

Thing-oriented programming

Inhaltsverzeichnis

I. Thing-oriented programming

II. Ercatons

III. Beispiele

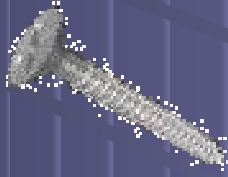
Teil I

Thing-oriented programming

Motivation

Software ist schwieriger als Hardware...

...aber es sollte nicht so sein!



Es ist kein Problem mit einer Säge eine Schraube zu verkleinern.

Jedoch ist es (beinahe) unmöglich `Schraube.class` zu ändern ohne bis in den Source Code zurück zugehen.

Das Ercato Manifest

Die Ausnahme ist die Regel.

- §1 Unsere Welt ist reich und komplex
statt gut strukturiert und einfach.
- §2 Software muss unregelmäßige Strukturen abdecken
statt die regelmäßigen.
- §3 Ein Software System ist ein organisches Wesen
statt einer Menge mathematischer Algorithmen.
- §4 Software Komponenten sind ein integrierter Teil unserer Welt
statt Wesen auf einer höheren Ebene.
- §5 Software Entwicklung entwickelt sich vom Kleinen ins Große
statt vom Konkreten ins Abstrakte.

Unterstützende Beobachtungen

- Direkt:
 - Das erhöhte Interesse in dem „Agile Manifesto“
 - Immer wieder das Versagen von großen Software-Projekten
 - Lächerlich ausgeschöpfte Software Budgets
- Indirekt:
 - Nur die IT braucht eine *Programmiersprache* fürs Modellieren
 - Wie einfach große Maschinen/Gebäude aus vielen kleinen skalierbaren Teilen zusammengefügt werden.

Thing-oriented programming

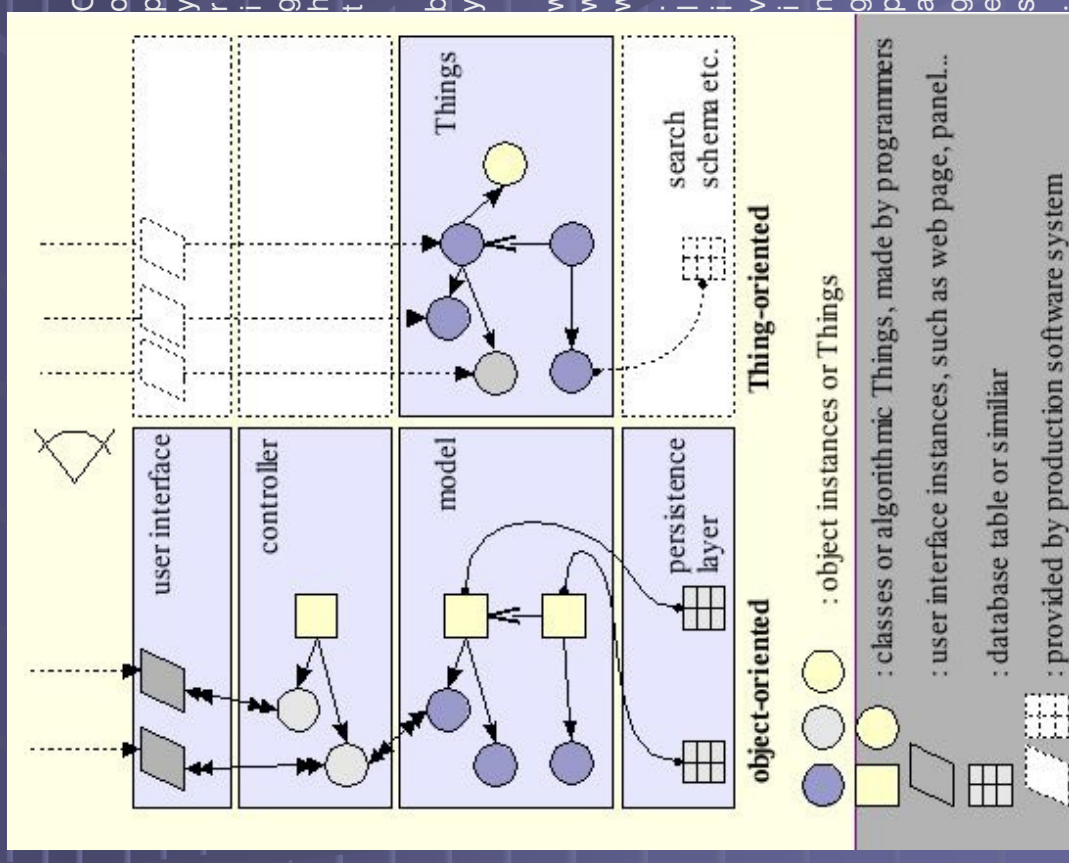
Def.: Thing

- Software Objekte sind Modellabhängig oder Modellbetrieben
- Modelle ändern sich
- Eine reiche Instanz, welche kein Modell braucht: „thing“
- things sind so nahe wie mögliche an *echten* Objekten
- Reiche Instanzen brauchen keine Klassen vgl. „prototype-based“
- things ermöglichen das Manifest



Things sind nackt

- Things implementieren das „nackte Objekt“ Geflecht per Definition
- *Things verstecken sich nicht*
- Alle Geschäftslogik ist in einem einzelnen Rang auch in Architekturen mit Mehrfachrängen.
- Things sind flexibler als das „nackte Objekt“ Geflecht. Sie können z.B. sich anziehen ;-)



Zusammenfassung: Things

- Things sind Software Wesen welche ihren echten Gegenständen so gut wie möglich ähneln.
- Wenn das echte Gegenstück virtuell ist, ist das Thing das Gegenstück. (Bankkonto, Steuererklärung,...)
- Things hängen nur an einer Programmiersprache, wenn Algorithmen involviert werden.

Teil II

Ercatons

E-Market -> E-Mercato -> ercato ->ercaton



Ercaton „implementiert“ Thing

- *Ercatons* beinhalten den Status und das Verhalten
- *Ercatons* sind Things basierend auf XML und z.B. Java
- *Ercatons* sind Geschäftsobjekte
- *Ercatons* sind Dokumente
- *Ercatons* verbinden viele alte Ideen in einem Konzept

Ercatons... beinhalten den Status und das Verhalten

Ein Ercaton beinhaltet

- ▶ Geschäftsstatus 1234567 \$
- ▶ Geschäftslogik transfer()
- ▶ Benutzeroberfläche web page, etc...
- ▶ Integration, Austausch und Ausgabe print, sent-to-SAP
- ▶ Modell Eigenschaften expands BankAcc

In eine einzige Form

...ohne programmieren zu müssen.

(nur mathematische Algorithmen brauchen Programmierung)

Mit Ercatons baut man, anstatt dass man sie programmiert.

Ercatons... sind Things basierend auf XML und z.B. Java

Def.: Ein Thing ist ein in sich geschlossenes Wesen
mit Identität, Verhalten, inneren Status und Struktur,
mit Benutzer- und Modellschnittstellen, mit Eigentumsrechten und
mit Selbstbestimmtem Lebenszyklus und
mit Rechten, in sowohl der Software als auch der Realität.

Def.: Ein Ercaton ist ein Thing
mit min. einer Modellschnittstelle zu XML,
mit Vererbung und Polymorphismus,
mit einer veränderlichen Web-Schnittstelle,
mit Datenbank und Transaktion Unterstützung und
mit autonomen Leben.

Das bedeutet, dass ein Ercaton unabhängig ist, z.B. hängt es von keiner Klasse ab, trotzdem hat es einen einzigartigen Namen und ist standhaft und geschützt, und jedes Ercaton ist ein Individuum, da es nicht zwei gleiche gibt.

Ercatons... sind Geschäftsobjekte

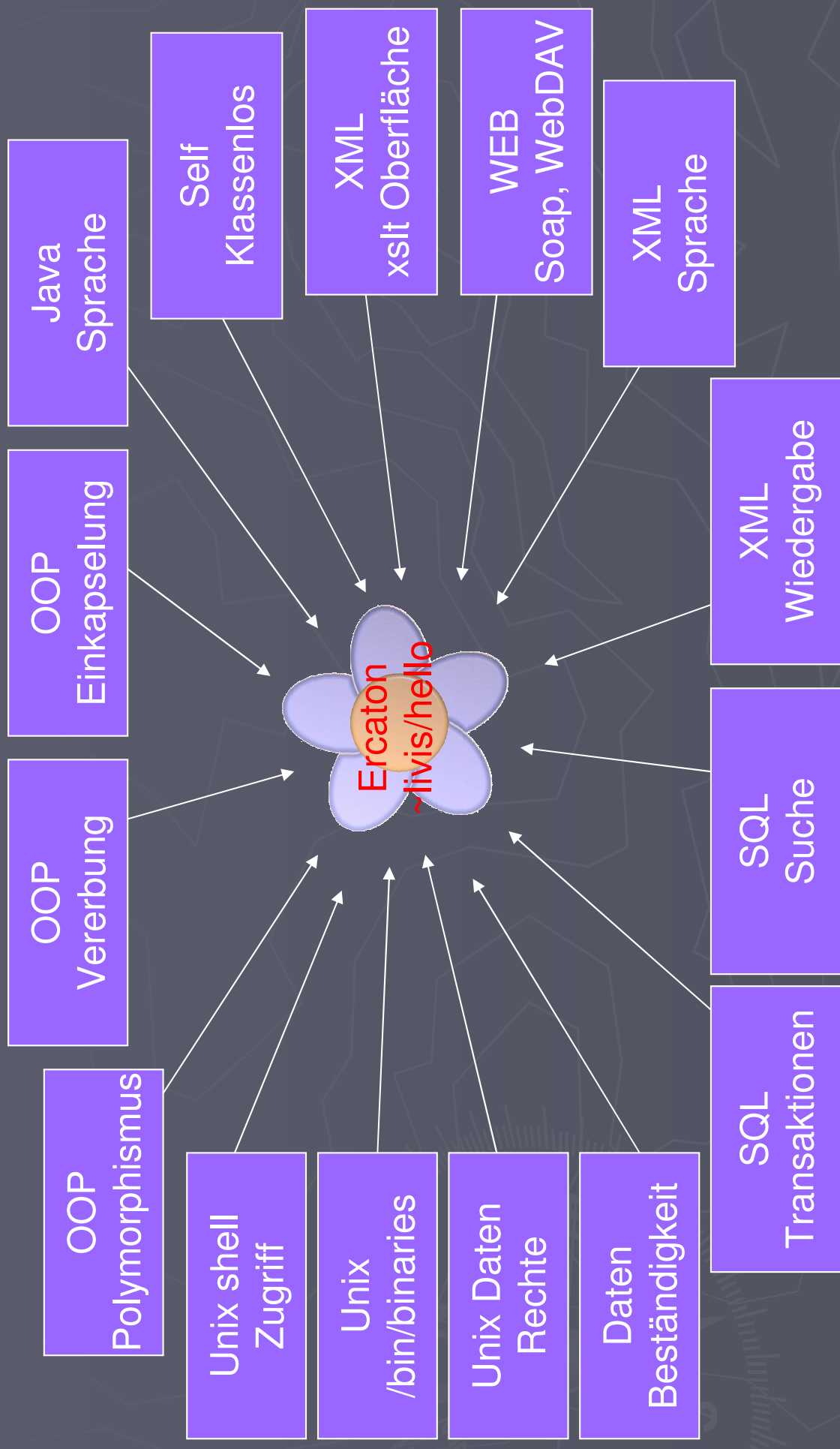
- ▶ Ercatons verbinden oft ein Geschäftsobjekt oder –Prozess. Geschäftswissen wird in XML und anderen optionalen Sprachen (incl. Java) repräsentiert.
- ▶ Dies ergibt Geschäftsobjekte welche völlig programmierbar, geschützt durch Transaktionen und Rechte, und unterstützt durch die Suche in einer Datenbank sind. Sie sind auch standardhaft per Definition.
- ▶ Ihre dynamische Vererbung erlaubt die Extraktion von gemeinsamen Teilen in die Geschäftslogik.
- ▶ Ercatons werden durch das XML Geschäftsmodell ausgedrückt. Dies formt eine Methode, nicht eine Rahmenbedingung.

Ercatons... sind auch Dokumente.

- ▶ Benutzerschnittstellen können von Vorlagen (Stylesheets) erzeugt werden, die sich dem Ausgabeziel anpassen können.
- ▶ Nicht-XML-Ressourcen werden als „resource“ Ercatons angezeigt. Dies schließt JAR Dateien ein und z.B. sich verändernder Code.
- ▶ Ercatons haben Versionsnummern.
- ▶ Ercatons haben Besitzer und können eine Kette von Möglichkeiten definieren, um ihren Status zu beschützen.
- ▶ SQL-ähnliche Anfragen von inneren und äußeren Anschlüssen können dazu benutzt werden, um XML-Daten zu erhalten.

Ercatons...

verbinden viele alte Ideen in einem Konzept



Zusammenfassung von Pattern

Prinzipiell ist ein System von Ercatons eine Agentengesellschaft, die sich untereinander Nachrichten senden können.

- ▶ Firewall-Pattern: selbst-schützend; schützt die Ercatons und unterscheidet zwischen Text und Daten
- ▶ Face-Pattern: selbst-erklärend; verändert das Aussehen von Ercatons
- ▶ Doctor-Pattern: selbst-heilend; verändert Ercatons und funktioniert als Garbage Collector
- ▶ Workflow-Pattern: selbst-konfigurierend; durch einen Index wird heraus gefunden, welche Transaktionen getätigt werden dürfen
- ▶ Structure Constraint-Pattern: selbst-optimierend; die innere Struktur wird normalisiert und werden bei Veränderung aufgerufen

Teil III

Beispiele

Ein richtiges Beispiel

Aufgabe:

Entwerfen & Implementieren von
einer „Adressen Manager“ Anwendung

Detail:

Soll ausführbar und netzwerkfähig sein

Budget:

...15 Personen Minuten

los geht's....

~livis/adr/bunny ist ein Ercaton

```
<?xml version="1.0" encoding="iso-8859-1"?>
<address xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
  <erc:id>~livis/adr/bunny</erc:id>
  <erc:clone>~livis/adr/base<erc:clone>
    <name    >Easter Bunny</name>
    <street  >Wiese 7</street>
    <zipcode >12345</zipcode>
    <phone   >0190 666 666</phone>
  </address>
```

Dies ist *Easter Bunnie*’s Adresse – und es ist ein Ercaton.

~livis/adr/bunny und (4) Freunde

Copyright by www.livis.de

Ercato Home - Microsoft Internet Explorer

Datei Bearbeiten Ansicht Favoriten Extras ?

Google Adresse http://www.living-pages.de/eri

in die Zwischenablage alle markieren

/Address

Name (>) **street** **zipcode** **city** [tmp]

[+] name: a b c d e f g h i j k l m n o p q r s t u v w x y z * 1 - 2 max: 20 Go

☐ !New 0

☐ Easter Bunny Wiese 7 12345 Waldbrühl

edit delete copy check

name	Easter Bunny
street	Wiese 7
zipcode	12345
city	Waldbrühl
phone	0190 666 666

http://www.living-pages.de/erc/saton~demo/Easter-Bunnyfwl0xr Internet

It's not a trick...

- Wir haben nur ein XML Dokument mit den minimalen Adressdaten von Mr. Easter Bunny geschaffen.

Wie können wir das Adressmanagerproblem lösen?

- Da wir von 3 (kl.) Ercatons erben und diese benutzen.

Vererbung wird in XML ausgedrückt.

- Normalisierung von Daten etc. (verhindert Duplikate etc.)!
- Ercatons geben einige Maßnahmen diese Problem zu lösen:
 - <erc:clone>
 - <erc:id-ref>
 - erc:expand

~livis/adr/base : Klasse oder Rahmen?

```
<?xml version="1.0" encoding="utf-8"?>
<address xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
          xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions" >
  <erc:id>~livis/adr/base</erc:id>
  <erc:type>prototype</erc:type>
  <erc:catalog category="/Address" id-ref="~livis/catalog"/>

  <name   erx:field-ref="string" erc:index="~livis/catalog"/>
  <street erx:field-ref="string" erc:index="~livis/catalog"/>
  <zipcode erx:field-ref="int"   erc:index="~livis/catalog"/>
  <city    erx:field-ref="string" erc:index="~livis/catalog"/>
  <phone   erx:field-ref="string"/>

  <erc:action name="edit">   /bin/edit   </erc:action>
  <erc:action name="delete"> /bin/rm!wizard </erc:action>
  <erc:action name="copy">   /bin/cp!forEdit </erc:action>
  <erc:action name="check">  ~livis/check.xsl
    <erc:arg name="default">Waldbröhl</erc:arg>
  </erc:action>
  <erc:trigger name="on-change">lcheck</erc:trigger>
</address>
```

~livis/adr/bunny - reloaded

```
<?xml version="1.0" encoding="utf-8"?>
<address xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions" >
  <erc:id>~livis/adr/bunny</erc:id>
  <erc:type>prototype</erc:type>
  <erc:catalog category="/Address" id-ref="~livis/catalog"/>

  <name   erx:field-ref="string" erc:index="~livis/catalog">Easter Bunny</name>
  <street erx:field-ref="string" erc:index="~livis/catalog">Wiese 7</street>
  <zipcode erx:field-ref="int"   erc:index="~livis/catalog"/>12345</zipcode>
  <city    erx:field-ref="string" erc:index="~livis/catalog"/>Waldbröhl</city>
  <phone   erx:field-ref="string">0190 666 666</phone>

  <erc:action name="edit">   /bin/edit   </erc:action>
  <erc:action name="delete"> /bin/rm!wizard </erc:action>
  <erc:action name="copy">   /bin/cp!forEdit </erc:action>
  <erc:action name="check">  ~livis/check.xsl
    <erc:arg name="default">Waldbröhl</erc:arg>
  </erc:action>
  <erc:trigger name="on-change">lcheck</erc:trigger>
</address>
```

~lvis/catalog – damit es komplett ist

```
<?xml version="1.0" encoding="utf-8"?>
<index-table xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
  <erc:type>index</erc:type>
  <erc:id>~lvis/catalog</erc:id>
  <erc:permission role="~any">rx</erc:permission>

  <erc:index><erc:name> category </erc:name></erc:index>
  <erc:index><erc:name> name </erc:name></erc:index>
  <erc:index><erc:name> street </erc:name></erc:index>
  <erc:index><erc:name> zipcode </erc:name>
    <erc:index-type>int</erc:index-type>
  </erc:index>
  <erc:index><erc:name> city </erc:name></erc:index>

  <erc:index><erc:name> RefCount </erc:name>
    <erc:index-type>int</erc:index-type>
    <erc:adder/>
  </erc:index>
</index-table>
```

~livis/check.xsl – damit es komplett ist

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3c.org/1999/XSL/Transform"
  xmlns="http://ercato.com/xmlns/ErcatoCore">
  <xsl:param name="default"/>
  <erc:id>~livis/check.xsl</erc:id>
  <erc:permission role="~any">rx</erc:permission>

  <xsl:template match="//city">
    <xsl:copy>
      <xsl:apply-templates select="@*" />
      <xsl:variable name="zip"><xsl:value-of select="//zipcode"/></xsl:variable>
      <xsl:variable name="zip2"><xsl:value-of select="substring($zip, 1, 2)"/></xsl:variable>
      <xsl:choose>
        <xsl:when test="$zip = ''"/>
        <xsl:when test="$zip2= '10'">Berlin</xsl:when>
        <!-- ..... -->
        <xsl:when test="$zip2= '81'">München</xsl:when>
        <xsl:when test="$zip='99084'">Erfurt</xsl:when>
      </xsl:choose>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@*|node()">
    <xsl:copy><xsl:apply-templates select="@*|node()" /></xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

XML Algebra um Vererbungen auszu-drücken

- Unten sind 3 Ercatons (XML Dokumente mit SVG)
 - Das mittlere Ercaton enthält die Haare und erbt vom rechten
 - Das Ergebnis ist das linke Ercaton

Dies ist wirklich nur mit einem <erc:clone>-tag realisiert worden.



~sample/whiskers-extends-plaintiger = ~sample/whiskers extends ~sample/plaintiger

C o p y r i g h t

b y w w w . l i v i n g p a g e s . d e

XML und Java in einem Ercato:

~livis/count

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<counter xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
  <count>0</count>
  <erc:id>~livis/count</erc:id>
  <erc:object lang="Java">
    <erc:class>com.ercato.lib.shared.Counter</erc:class>
    <erc:archive>~livis/lib/shared.jar</erc:archive>
  </erc:object>
  <erc:action name="main"> lincrement </erc:action>
  <erc:action name="increment">
    <erc:permission role="~livis/roles/friend">bx</erc:permission>
    <erc:arg name="amount">1</erc:arg>
    <erc:native lang="Java">
      <erc:method>increment</erc:method>
      <erc:parameter name="amount" type="int"/>
      <erc:returns type="int"/>
    </erc:native>
  </erc:action>
</counter>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<counter
  xmlns="http://ercato.com/xmlns/ErcatoCore">
  <erc:clone>~livis/count</erc:clone>
  <count>19</count>
  <erc:id>~livis/count2</erc:id>
</counter>
```

XML und Java in einem Ercaton: *~lvis/count*

- Ercaton Aktionen sind Sprachen-unabhängig
- So sieht *~lvis/count!increment* in Java aus:

```
package com.ercato.lib.shared;
import com.ercato.core.*;
import org.w3c.dom.*;

public class Counter extends ErcatonObject implements Action {
    Text counter;

    protected void evaluateElement (EvaluationContext ec, String tag, String nsuri) {
        if ("count".equals (tag)) counter = ec.getTextNode (false);
    }

    public int increment (int amount) {
        int count = Integer.parseInt (counter.getData ());
        count += amount;
        counter.setData (String.valueOf (count)); touch ();
        return count;
    }
}
```

Pro und Contra

Pro:

- Es können viele Zeilen Code gespart werden
- Die Programme sind gut wieder benutzbar
- Unter den richtigen Voraussetzungen kann man schnell programmieren

Contra:

- Flexibilität für andere Anwendung ist ungewiss
- Aufsetzen der Laufzeitumgebung ist viel Programmieraufwand
- Datensätze sind größer, da Text und nicht binäre abgespeichert wird

Es ist ein Ansatz in eine gute Richtung und ist nicht nur graue Theorie, jedoch kann man das volle Potential oder alle Probleme erst abschätzen, wenn es in mehr Bereichen eingesetzt wird.

Teil IV

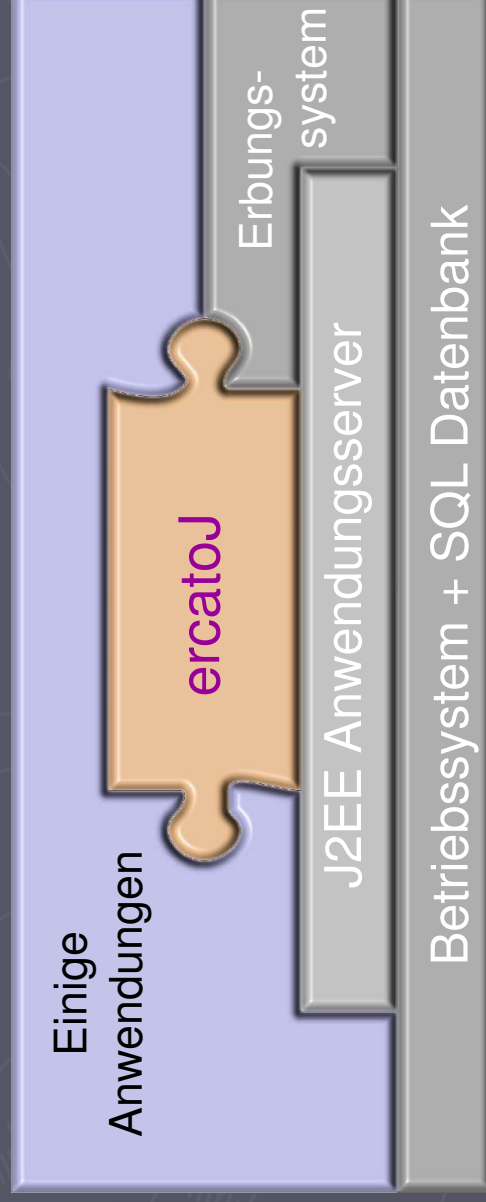
ercatoJ: eine Ercaton Laufzeitumgebung



ercatoJ Übersicht

Als erstes brauchen Ercatons nur eine bestimmte Anzahl von ` < ` -Zeichen zum existieren.
Für mehr feinere Eigenschaft braucht man eine Laufzeitumgebung.
Eine solche Umgebung ist ercatoJ, eine auf J2EE basierende Entwicklung.
Die Version 1.0 läuft im richtigen Einsatz und alle Grundideen sind implementiert.
Die Umgebung ercatoJ ist aus JavaBeans zusammengesetzt und braucht nicht umgestellt werden, wenn sich Ercatons ändern.

Jedes neue Ercaton bedient
sich von der Komplexität von J2EE.



ercatoJ Eigenschaften

1. Ercatons werden auf Enterprise JavaBeans (EJBs) gesetzt.
2. Die starke Algebra von XML sichert die Vererbung.
3. Das Verhalten von Ercatons kann in Java und/oder XSLT geschrieben sein.
4. Es kann mit WebServices und einfachem XML kommuniziert werden.
5. Die Benutzerumgebung ist web basierend.
6. Alles kann ein Ercaton sein, incl. Bilder und binäre Code.
7. Komplexe Datenbankenschemas werden generiert und synchronisiert.

ercatoJ Zusammenfassung

- ▶ Thing-oriented programming ist die Grundlage.
- ▶ Ercatons sind vollkommen spezifiziert
 - durch XML (und Programmiersprachen wie Java oder C#)
 - mit Benutzeroberflächen, Suchfunktionen, ...
 - Unabhängig von den zugrunde liegenden Technologien (SQL,...)
- ▶ ercatoJ ist eine Laufzeitumgebung um mit und zwischen Geschäftslösungen zu arbeiten. Sie ist eine EJB-bassierende J2EE Anwendung.
- ▶ ercartoJ wird mit der Standardbibliothek von Serviceercatons aus geliefert, *erx*.