

XML-Morphing: EJBs - eine Frage des Stils?

Knut H. Meyer
danet IS GmbH, Stuttgart

W i s s e n w i e e s g e h t

XMM

xpath

XSL

Komponenten

DOM

XSLT

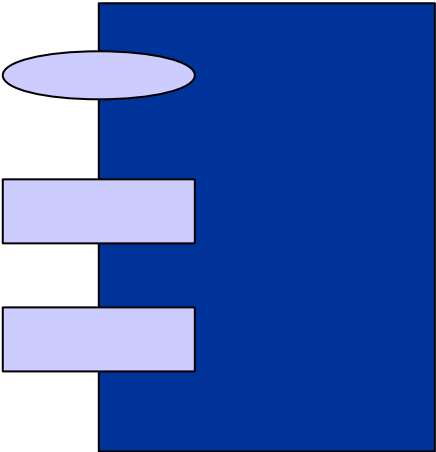
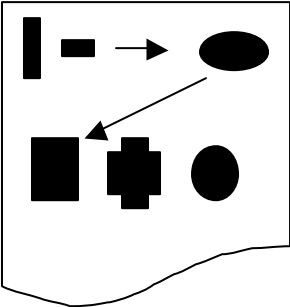
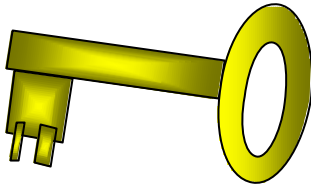
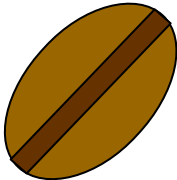
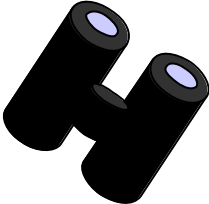
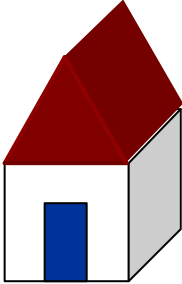
EJB

SAX

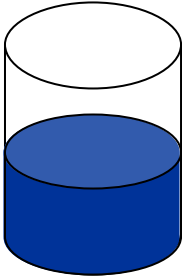
XMI

- ✍ EJBs sind keine JavaBeans!
- ✍ server-seitige Komponenten
- ✍ Eingebautes Transaktionsmanagement
- ✍ Transport unabhängig
- ✍ vier Kategorien:
 - ✍ stateless Session Beans
 - ✍ stateful Session Beans
 - ✍ container-manager Entity Beans
 - ✍ bean-managed Entity Beans

EntityBean (container-managed) Bestandteile



EntityBean



- ✍ Trennung einer semantischen Einheit:
 - ✍ Deployment Descriptor und Bean
 - ✍ Methodendefinitionen im Remote Interface und der Bean Klasse
 - ✍ Methodendefinition im Home Interface und in der Bean Klasse
 - ✍ Datenbanktabelle und Deployment Descriptor und Bean
- ✍ Folgen:
 - ✍ zwangsläufige Inkonsistenzen
 - ✍ keine Dokumentation (JavaDoc) pro EJB

- ✍ Copy and Paste ist kein Designmodell/Pattern
- ✍ Wizards sind was für Warmduscher
- ✍ Der redundante Tippaufwand fördert Fehler und Unlust
 - ✍ Eine abstrakte Beschreibungssprache muß her

X-tensible:

-  leichte Erweiterbarkeit im Lichte neuer Erkenntnisse

Maschinelle Verarbeitung:

-  Geringer Aufwand für die Implementierung von Werkzeugen, die die Beschreibungssprache benutzen

Lesbarkeit:

-  Wenig Redundanz (kein Copy & Paste nötig) oder syntaktischer Ballast
-  Einfache Editierbarkeit (mit dem EMACS)

Was zeichnet XML aus?

- ✍ Eine Vereinfachung von SGML
- ✍ Fertige Parser oder Transformationswerkzeuge frei verfügbar
- ✍ Standardisiert
- ✍ Erweiterbar
- ✍ Lesbarkeit:

```
<rechnung>
```

```
  <kunde>Knut Meyer</kunde>
```

```
  <verkaeufer>XMLTrek Ltd.
```

```
    <slogan>To boldly go where no company has gone  
    before</slogan>
```

```
  </verkaeufer>
```

```
  <ware>Pokemon, schwarz</ware>
```

```
  <datum jahr=„2000“ monat=„Juni“ tag=„28“/>
```

```
</rechnung>
```

```
<business-object name=„MeinObjekt“ >
  <db pool-name=„DBPool“ table=„objects“/>
  <attribute name=„MeinAttribut“ type=„String“ primary-
    key=„yes“/>
  <finder name=„ByAttribut“ >
    <expression>(= $attr attribut)</expression>
    <parameter name=„attr“ type=„String“/>
  </finder>
  <business-method name=„MeineBusinessMethode“ >
    <parameter name=„param“ type=„int“/>
  </business-method>
</business-object>
```

- ✍ DTD: Document Type Definition
 - „Syntaxbeschreibung eines Dokumenttyps“
 - Beispielsweise die Beschreibung eines Dokumenttyps
 - „Enterprise Java Bean Definition“

```
<!DOCTYPE business-object [  
<!ELEMENT business-object ( description,  
    persistence,  
    control-descriptors,  
    environment,  
    attributes,  
    associations?,  
    create-methods?,  
    remove-methods?,  
    business-methods?,  
    sql-finders?,  
    finders? ) >  
<!ATTLIST business-object  has-history      (yes|no) "no"  
    name          CDATA #REQUIRED  
    package       CDATA #REQUIRED >
```

- ✍ XMI: XML Metadata Interchange
Offenes Modell zum Informationsaustausch von Modellen und Daten. XMI basiert auf OMGs MOF, UML und auf XML
- ✍ Entwickelt von: IBM, Unisys, Oracle Corporation, Cooperative Research Centre for Distributed Technology (DSTC), Platinum Technology Inc., Fujitsu, SofTeam, Recerca Informatica und Daimler-Benz.
- ✍ XMI ist sehr aufwendig zu schreiben, außer man benutzt Werkzeuge wie bspw. Rational Rose mit dem XMI Toolkit von IBM Alphaworks.
- ✍ Rational Rose ist sehr stark objekt-orientiert aber nicht komponenten-orientiert.
- ✍ Einige Informationen lassen sich nicht direkt in Rational Rose modellieren.

Aber es gibt noch mehr...

- ✍ Leere Rahmen für die Businesslogik Implementierung
- ✍ Automatische Getter/Setter Methoden
- ✍ Automatische Create/Remove Methoden
- ✍ Generierung **einer einzigen** Dokumentation pro Komponente
- ✍ Benutzung von komplexen Typen in EJBs:
 - ✍ Speicherung der serialisierten Objekte als BLOBs
- ✍ Automatische Primärschlüsselgenerierung:
 - ✍ ORACLE Sequences

- ✍ Verwaltung von Historien von Objekten
- ✍ Möglichkeit der Generierung automatischer Tests
- ✍ Vergleich einer reellen Datenbank gegen eine XML Datei

- ✍ Eine EJB sollte im Idealfall nichts wissen über Beziehungen zu anderen EJBs (Wiederverwendbarkeit):
 - ✍ Die Beziehungslogik wird in eine übergeordnete (stateless) SessionBean ausgelagert.
 - ✍ SessionBean erlaubt Verknüpfung von Relationen, das Lösen derselben und Navigation durch die Relationen
- ✍ ManagerBeans

✍ Navigation/Suche:

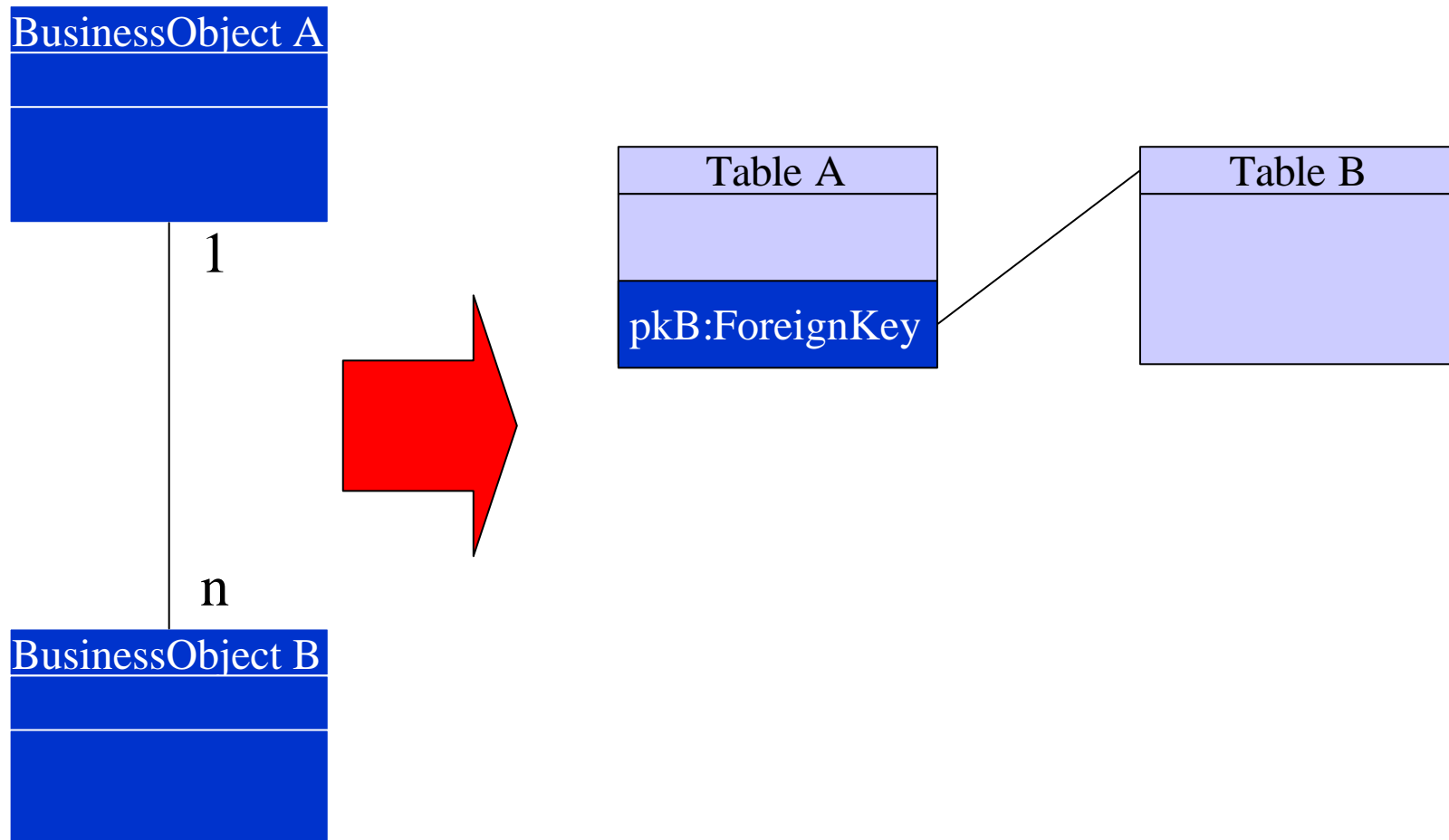
- ✍ Ansteuern der EJBs, die in Beziehung stehen

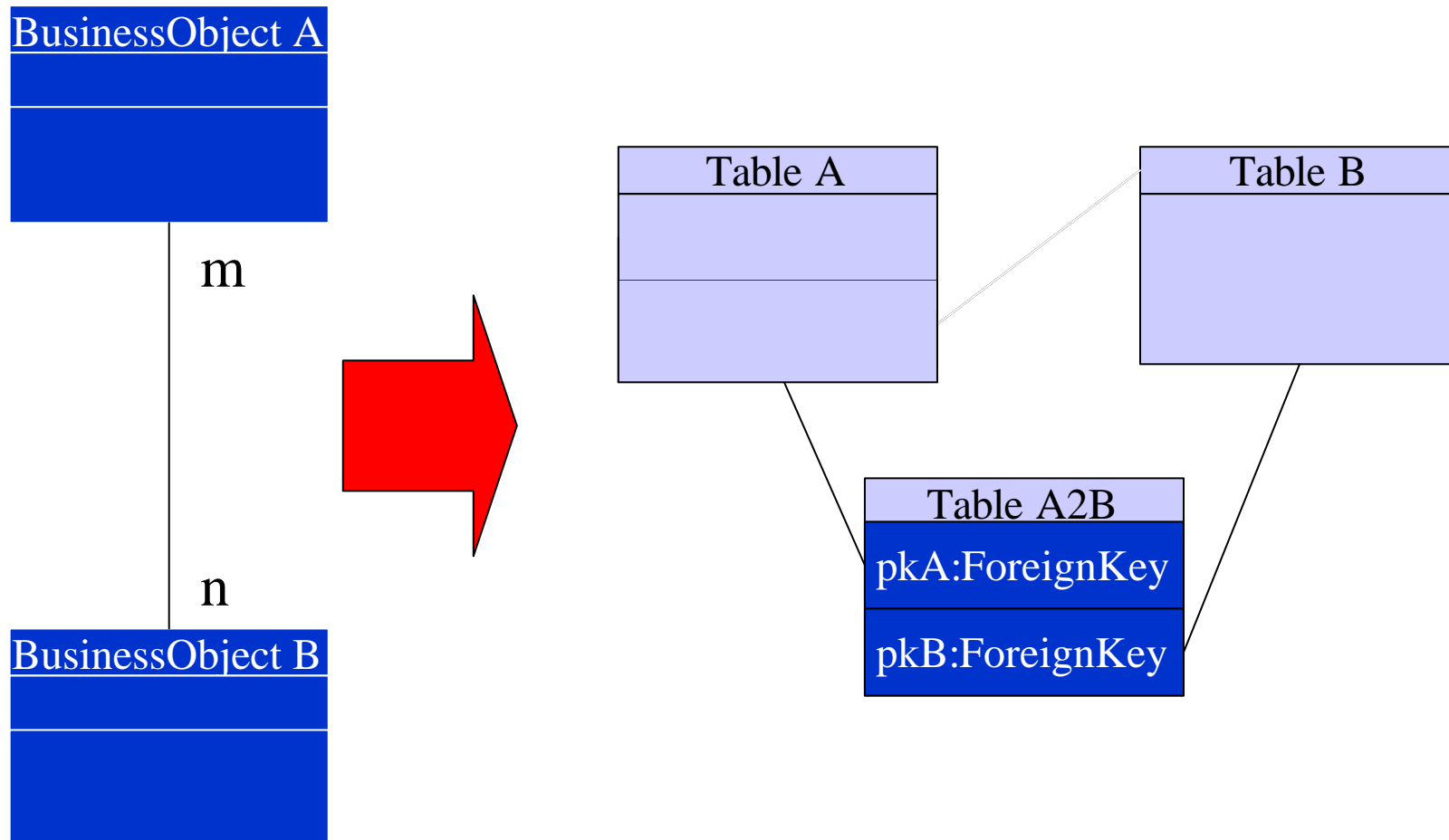
✍ Verbinden/Trennen:

- ✍ Herstellen oder Trennen einer Beziehung zwischen zwei oder mehreren EJBs

✍ Löschen:

- ✍ Löschen einer EJB und aller zugehörigen Relationen, eventuell sogar das Löschen untergeordneter EJBs





✍ Morphen mit Stil:


✍ Stylesheetmechanismen:


✍ XPath, XSLT, XSL


✍ Morphen ohne Stil:

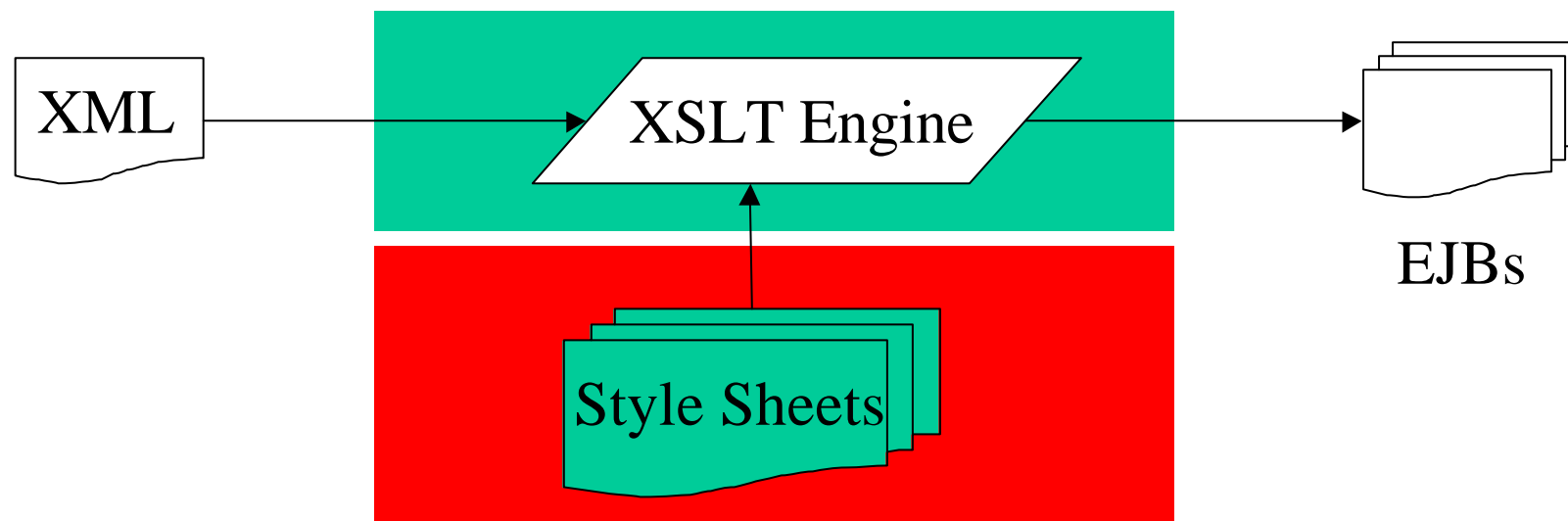
✍ Selbstgeschriebene Parser:

✍ SAX oder DOM?

-  **XPath:** Eine Sprache zur Addressierung von XML Dokumentteilen.

-  **XSLT:** Eine Sprache, um ein XML Dokument in ein anderes XML Dokument zu transformieren. Zur Adressierung wird XPath verwendet.

-  **XSL:** eXtensible Style Language. XPath + XSLT + FO (Formatting Objects)



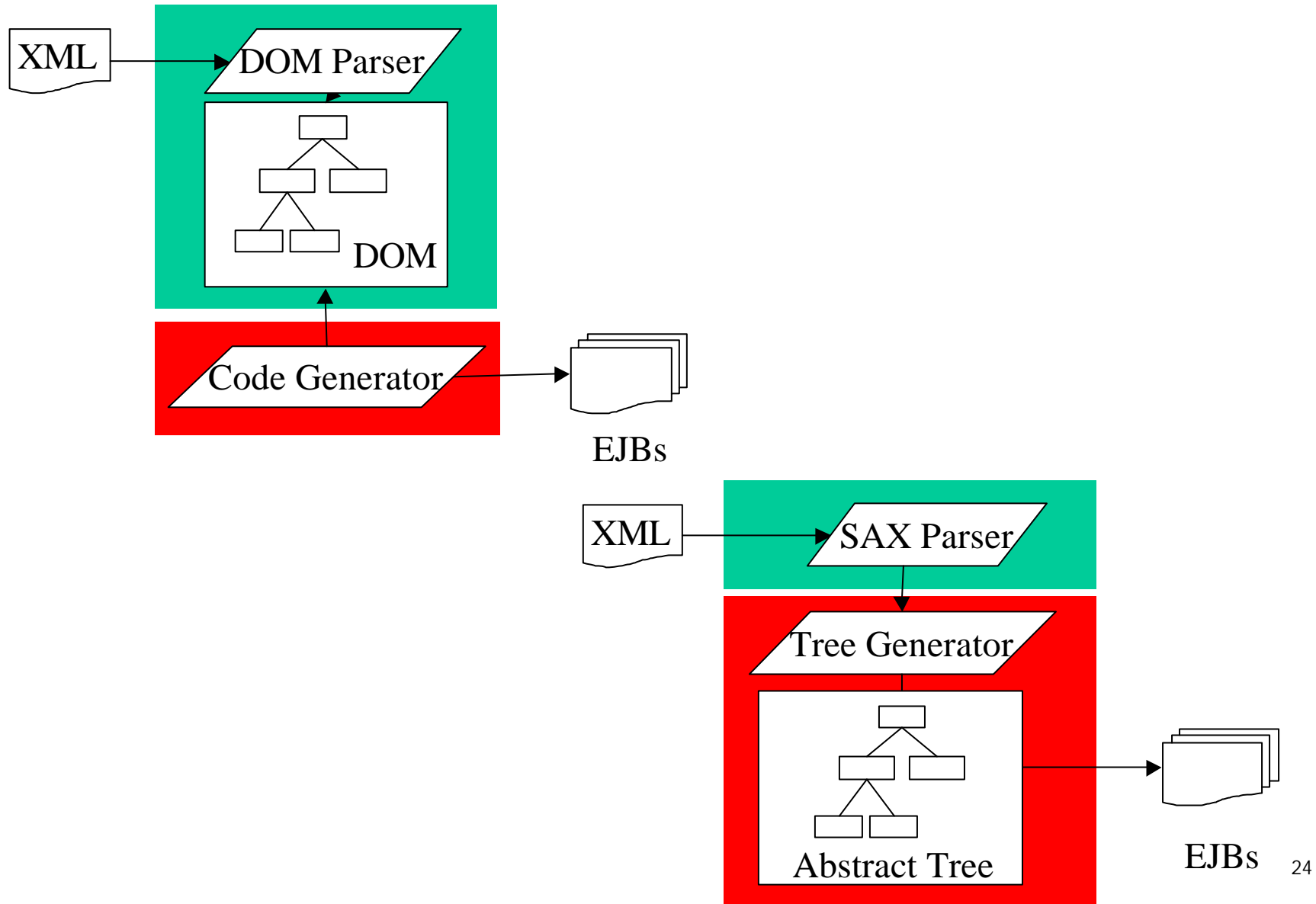
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0" result-ns="">
  <xsl:template match="business-object">
    ...
    <xsl:text>
      public void.ejbCreate(</xsl:text>
      <xsl:for-each select="attributes/attribute">
        <xsl:value-of select="@type"/>
        <xsl:text> the</xsl:text>
        <xsl:value-of select="translate(substring(@name,1,1),'qwertzuiopas
dfghjkllyxcvbnm','QWERTZUIOPASDFGHJKLYXCVBNM')"/>
        <xsl:value-of select="substring(@name,2)"/>
        <xsl:if test="not(position()=last())">
          <xsl:text>,</xsl:text>
        </xsl:if>
      </xsl:for-each>
      <xsl:text>)</xsl:text> throws CreateException; </xsl:text>
    ...
```



- ✍ XSLT ist eine Sprache zur Transformation von XML in XML
- ✍ Unlesbar:
 - ✍ Durchmischung Java Code und XSLT ist schlecht zu erkennen
 - ✍ Programmiersprachen mit XML darzustellen ist keine gute Idee
- ✍ Konsistenzprüfungen:
 - ✍ Kaum Konsistenzprüfungen oder Abgleich mit der Datenbank in XSL möglich

- ✍ **DOM:** Document Object Model. Eine API Definition für den Zugriff auf XML oder HTML Dokumente.
- ✍ **SAX:** Simple API for XML event based parsing.
- ✍ **Write 'n' Run:** Write Once & Run Anywhere = JAVA

Morphen ohne Stil: DOM versus SAX



- ✍ Erzeugung eines Objektmodell aus XML
- ✍ Auffüllen mit bekannten Daten:
 - ✍ automatisches Create mit allen Attributen
 - ✍ Getter/Setter
- ✍ Konsistenzprüfungen
- ✍ Auswertung von Code Templates für alle Ausgabedateien
- ✍ Code Template Fähigkeiten:
 - ✍ Bedingungen
 - ✍ Schleifen
 - ✍ Variablenersetzungen
 - ✍ Rekursion

```
createMethod.comment=\
```

```
/**\n\
```

```
 * ${comment} ${ejb.name}.\n\
```

```
 * ${description}\n\
```

```
 *\n\
```

```
 ${parameters:parameters.javadoc:parameters.javadoc.separator}\n\
```

```
 *\n\
```

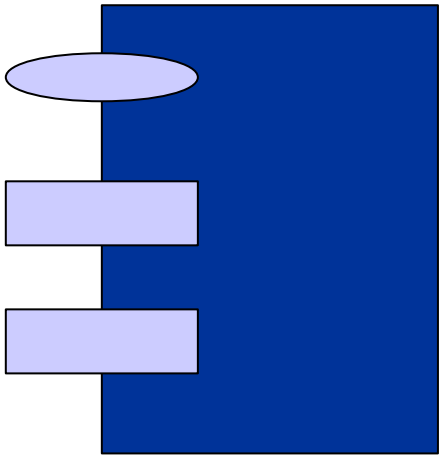
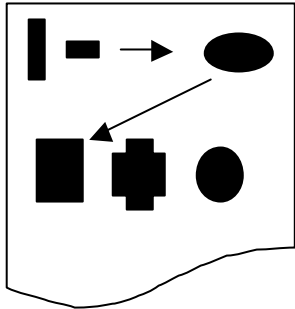
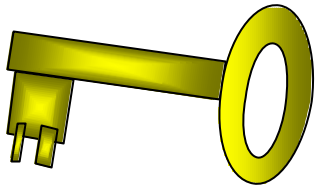
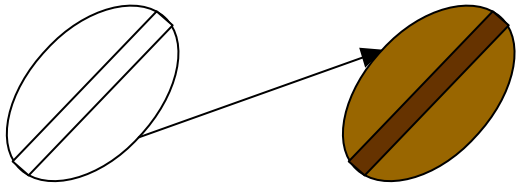
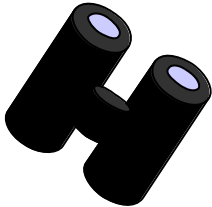
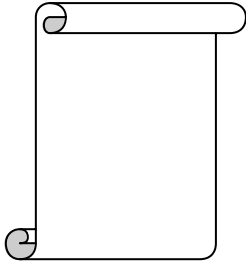
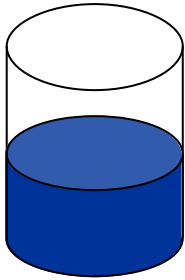
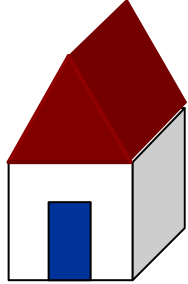
```
 * @returns  ${ejb.name} \tthe created bean\n\
```

```
 */
```

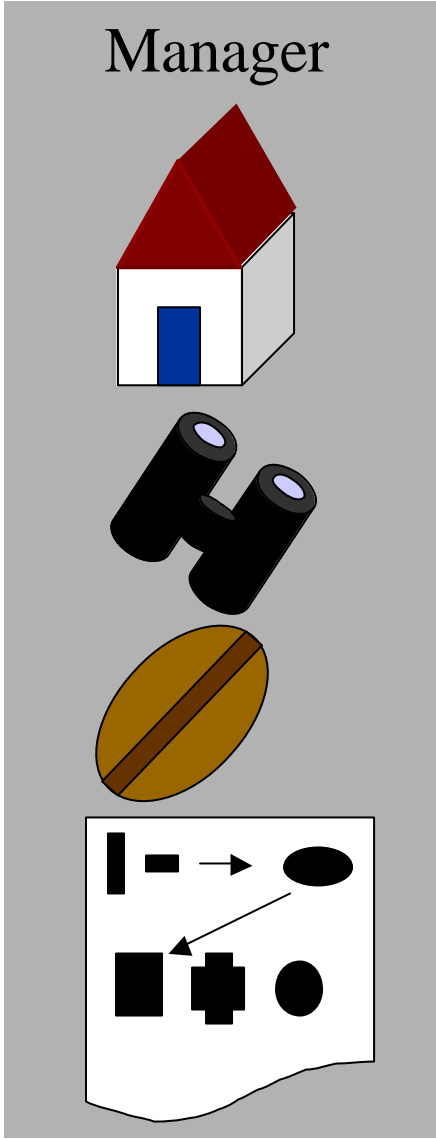
```
parameters.javadoc=  * @param  ${name} \t ${description}
```

```
parameters.javadoc.separator=\n
```

Was dabei raus kommt



Komponente

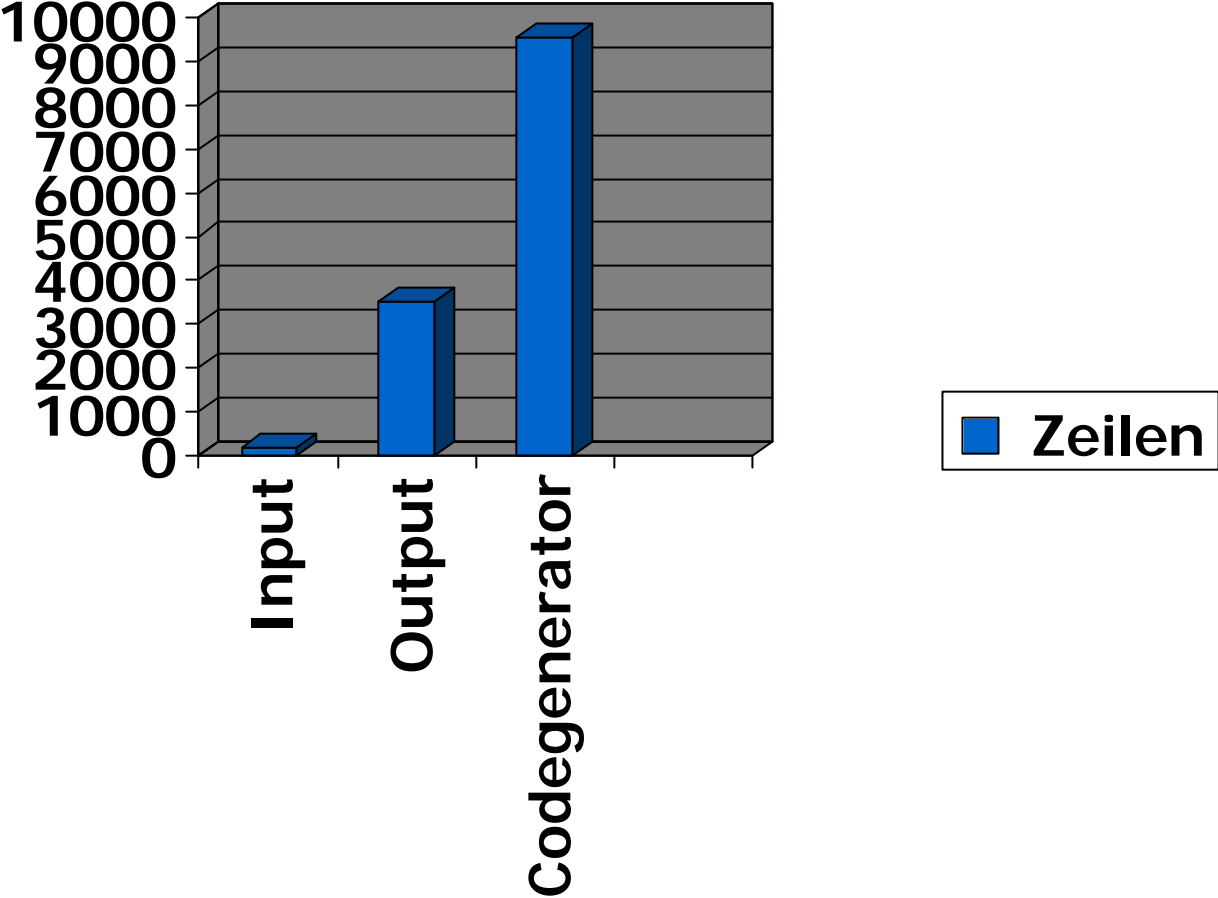


```
<?xml version="1.0"?  
<business-object name="IPInterface" package="ip" >  
  <attributes >  
    <attribute name="ipaddress" type="IPAddress"/>  
  </attributes >  
  <associations >  
    <association to="DeviceType" type="many-to-one"/>  
  </associations >  
  <create-methods >  
    <create-method >  
      <parameters >  
        <parameter name="ipaddr" type="IPAddress"/>  
      </parameters >  
    </create-method >  
  </create-methods >  
  <finders >  
    <finder name="ByIPAddress" expr="(= $addr ip)" >  
      <parameters >  
        <parameter name="addr" type="int"/>  
      </parameters >  
    </finder >  
  </finders >  
</business-object >
```

```
/*  
 * DO NOT EDIT!!! This source was generated automatically  
 */  
package ip;  
  
import javax.ejb.*;  
import java.rmi.RemoteException;  
import utils.XMLBean;  
import utils.IPAddress;  
  
/**  
 * abstract bean class for the entity bean IPInterface.  
 */  
public abstract class AbsIPInterfaceBean extends XMLBean  
    implements EntityBean  
{  
  /**  
   * entity context  
   */  
  protected transient EntityContext ctx;  
  
  public long dbIpaddress;  
  transient protected IPAddress ipaddress;
```

- ✍ Definition der Business Objekte
- ✍ Implementierung der Businesslogik/-methoden
- ✍ Implementierung der GUIs
- ✍ Testen
- ✍ Installieren

- ✍ Single-source:
 - ✍ Single Point of Error
- ✍ Um zusätzliche Fähigkeiten aller Businessobjekte einzuführen, ist nur eine Änderung am Codegenerator notwendig
- ✍ Höhere Effizienz
- ✍ Weniger Fehler in den EJBs
- ✍ EJBs sind konsistent:
 - ✍ Relationshandling
 - ✍ Naming
 - ✍ Aufbau



- ✍ Keiner kann mehr EJBs programmieren
 - ✍ Das Wizard-Phänomen
- ✍ Hand-optimierter Code kann besser sein

- + Automatische DB Connection Pooling
- + Automatische Threadverwaltung
- + Objektpooling
- + einfache SQL Kapselung
- + unabhängig vom Transportmechanismus
- + Transaktionsmechanismus
- Synchron
- Benutzung im Client aufwendig (lookup, create, Exceptions)
- langsam, wenn einzelne Attribute hin- und hergeschoben werden
- Debugging aufwendig (erzeugter Code des Applikationsservers nicht lesbar)
- Abhängig vom Applikationsserver Hersteller
- Relationale Schemata nicht gut unterstützt