

Zwischenbericht Großer Beleg

Enterprise JavaBeans und JavaBeans

Sean Eikenberg

Zielstellung des GB / Gliederung



- Zusammenhang zwischen EJB & JavaBeans ausarbeiten.
- Mögliche Vereinigung beider Technologien diskutieren.
- Entwurf eines Generators, der die Integration von EJB innerhalb des JavaBeans-Komponentenmodells ermöglicht.
- Nachweis der Funktionstüchtigkeit des Generators anhand einer Beispielanwendung.

1. Zusammenhang: EJB - JavaBeans

- Behauptung: EJB-Technologie ist eine Erweiterung der JavaBeans-Technologie!
- Stimmt diese Behauptung?
- Erweiterung im Sinne der Produktpalette?
- Oder abwärtskompatible Erweiterung des Komponentenmodells?
- Vergleich beider Komponentenmodelle:

1. Zusammenhang: EJB - JavaBeans

Eigenschaften	JavaBeans	EJB
Programmiersprachenmodell	JDK 1.1	JDK 1.1
Laufzeitumgebungen	Entwicklungs- & Laufzeitumgebung	Container
Aufrufbare Methoden	Konfigurations- & fachliche Methoden	Verwaltungs- & Geschäftsmethoden
Konfigurierbare Eigenschaften	✓	Zustandsattribute, Datenfelder, externe Parameter

1. Zusammenhang: EJB - JavaBeans

Eigenschaften	JavaBeans	EJB
Individuelle Anpassung	Objektserialisierung	Deployment-Deskriptor
Introspektion	Java-Reflection, BeanInfo	EJBMetaData, Deployment-D.
Dauerhafte Speicherung	Objektserialisierung	Containerbasiert, Datenbanken
Komponenten-typen	Visuelle, nicht-visuelle Komponenten	session beans, entity beans

1. Zusammenhang: EJB - JavaBeans

Eigenschaften	JavaBeans	EJB
Komponenten- logik	Fachliche Klasse, Hilfsklassen	Fachliche Klasse, Verwaltungs- & Zugriffsschnittstelle, Hilfsklassen, Deployment-D.
Visuelle Manipulation	✓	✗
Ereignis- verarbeitung	✓	✗

1. Zusammenhang: EJB - JavaBeans

Eigenschaften	JavaBeans	EJB
Beliebiger Ressourcenzugriff	✓	✗
Beliebige Ausnahmen	✓	✗
Verteilungsfähigkeit	Java RMI, Java IDL, JDBC	✓
Transaktionsunterstützung	✗	✓

1. Zusammenhang: EJB - JavaBeans

Eigenschaften	JavaBeans	EJB
Sicherheitsmanagement	✗	✓
Komponentenumgebung	✗	✓

1. Zusammenhang: EJB - JavaBeans

- Zusammenfassung:
 - JavaBeans unterstützt keine Containerdiensten (Sicherheit, Transaktionen, Verteilung).
 - EJB unterstützt nicht freien Ressourcenzugriff, Ereignisverarbeitung oder visuelle Komponenten.
 - Grundsätzlich verschiedene Komponentenlogik.
- Fazit: EJB ist keine direkte Erweiterung von JavaBeans!
- Dennoch gibt es eine große Menge von Gemeinsamkeiten...

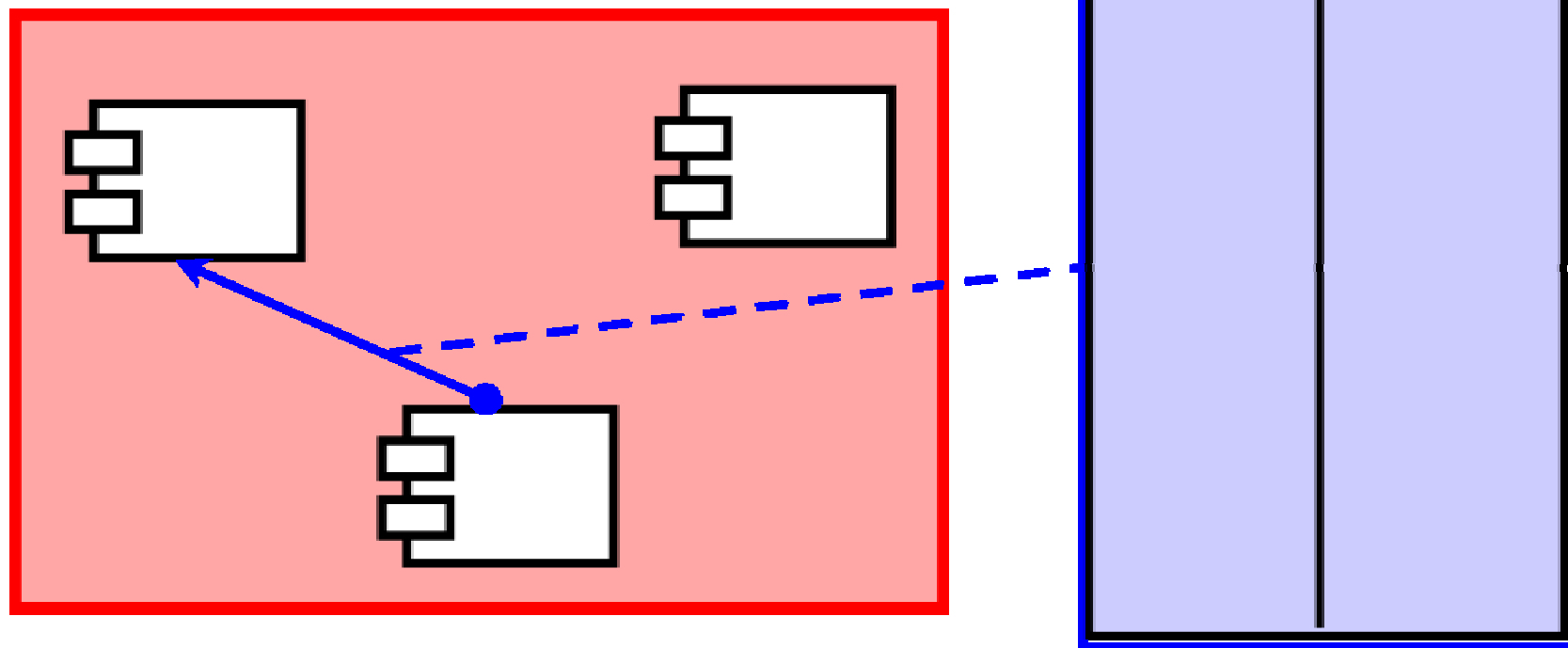
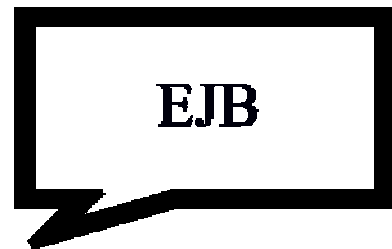
Zielstellung des GB / Gliederung

- Zusammenhang zwischen EJB & JavaBeans ausarbeiten.
- ? • Mögliche Vereinigung beider Technologien diskutieren.
- Entwurf eines Generators, der die Integration von EJB innerhalb des JavaBeans-Komponentenmodells ermöglicht.
- Nachweis der Funktionstüchtigkeit des Generators anhand einer Beispielanwendung.

2. Vereinigungsmodelle: EJB & JavaBeans

- Trotz festgestellter Unterschiede erscheint eine „Vereinigung“ beider Komponentenmodelle als sinnvoll!
- Beispiel: Unternehmen, in denen beide Technologien genutzt werden und wo durch eine „Vereinigung“ Kosten eingespart werden könnten...
- Vier denkbare Vereinigungsmodelle:

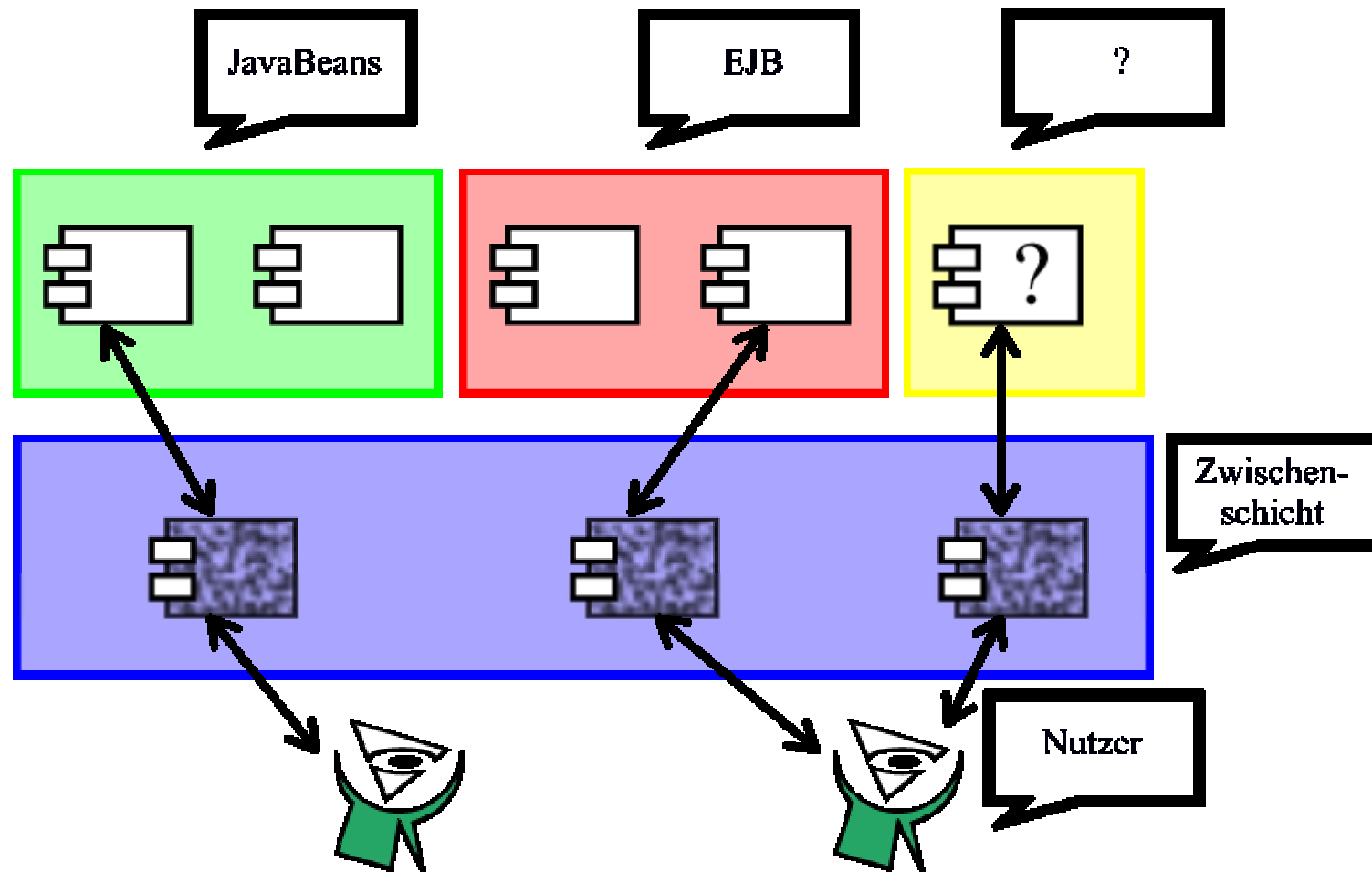
„EJB & visuelle Anwendungserstellung“



„EJB & visuelle Anwendungserstellung“

- Vorteil(e):
 - Einfache (visuelle) Erstellung und Verknüpfung von EJB-Anwendungen (Application Assembler)
- Nachteil(e):
 - Hoher Komplexitätsgrad, da Anpassung der Entwicklungsumgebungen nötig und/oder Entwurf einer Zwischenschicht

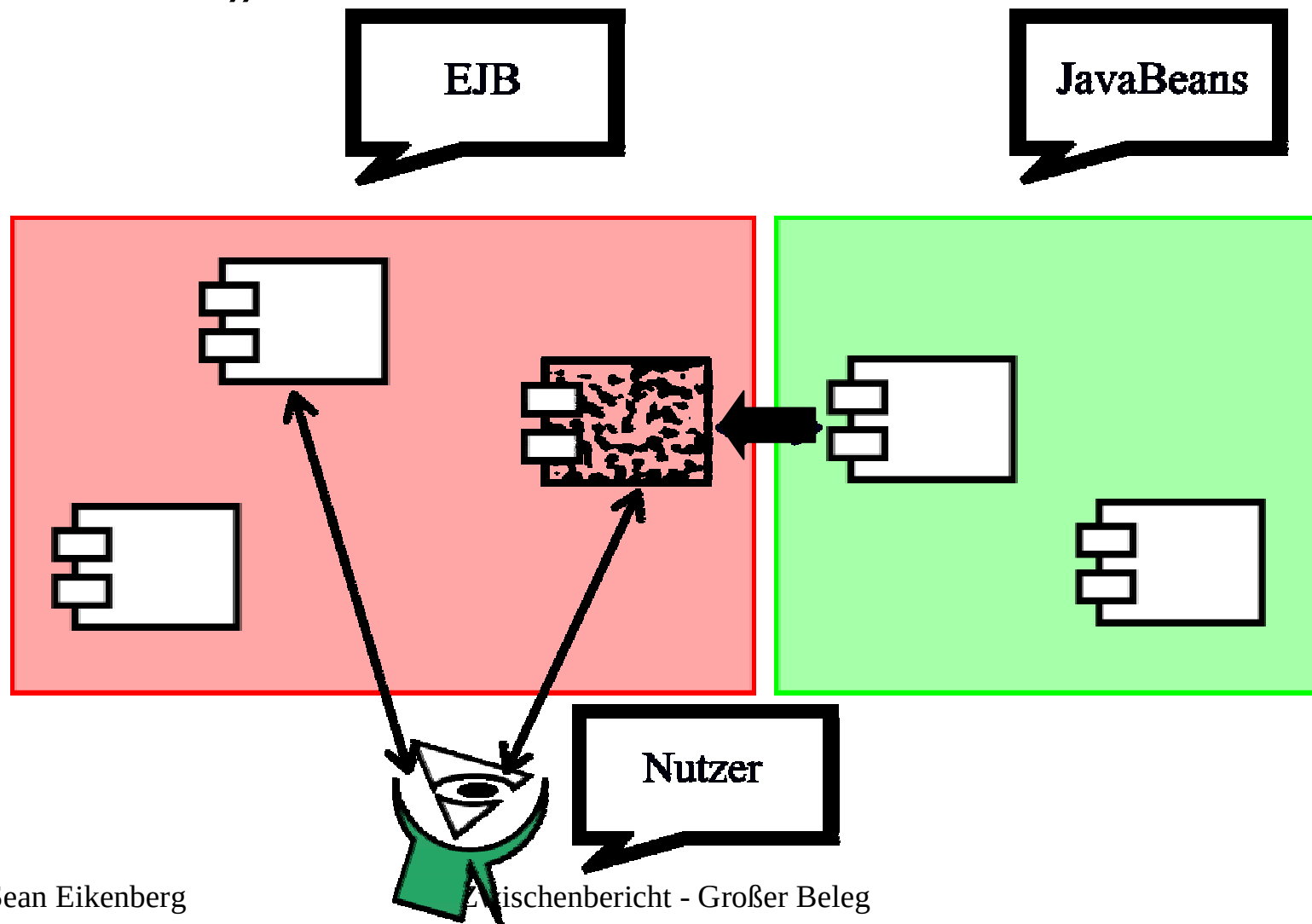
„Integratives Komponentenmodell“



„Integratives Komponentenmodell“

- Vorteil(e):
 - Einheitliche Komponentenschnittstelle
 - Einfache Integration weiterer Komponentenmodelle (Erweiterungsfähigkeit)
- Nachteil(e):
 - Hoher Komplexitätsgrad (Implementierung, Schulung)
 - Verlust spezieller Komponenteneigenschaften
 - Verlust bisheriger Entwicklungswerkzeuge

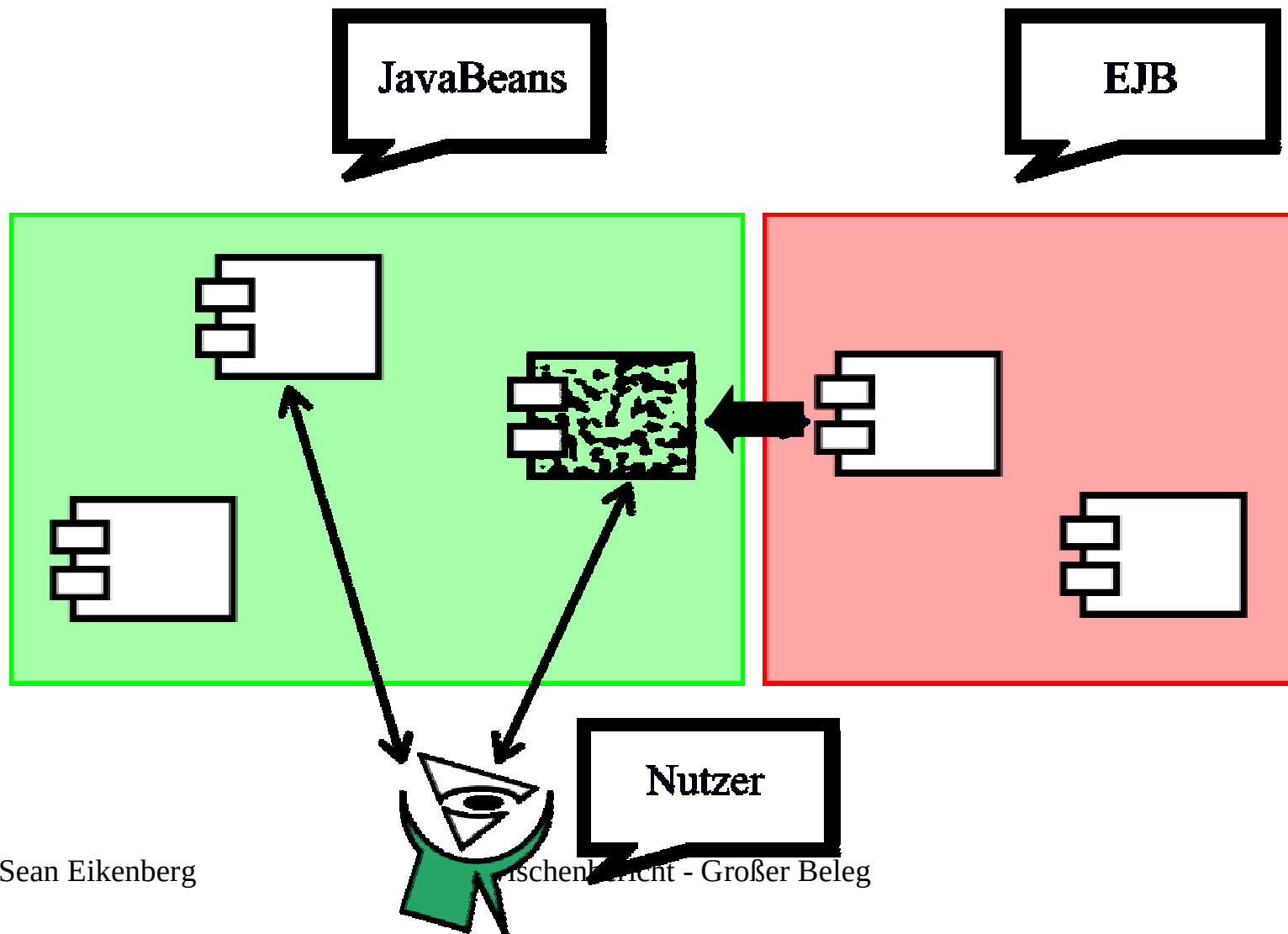
„Übersetzte JavaBeans“



„Übersetzte JavaBeans“

- Vorteil(e):
 - Wiederverwendung von Komponenten
- Nachteil(e):
 - JavaBeans-Eigenschaften gehen verloren
 - freien Ressourcenzugriff
 - Ereignisverarbeitung
 - Visuelle JavaBeans

„EJB-Proxies“



„EJB-Proxies“

- Vorteil(e):
 - EJB sind innerhalb von Entwicklungsumgebungen visuell mit JavaBeans verknüpfbar
 - EJB-Eigenschaften bleiben erhalten
- Nachteil(e):
 - Modifizierte EJB-Komponentenschnittstelle (Kompatibilität)

2. Vereinigungsmodelle: EJB & JavaBeans

- Fazit:
 - Zwei eher theoretische Modelle (hoher Komplexitätsgrad)
 - Zwei kurzfristig realisierbare Modelle (pragmatischer Ansatz)
- Als Ausgangsbasis für den zu implementierenden Generator ist das „EJB-Proxies“-Modell zu nehmen!

Zielstellung des GB / Gliederung

- Zusammenhang zwischen EJB & JavaBeans ausarbeiten.
- Mögliche Vereinigung beider Technologien diskutieren.
- ? • Entwurf eines Generators, der die Integration von EJB innerhalb des JavaBeans-Komponentenmodells ermöglicht.
- Nachweis der Funktionstüchtigkeit des Generators anhand einer Beispielanwendung.

3. Generatorentwurf: „EJB-Proxies“

- Aktueller Stand der GB-Arbeit!
- Idee:
 - Basierend auf vorhandenen EJB (session & entity beans) JavaBeans generieren, die als Proxy fungieren.
 - Diese „EJB-Proxies“ sind dann z.B. in Entwicklungsumgebungen als verknüpfbare Komponenten einsetzbar und erleichtern damit den Entwicklungsprozess von verteilten Anwendungen!

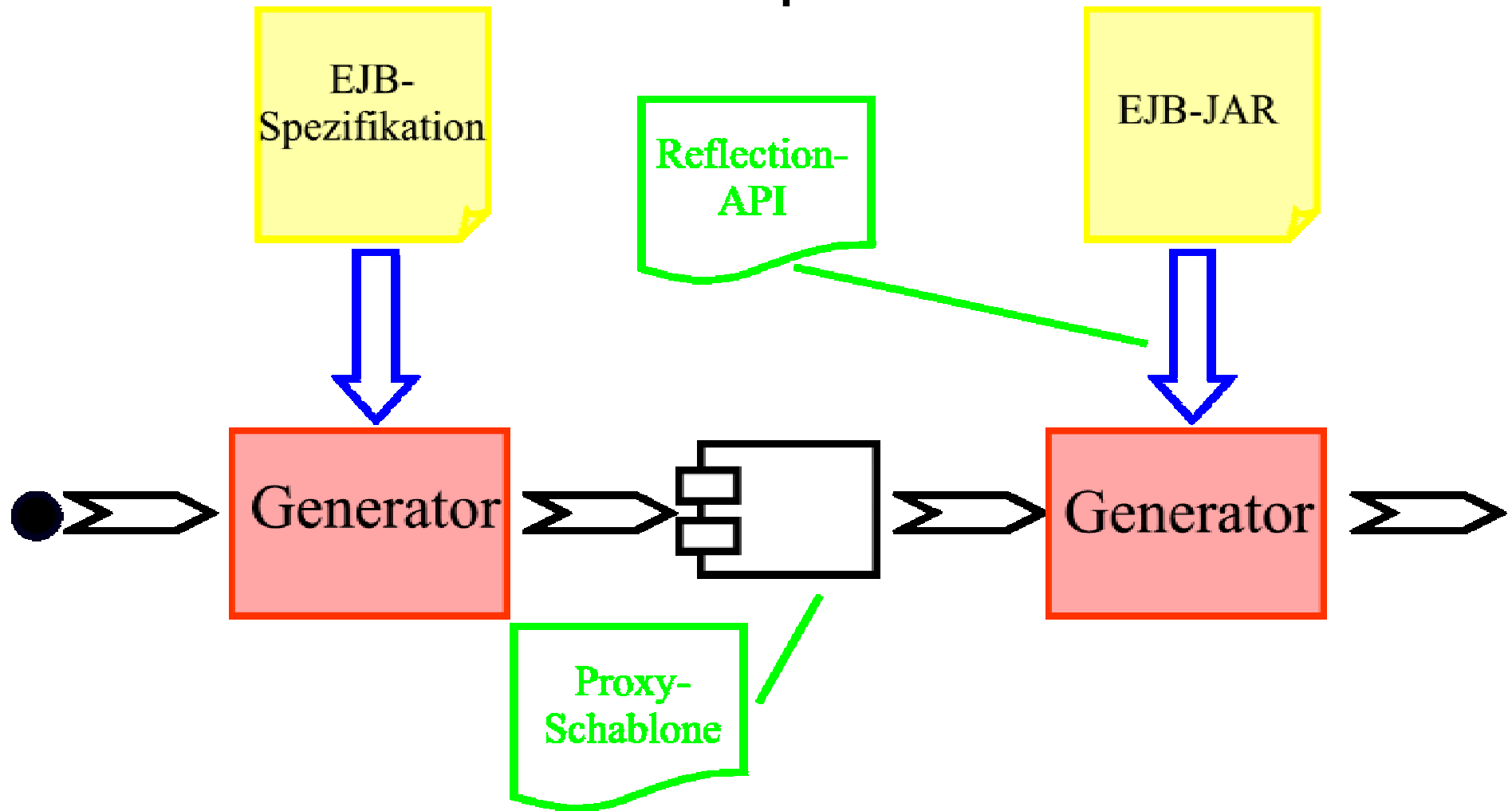
Voraussetzungen für den Generator

- EJB-Spezifikation:
 - Vorgegeben Schnittstellen (*EJBHome*, *EJBObject*)
- EJB-Jar-Files:
 - Fachliche Klasse
 - ➡ • Verwaltungs- & Zugriffsschnittstelle
 - ➡ • Hilfsklassen (z.B. eigene Exceptions)
 - ➡ • Deployment-Deskriptor
- Nutzereingaben

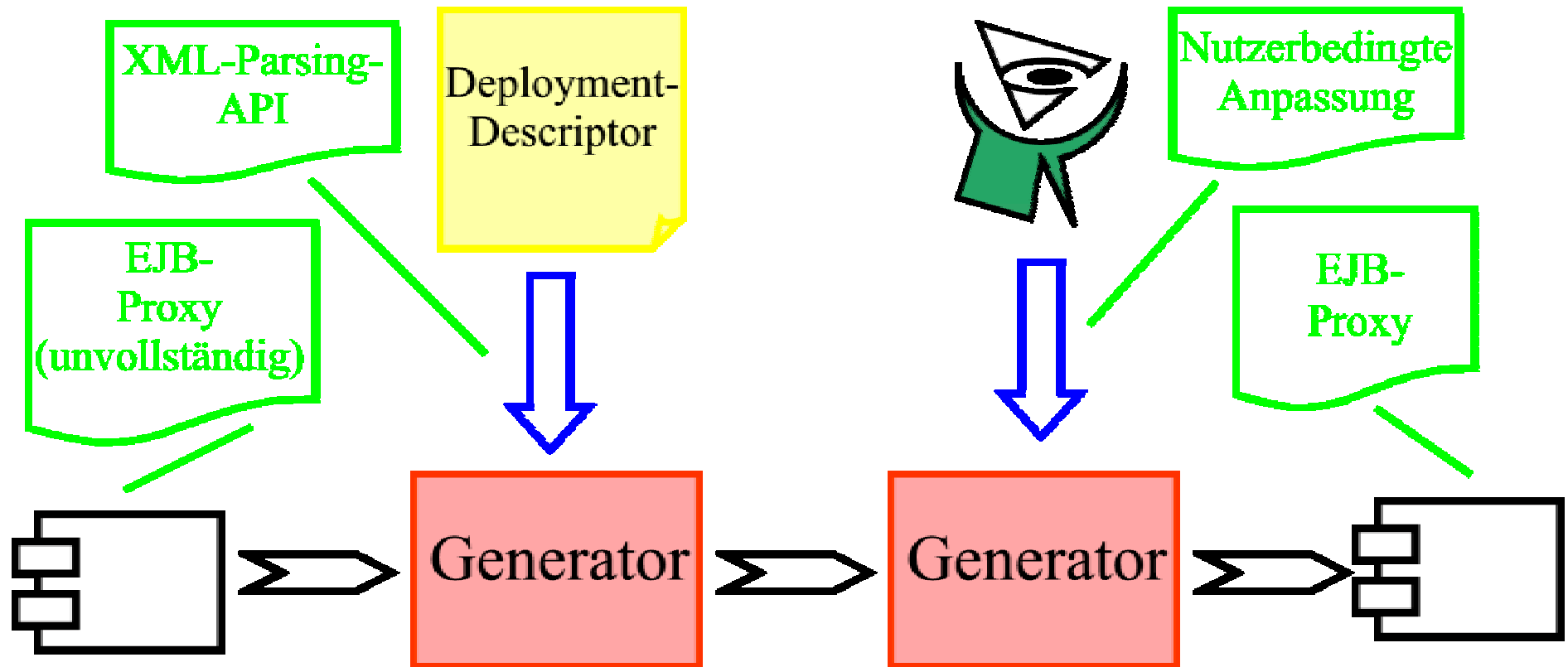
Generatorwerkzeuge

- Java-Reflection-API
- XML-Parsing-API

Generatorprozess



Generatorprozess



Code-Ausschnitt: Proxy-Schablone

```
import javax.ejb.EJBObject;
[...]  
public class ProxySchablone implements java.io.Serializable {  
    private transient EJBHome home;  
    private transient EJBObject remote;  
    public EJBMetaData getEJBMetaData() throws  
        RemoteException {  
        try {  
            return home.getEJBMetaData();  
        } catch (RemoteException e) { throw e; }  
    }  
[...]
```

Code-Ausschnitt: Proxy-Schablone

```
import javax.ejb.Handle;
public class ProxySchablone implements java.io.Serializable {
    [...]
    public Handle getHandle() throws
        RemoteException {
        try {
            return remote.getHandle();
        } catch(RemoteException e) {
            throw e;
        }
    }
    [...]
}
```

Code-Ausschnitt: EJB-Proxy

```
public class ShoppingCartProxy implements java.io.Serializable {  
    /** Holds value of property jndiName. */  
    private String jndiName = "ShoppingCart";  
    // This is EJB-specific  
    private transient ShoppingCartHome home;  
    // This is EJB-specific  
    private transient ShoppingCart remote;  
    /** Creates new Prototype */  
    public ShoppingCartProxy() {  
        this.connectToEJB();  
    }  
    [...]
```

Code-Ausschnitt: EJB-Proxy

```
private void connectToEJB() {  
    Object objref;  
    try {  
        // Establish connection to EJB  
        initCont = new InitialContext();  
        objref = initCont.lookup(this.jndiName);  
        // Get ejbHomeObject  
        home = (ShoppingCartHome)  
            PortableRemoteObject.narrow(objref,  
                ShoppingCartHome.class);  
    } catch(NamingException e) {  
    } catch(ClassCastException e) { }  
}
```

Code-Ausschnitt: EJB-Proxy

```
public void create(int count) throws CreateException,  
        RemoteException {  
    try {  
        // Get ejbObject  
        remote = home.create(count);  
    } catch(RemoteException e) {  
        throw e;  
    } catch(CreateException e) {  
        throw e;  
    }  
}
```

Code-Ausschnitt: EJB-Proxy

```
public void remove(String article) throws
    RemoteException, ArticleNotInShoppingCart {
    try {
        remote.remove(article);
    } catch(RemoteException e) {
        throw e;
    } catch(ArticleNotInShoppingCart e) {
        throw e;
    }
}
[...]
} // End of class ShoppingCartProxy
```


Zielstellung des GB / Gliederung

- Zusammenhang zwischen EJB & JavaBeans ausarbeiten.
- Mögliche Vereinigung beider Technologien diskutieren.
- Entwurf eines Generators, der die Integration von EJB innerhalb des JavaBeans-Komponentenmodells ermöglicht.
- ? Nachweis der Funktionstüchtigkeit des Generators anhand einer Beispielanwendung.

4. Funktionsnachweis anhand einer Beispielanwendung

- Voraussetzungen:
 - Entwurf & Implementierung von EJB
 - Alle EJB-Typen abdecken
 - Nutzung der Komponentenumgebung, des Sicherheitsmanagements und der Transaktionsunterstützung
 - Entwurf & Implementierung einer verteilten Anwendung
 - Client basiert auf visuellen JavaBeans

4. Funktionsnachweis anhand einer Beispielanwendung

- Erbringung des Nachweises:
 - Aus vorhandenen EJB mit Hilfe des erstellten Generators die EJB-Proxies generieren
 - Anwendungs-Client innerhalb Entwicklungsumgebung visuell mit EJB-Proxies verknüpfen
 - Verteilte Anwendung auf Funktionstüchtigkeit testen

Hinweis

- Ansatz für „Java class to EJB“-Generator („Übersetzte JavaBeans“) zu finden unter:
 - <http://java.sun.com>
 - Suchbegriff: „Tony Loton“ (Autor)
 - ODER:
 - <http://developer.java.sun.com/developer/technicalArticles/ebeans/reflection/index.html>