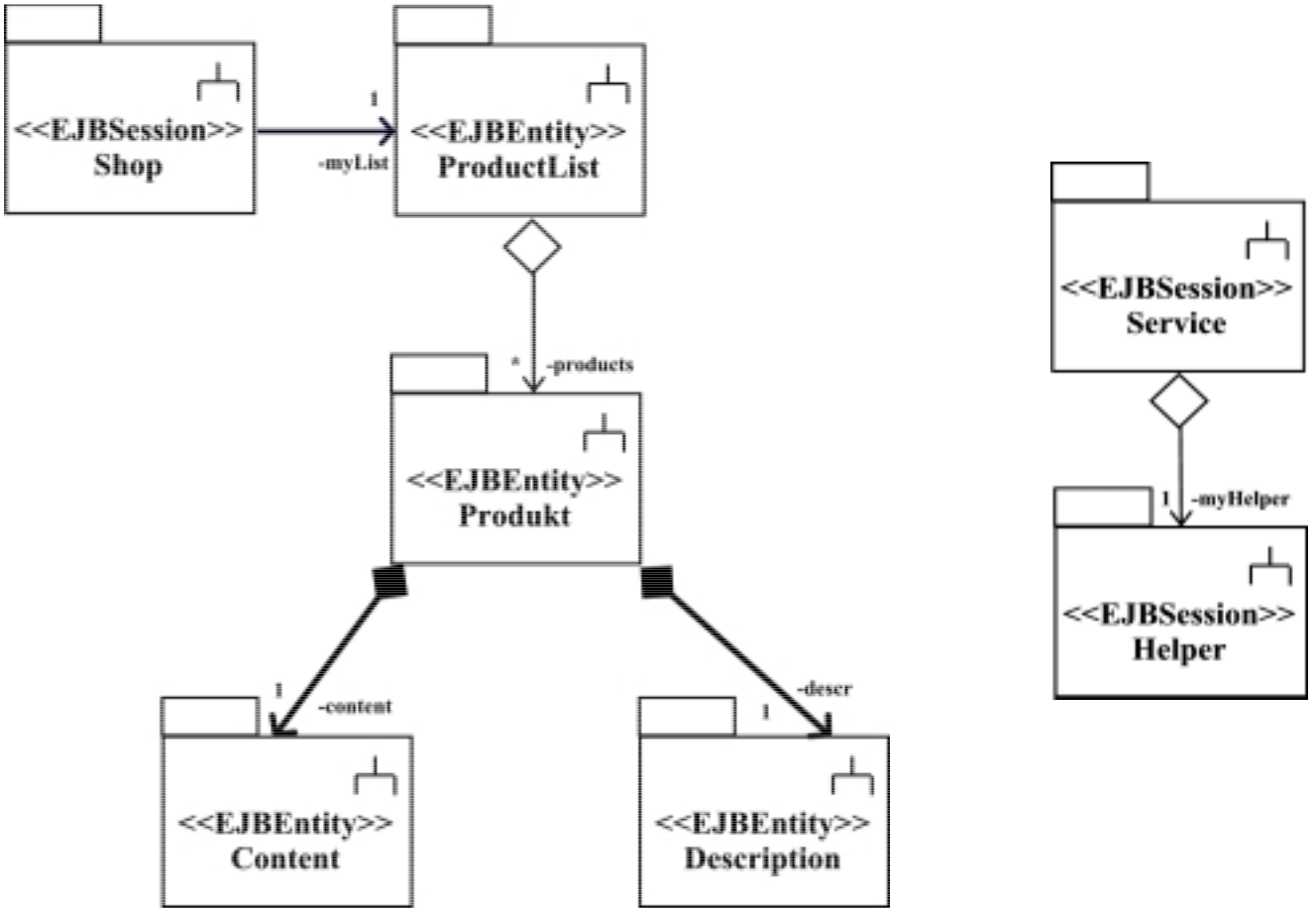
- Abbildungsvorschriften: Modell -> EJB-Komponentenimplementierung

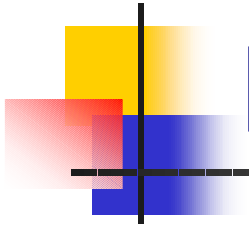


Abbildungsvorschriften

- Abbildung von UML-Modellen auf EJB-Komponentenimplementierung:
  - Component-/Home-Schnittstelle(n)
  - Implementierungsklassenrumpf
  - Deployment-Descriptor
- Unterscheidung zwischen
  - Komponentenstruktur
  - Komponentenrelationen

Komponentenrelationen





Beispielcode (1)

```
public class Service implements SessionBean {
    // ...
    private Helper myHelper;
    private void associateHelper() throws CreateException {
        // ...
        try {
            iniCtx = new InitialContext();
            o = iniCtx.lookup("java:comp/env/ejb/Helper");
            home = (HelperHome)PortableRemoteObject.narrow(o, HelperHome.class);
            o = home.create();
            this.myHelper = (Helper)PortableRemoteObject.narrow(o,Helper.class);
        } catch(java.lang.Exception e) {
            // ...
        }
    }
```

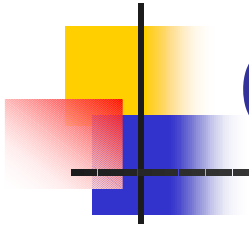
Beispielcode (2)

```
    // ...
    public void ejbCreateService() {
        this.assoziateHelper();
        // ...
    }
    // ...
} // End of ServiceBean
```

Implementierungsklasse (Forts.)

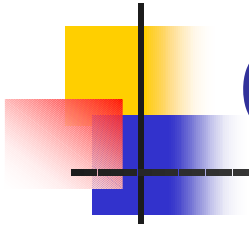
Deployment-Descriptor (Auszug)

```
<enterprise-beans>
    <session>
        <ejb-ref>
            <ejb-ref-name>ejb/Helper</ejb-ref-name>
            <ejb-ref-type>Session</ejb-ref-type>
            <home>HelperHome</home>
            <remote>Helper</remote>
        </ejb-ref>
    </session>
</enterprise-beans>
```



Gliederung (Aufgabenstellung)

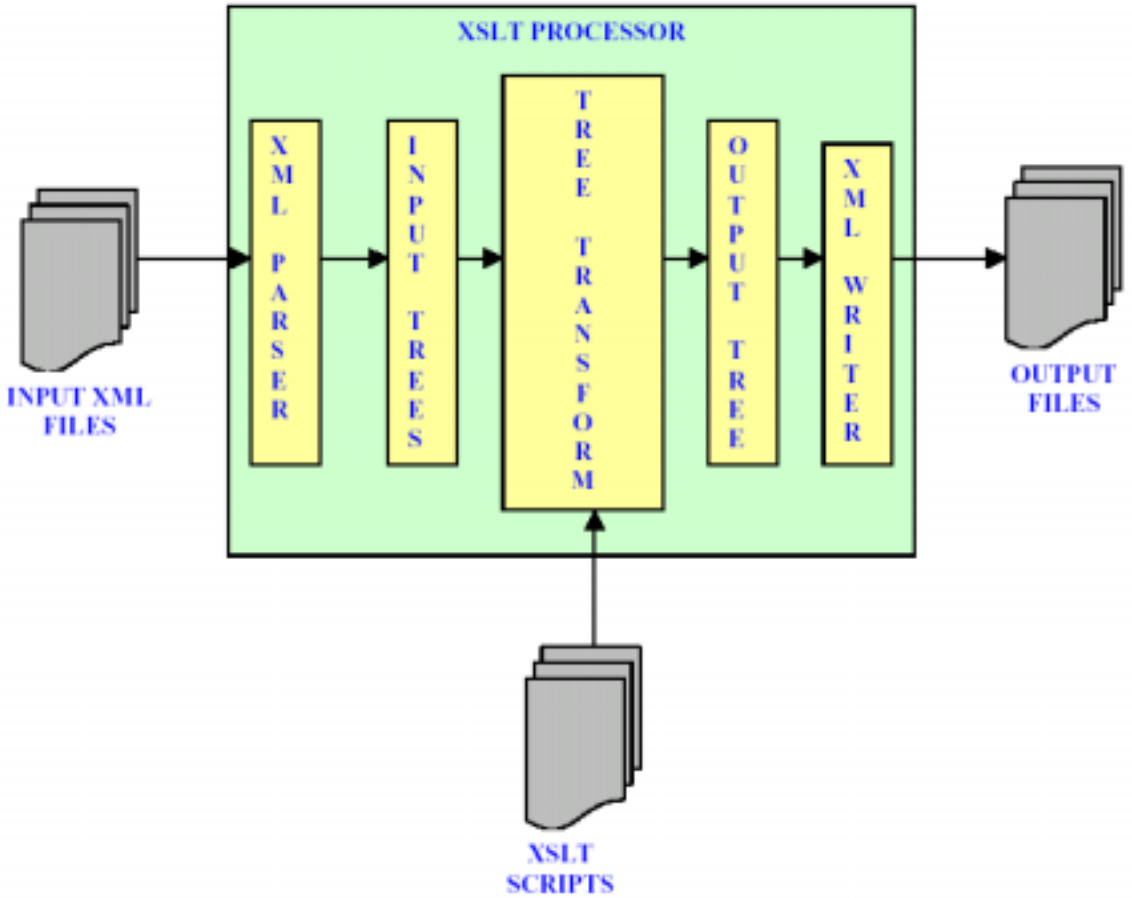
- Prototypische Umsetzung: Generator

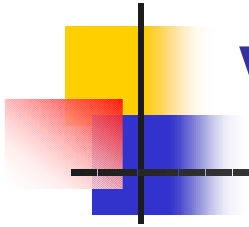


Generatorarchitektur

- Quelle: UML-Modell als XMI-Datei
- Ziel: EJB-Komponentenimplementierung (Component-/Home-Schnittstelle(n), Implementierungsklassenrumpf, Deployment-Descriptor)
- Prinzip: Code-Generator mittels XML-Dokumententransformation
- Lösung: (teilweise) Verwendung von XSLT

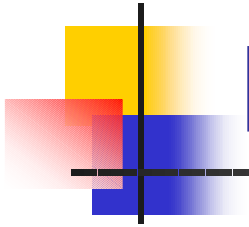
XSLT-basierte Codeerzeugung





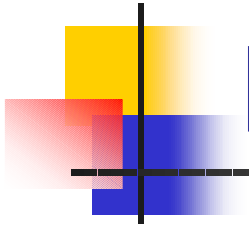
Vor- & Nachteile von XSLT

- Internationaler Standard (W3C) [+]
- Verwendung vorhandener XSLT-Prozessoren (Parser, Transformator & „Writer“) [+]
- Hohe Flexibilität, da jederzeit an veränderte Ausgangsgrammatik anpassbar (XSLT-Skript) [+]
- Einbettung in Java möglich (JAXP) [+]
- XSLT ist eine funktionale Sprache [-]
- XSLT-Prozessor ignoriert Syntax-Fehler [-]



Meilensteine

- EJB 2.0 / Relationen ergänzen
- Abbildungsvorschriften ausarbeiten
- Generatorkonzept umsetzen:
  - XSLT-Skript
  - Java-basierter Generatorentwurf
- UML-Beispielmodelle in Form von XMI-Dateien ausarbeiten und mittels Generator transformieren (Nachweis)



Diskussion (se3@inf.tu-...)

- „Session Beans“:
  - Assoziationen/Aggregationen semantisch sinnvoll?
  - Bidirektionale Assoziationen sinnvoll?
  - Multiplizitäten > 1 sinnvoll?
- UML-Metamodellelement „Subsystem“ oder „Component“ für Komponentenspezifikation?
- „Message-Driven Bean“ = Komponente?
- ...