

Datenspeicherung mit der DDT unter Java

7. Juni 2004

Rechtlicher Hinweis

Dieser Beitrag ist lizenziert unter der UVM Lizenz für die freie Nutzung unveränderter Inhalte.

Zusammenfassung

Relationale Datenbank-Systeme sind langlebige Gesellen. Seit 15 Jahren immer wieder tot gesagt, sind sie immer noch populär - trotz Objekt-orientierter bzw. XML-basierter Datenbanken.

Dazu im Widerspruch stehen der Erfolg der Objektorientierten Entwicklung und XML. Das führt dazu, dass viel Entwicklungsaufwand betrieben wird, um XML- und Objekt-basiertes Daten-Handling auf relationale Datenbanken abzubilden.

Fertige Software-Schichten für die notwendige Abstraktion sind sowohl von kommerziellen Anbietern, als auch von OpenSource-Projekten verfügbar; sie beschränken sich aber meist auf die reine Datenbank- und SQL-Abstraktion. Die DynamicDataTechnology (DDT) des Portals-Systems open-EIS geht mehrere Schritte weiter und unterstützt den Entwickler neben dem Datenbank-Design und -Programmierung auch bei der Front-End-Entwicklung.

Die Firma Community4You veröffentlichte den vollständigen DDT- und open-EIS-Quellcode unter GPL; der Einsatz ist für nicht-kommerzielle Projekte und kleine kommerzielle Projekte frei von Lizenz-Gebühren.

Welche Funktionen bietet DDT?

- Zugriff und Suche auf Daten über Vererbung bestehender Klassen analog zum Zugriff auf relationalen Datenbanken ohne gesondertes SQL.
- Einsortierung der Daten in Baumstrukturen.
- Transparente Lokalisierung von Daten.
- Einschränkung des Zugriffs auf Daten über Benutzer-Rechte und -Rollen und durch Lokalisierung.
- Verknüpfung von Daten 'on-the-fly'
- Pflege der Daten über web-basiertes, automatisch erzeugtes Pflege-Interface.
- vollständige Replikationsfähigkeit von Inhalten und Datenbankstrukturen

1 Datenspeicherung mit der DDT unter Java

1.1 Hintergründe

Bevor wir darauf eingehen, wie man die DDT für eigene Projekte einsetzt, werfen wir einen Blick auf die Probleme, die zur DDT geführt haben.

1.1.1 Ein Wissensmanagement-System entsteht

Seit mehr als 3 Jahren arbeitet die Firma Community4you an ihrem Wissensmanagement-System 'open-EIS'. Das Haupt-Problem für ein solches System oder allgemein bei der Verwaltung beliebiger Informationen ist ihre Speicherung und der Zugriff darauf.

Warum soll das ein Problem sein? Informationen haben eine Struktur - können also auch automatisiert verarbeitet werden. So lautet die einfache Antwort. Sie hat nur einen Schönheitsfehler: meist erfassen wir die Struktur zu allgemein oder sie ist zu speziell. In beiden Fällen entstehen Anwendungen, die immer nur für spezielle Anwendungsfälle geeignet sind; und häufig Middleware erfordern, wenn darum geht, dass zwei Anwendungen zusammen arbeiten sollen, die Wissen speichern.

Warum erfassen wir die Struktur zu genau bzw. zu allgemein? Weil wir, die Programmierer, faul sind! Warum sollten wir sonst intensiv mit etwas arbeiten, das uns Arbeit abnimmt?

Was spricht dagegen die Strukturen von Informationen genauso zu erfassen, wie sie sind? Weil es ganz unterschiedliche Arten von Information gibt - und für jede einzelne die notwendigen Datenbank- und Programmier-Arbeiten zu machen Zeit und Geld kostet.

Deshalb beschränken sich viele Systeme darauf nur ein bestimmtes Fach-Gebiet abzudecken.

Das führt dann zu Situationen wie dieser: die Buchhaltung kann eben nicht mal schnell eine elektronische Telefon-Notiz über einen Kunden an den Vertrieb schicken, weil sie verschiedene System benutzen, die zwar beide Telefon-Notizen verwalten, aber in verschiedenen Strukturen organisieren.

Soviel zum Problem, was ist die Lösung - bzw. warum ist die DDT die Lösung?

Eigentlich ist unser Problem weniger, dass es so viele Arten von Informationen gibt, vielmehr sollte der Einsatz neuer Strukturen nur einfach und schnell gehen, auch bei laufendem Betrieb. Dann könnten wir quasi am Anfang einen Basis-Satz an Strukturen definieren, und wenn es notwendig wird, Strukturen anpassen und neue hinzufügen.

Und genau das bietet die DDT: beliebige Daten-Strukturen für jede Art von Medien zu verwalten.

Aber das ist für das Wissensmanagement nur die halbe Miete: auch wenn wir in der Lage sind Information einzugeben und zu durchsuchen, müssen wir sie auch verknüpfen können. Ein System, das zwar Kunden, Termine und Meeting-Protokolle verwaltet ist zwar praktisch, aber es wird unbequem, wenn der Anwender diese Informationen nicht zusammenführen kann.

Bedingt durch die Technik werden solche Verknüpfungen häufig 'hart verdrahtet': ein Termin kann auf einen Kunden verweisen, aber nicht auf das Protokoll, weil es der Programmierer nicht vorgesehen hat. Wehe, diese Funktion soll ergänzt werden - spätestens dann wird ein Programm-Update fällig.

Mit der DDT sind diese Verknüpfungen kein Problem, unabhängig davon, welche Strukturen verknüpft werden sollen.

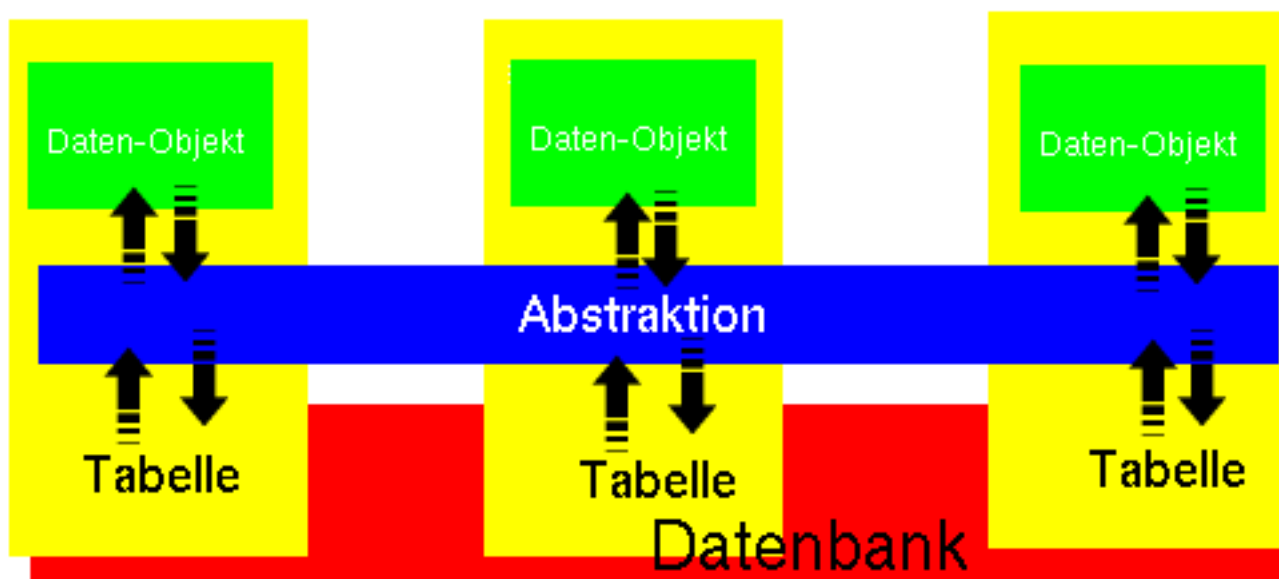
Ein drittes Problem hängt eng mit relationalen Datenbanken zusammen: es ist mit ihnen nicht direkt möglich Daten hierarchisch zu speichern. Datensätze frei zu kategorisieren ist aber im Wissensmanagement und nicht nur dort unumgänglich. Diese Fähigkeit muß für jede Struktur explizit implementiert werden; in der Datenbank wie im Programm selbst.

1.1.2 Wie funktioniert das?

Im Prinzip ist die DDT eine Datenbank-Abstraktion, tatsächlich ist dieser Begriff aber viel zu einschränkend.

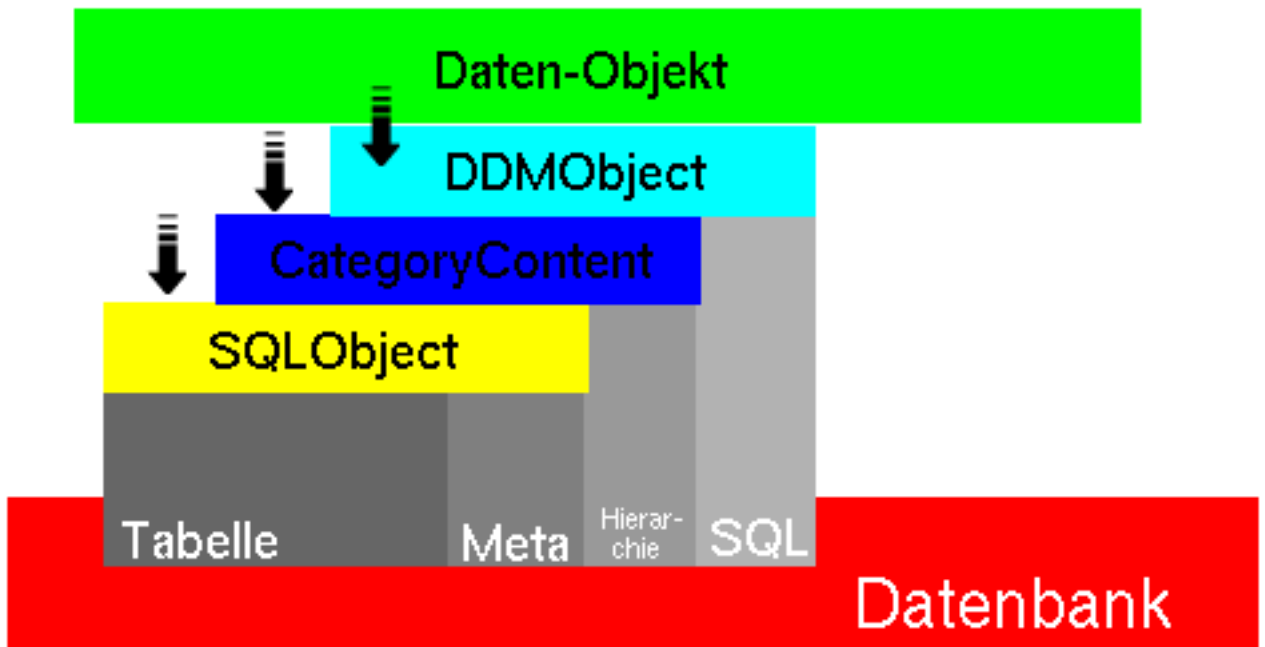
Datenbank-Abstraktion bedeutete ursprünglich nur unabhängig von einem bestimmten Datenbank-Management-Systems eines Herstellers; Beispiele dafür sind ODBC oder JDBC.

Im weiteren Sinne kapseln Datenbank-Abstraktionen relationale Datenbanken in Objekten und erlauben den Zugriff auf diese in Objekt-orientierter Form und bieten Ansätze SQL-Abfragen (teilweise) automatisch zu generieren. Beispiele dafür sind das Apache Turbine Projekt oder DB.DataObject in PHPs PEAR.



In der Grafik sieht man, dass einfach eine Schicht auf eine andere aufsetzt - ohne das tatsächlich der Funktions-Umfang zunimmt.

Aber auch letztere nehmen uns nicht die eigentliche Denkarbeit ab über das Datenbank-Design. Ob mit oder ohne: sie erfüllen nicht unsere obigen Forderung nach möglichst einfacher Erstellung neuer Strukturen. Hier kommt die DDT ins Spiel.



Schlüsseln wir die Grafik auf: Die Daten-Abstraktion besteht aus drei Schichten. Die unterste Schicht kapselt den Zugriff auf die eigentliche Tabelle und gewährleistet die Datenbank-Unabhängigkeit. Hier können wir auf die Daten in Form von Listen bzw. konkret auf einen einzelnen Datensatz zugreifen.

Die mittlere Schicht kombiniert den Zugriff auf diese Daten mit frei wählbaren Kategorien. Wir können also Datensätze anhand ihrer Kategorie einschränken.

In der obersten Schicht kommt die Fähigkeit hinzu, die notwendigen SQL-Statements anhand der Strukturbeschreibung selbstständig zu erzeugen.

Ein Daten-Objekt kann auf jede dieser Schichten aufsetzen, je nachdem, wie es die Situation erfordert.

In der Grafik finden Sie die Begriffe „Meta“ und „SQL“ beide stehen eigentlich für das gleiche: Meta-Daten über eine Daten-Struktur in der Datenbank. Diese Meta-Daten umfassen:

- Informationen zur Tabelle und ihrer Struktur
- Informationen zur Darstellung der Daten
- Dokumentation zur Struktur

Diese Meta-Daten gehen weit darüber hinaus, was Sie an Informationen über eine Tabelle vom DBMS selbst erhalten. Die Meta-Daten gewährleisten es, den Inhalt der Tabelle und die Tabellenstruktur vollständig zu replizieren - echt Datenbank-unabhängig.

1.2 Wie erstelle ich die Tabellen und die Metadaten?

Wie jede andere höhere Datenbank-Abstraktion definiert die DDT eigene Mechanismen zur Beschreibung von Datenstrukturen, während aber viele andere auf XML setzten, setzt die DDT auf eine SQL-artige Sprache, deren Befehle mit „echtem“ SQL kombiniert werden können.

Es gibt eine Reihe guter Gründe SQL weiter zu entwickeln, statt XML auf zu setzen:

- es ist kein Wechsel zwischen XML für die Struktur-Definition und SQL für das Einpflegen von Daten erforderlich
- es existiert kein offener und verbindlicher Standard für die Definition von Datenbank-Strukturen mit XML
- XML ist geschwätzig und erfordert mehr Tipparbeit
- SQL ist im Datenbank-Umfeld vertrauter als XML ;-)

Der DDT eigene SQL-Dialekt nennt sich Meta-SQL.

1.2.1 Begrifflichkeiten

Bevor wir einen Blick auf Meta-SQL werfen, müssen wir kurz einige Begriffe erklären.

Ein **Template** ist die Beschreibung einer Datenstruktur, vergleichbar mit einer Datenbank-Tabelle - ein Template wird in der Datenbank durch eine Tabelle oder mehrere abgebildet.

Eine **Komponente** ist ein atomares Element in einem Template und entspricht immer einer Spalte in einer Datenbank-Tabelle.

Ein **Dokument** enthält Informationen, die nach den Vorgaben eines Templates oder mehrerer aufgebaut sind. Es ist vergleichbar einem Datensatz in einer Datenbank-Tabelle. Ein Dokument kann aber auf mehreren Datensätzen in verschiedenen Tabellen basieren.

1.2.2 Meta-SQL

Die Funktion von Meta-SQL ist die Beschreibung von Templates und ihrer Komponenten.

Ein Beispiel:

```
/**
<template name="c4u_wbt_lesson" package="c4u_wbt" >
  <intro>Template to store lessons.</intro>
  <doc>
    This template is used to store lessons.
  </doc>
  <component name="headline">
    <doc>
      The headline of the lesson.
    </doc>
  </component>
  <component name="pc">
    <doc>Position number: Lessons are ascending stored by this number.
    </doc>
  </component>
  <component name="credits">
    <doc>
    </doc>
  </component>
  <component name="intro">
    <doc>
      The introduction of the lesson. This can be a short text which does briefly describe
    </doc>
  </component>
  <component name="media">
    <doc>
      Media attachments.
    </doc>
  </component>
</template>
```

```

        </doc>
    </component>
    <component name="txt">
        <doc>
            The text of the lesson.
        </doc>
    </component>
    <component name="extern_text">
        <doc>
            Additional text for external usage. (e.g. display in the internet)
        </doc>
    </component>
    <component name="intern_text">
        <doc>
            Additional text for internal usage only.(e.g. display in the intranet)
        </doc>
    </component>
</template>
*/

```

```
CREATE standard TABLE
```

```
c4u_wbt_lesson
```

```

LOGGED
FLAGS c4u_guid, c4u_status, c4u_author, c4u_lastchange, c4u_searchable
(

```

```
headline
```

```

varchar(255) c4u_standard
    NOT NULL
    LANGDEP
    FLAGS c4u_listable, c4u_orderable, c4u_searchable, c4u_showletters
    GENERALTYPE c4u_descr,

```

```
pc
```

```

int c4u_standard
    NOT NULL
    FLAGS c4u_listable, c4u_orderable, c4u_searchable
    GENERALTYPE c4u_pc,

```

```
credits
```

```

int c4u_standard
    FLAGS c4u_listable, c4u_orderable, c4u_searchable,

```

```
intro
```

```

lvarchar c4u_standard
    LANGDEP
    FLAGS c4u_searchable
    GENERALTYPE c4u_intro,

media

notype c4u_genmedia
    NUMBERED
    LANGDEP
    GENERALTYPE c4u_media,

txt

notype c4u_textl
    LANGDEP
    GENERALTYPE c4u_text,

extern

-

text

notype c4u_textl
    NUMBERED
    LANGDEP
    GENERALTYPE c4u_externtext,

intern

-

text

notype c4u_textl
    NUMBERED
    LANGDEP
    GENERALTYPE c4u_interntext
);

```

Unser Text besteht aus zwei Teilen: am Anfang finden wir die (optionale) Dokumentation des Templates, danach folgte dessen Definition.

Der Name des Templates ist `c4u_wbt_lesson`, seine Komponenten sind u.a. `headline`, `pc`, `intro`, `media` usw. Die Definition umfasst alle Eigenschaften des Templates und die Eigenschaften der Komponenten.

Für das Template bestimmen wir über das Schlüsselwort `LOGGED`, dass alle Änderungen an einem Dokument dieses Templates DDT intern vermerkt werden sollen. Die Template-Flags bestimmen zusätzliche Eigenschaften eines Dokumentes - aus Programmierer-Sicht sind es vordefinierte Komponenten.

Gehen wir zu den Komponenten: betrachten wir mehrere Schlüsselwörter anhand von `headline`. Als erstes wird der SQL-Datentyp definiert, der Wert `c4u_standard` hat Einfluß auf die Umsetzung des Templates in reale Datenbank-Tabellen. Das Schlüsselwort `NOT NULL` besagt, dass diese Komponente einen Wert haben muß - es kann optional auch ein Standardwert vorgegeben werden.

Eine besondere Fähigkeit der DDT ist die integrierte Mehrsprachigkeit von Dokumenten. Eine Komponente, die je nach Sprache einen anderen Inhalt besitzt, wird durch LANGDEP markiert. Die nachfolgenden Flags bestimmen die Anzeige-Eigenschaften dieser Komponente.

Über den GENERALTYPE einer Komponente ist eine automatische Konvertierung von Dokumente zwischen verschiedenen Templates möglich.

Unter anderem bei der Komponente media wird das Schlüsselwort NUMBERED benutzt. Damit erlauben wir, dass diese Komponente nicht nur ein Element aufnehmen kann, sondern unbegrenzt viele.

Alle diese Informationen über das Template werden in der Datenbank gespeichert. Die dafür notwendigen Tabellen sind selbst in Form von Templates definiert. Templates sind im Sinne der DDT damit auch nur Dokumente vom Template-Typ 'Template'.

1.3 Die Programmierung

1.3.1 Ein Daten-Objekt erzeugen

Implementieren wir ein Objekt, das ein Buch-Dokument kapselt.

```
public class Buch extends DDMObject {

    /**
     * Constructor
     */
    public Buch() {
        super();
    }

    /**
     * Get the title of the book
     * @return the title of the book
     */
    public String getTitel() {
        return this.getString("titel");
    }
    // usw.
    ...
}
```

Wir implementieren die üblichen Getter/Setter-Methoden, die notwendigen Methoden für die eigentliche Definition der Abfrage werden durch DDMObject bereitgestellt, von dem wir unsere Klasse ableiten.

Eins fällt auf: wir geben nicht an, auf welche Tabellen bzw. Datenstrukturen wir uns eigentlich beziehen! Obwohl das eigentlich typisch wäre für Datenbank-Abstraktionen. Hier kommt die DDT ins Spiel - unsere Klasse kapselt ein Dokument. Die Struktur eines Dokumentes wird durch ein Template beschrieben. Durch welches Template können wir zu Laufzeit bestimmen. Damit ist unsere Buch-Klasse äusserst flexibel und nicht an ein bestimmtes Datenbank-Design gebunden.

1.3.2 Der Daten-Zugriff

Für den Zugriff auf unsere Daten müssen wir unsere Buch-Klasse mit einem Template verknüpfen.

```
// Code für DB-Verbindungsaufbau
...
```

```

Buch buch = new Buch();

// Sprache setzen
buch.setLanguage(language);

// Template und SQL zuweisen
Template t=templatecache.getTemplateFromName
    ("template_buch",language.getLangid());
Component c=templatecache.getComponentFromName
    ("template_buch",language.getLangid());
DDMPool d=templatecache.getDDMPoolFromName("template_buch")
buch.initPool(t,c,templateflag,d);

// Buch-Dokument mit dem Identifier 12 holen
buch.setId("12");

// Titel ausgeben
out.println(buch.getTitel());

```

Nach dem Erzeugen eines Buch-Objektes weisen wir ihm eine Sprache zu, alle folgenden Datenzugriffe beziehen sich auf Dokumente bzw. Inhalte der eingestellten Sprache.

Im zweiten Schritt erzeugen wir aus der Template-Definition die notwendigen Struktur-Beschreibungen und SQL-Statements. Diese Information weisen wir dem Buch-Objekte zu; es „weiß“ damit wie auf die eigentlichen Daten zugegriffen werden soll.

Dann führen wir eine Abfrage durch: wir fragen nach dem Dokument mit dem Identifier 12. Achtung, dieser Identifier bezieht sich auf ein Dokument, nicht etwa auf einen Primary Key in einer Datenbank-Tabelle. Der Identifier ist sprachunabhängig.

Das DDMObject von dem unsere Klasse abgeleitet ist, bietet alle notwendigen Methoden um ein Dokument zu erzeugen, Inhalte zu ändern, zu löschen und um nach Dokumente zu suchen.

1.3.3 Einbindung in JSP-Seiten

Die DDT bzw. open-EIS wurden von Beginn an als Web-Anwendung konzipiert; sie bietet deshalb einen großen Satz an JSP-Tags zur einfachen Verwendungen von Daten-Objekte in Webseiten.

Hier der vollständige Quellcode einer JSP-Seite, die eine Liste aller Bücher einer vorgewählten Kategorie zeigt. Die Seite zeigt standardmäßig zwanzig Bücher an, und bietet die Navigation für den Zugriff auf die nächsten zwanzig Bücher bzw. die vorherigen. Es kann nach verschiedenen Komponenten sortiert werden, z.B. nach Buch-Titel oder dem Buch-Autor. Ebenfalls integriert ist eine Anwahl der Bücher nach dem Anfangsbuchstaben des aktuell ausgewählten Sortierkriteriums. Und natürlich bietet sie die Links für die Detail-Ansicht eines Dokumentes und für dessen Bearbeitung.

```

<%@ include file="include.jsp" %>

<core:inithistory name="bookhistory" type="view"/>

<core:initsqlobject name="books" classname="Buch" type="view"
    template="template_buch" history="bookhistory"/>

<core:header/>
<core:body/>

<core:admin_form sqlobject="books" history="bookhistory"
    listpage="buch/view.jsp" editpage="buch/buch_dlg.jsp"
    showpage="buch/show.jsp" module="c4u.buch">

```

```

<core:menu>
<core:menuentry type="new" text="c4u.buch.buch.view.newbuch"/>
<core:menuentry type="notification"/>
<core:menuentry type="showletters"/>
<core:menuentry type="import"/>
<core:menuentry type="export"/>
</core:menu>

<br>
<core:title>
  <td>
    <core:modul/>
  </td>
  <td align="right" nowrap>
    <core:font/>
    <core:mapselect form="adminform" templateid="c4u_buch_buch"/>
  </font>
  </td>
</core:title>

<core:admin_resultheader>

<core:admin_resultdata name="select"/>
<core:admin_resultdata name="status" type="sortlink" value="gelesen"/>
<core:admin_resultdata name="id" type="sortlink"/>
<core:admin_resultdata name="title" type="sortlink"
  value="<%=language.getText(\ "c4u.buch.buch.view.title\ ")%"/>
<core:admin_resultdata name="autor" type="sortlink"
  value="<%=language.getText(\ "c4u.buch.buch.view.autor\ ")%"/>
<core:admin_resultdata name="cat_id" type="sortlink"/>
<core:admin_resultdata name="last_change" type="sortlink"/>
<core:admin_resultdata name="edit" type="editlink"/>

</core:admin_resultheader>

<core:admin_resultbody>

<!-- the body columns -->
<core:admin_resultdata name="select"/>
<core:admin_resultdata name="status"/>
<core:admin_resultdata name="id" type="showlink"/>
<core:admin_resultdata name="title" type="sortlink"
  value="<%=HTMLConv.replaceLinks(HTMLConv.HTMLSpecialChars(StringTool.cut(books.getStringing
<core:admin_resultdata name="autor" type="sortlink"
  value="<%=HTMLConv.replaceLinks(HTMLConv.HTMLSpecialChars(StringTool.cut(books.getStringing
<core:admin_resultdata name="cat_id" type="categorylink"/>
<core:admin_resultdata name="last_change"/>
<core:admin_resultdata name="edit" type="editlink"/>

</core:admin_resultbody>

</core:admin_form>

```

```
</font>
</body>
</html>
```

Der Quellcode für Seiten für die Detail-Ansicht und zum Bearbeiten eines Dokumentes sieht fast genauso aus. Einzig die Seite zum Speichern von Änderungen unterscheidet sich:

```
<%@ include file="include.jsp" %>

<core:inithistory name="bookhistory" type="controller"/>
<core:initsqlobject name="book" classname="Buch" type="controller"/>

<core:admin_form sqlobject="book" history="bookhistory"
  listpage="buch/view.jsp" editpage="buch/buch_dlg.jsp"
  showpage="buch/show.jsp" module="c4u.buch" type="controller">

<core:admin_dialoghandler type="<%=DialogTag.CANCEL_BUTTON%"/>
<core:admin_dialoghandler type="<%=DialogTag.DELETE_BUTTON%"/>
<core:admin_dialoghandler>
<%
  if (bookhistory.getCurrentPage().equals("c4u.buch.buch.dlg.common_tab"))
  {
    %>
    <core:admin_resultdata name="titel"/>
    <core:admin_resultdata name="autor"/>
    <core:admin_resultdata name="intro"/>
    <%
  }

  book.setObject("guid",request.getParameter("guid"));
  book.setObject("status",request.getParameter("status"));
  book.setObject("l_status",request.getParameter("l_status"));
%>
</core:admin_dialoghandler>
<core:admin_dialoghandler type="<%=DialogTag.APPLY_BUTTON%"/>
</core:admin_form>
```

Auch wenn es nicht so aussieht, obiger Code speichert alle Änderungen an einem Dokument dauerhaft.

Im Prinzip können wir sogar auf spezielle JSP-Seiten verzichten, die DDT bietet den sogenannten DDM-Modus für JSP-Seiten, der die Informationen für die Darstellung aus der Definition des Templates holt.

1.4 Abschluß

Dieser Text gibt nur einen kurzen Überblick darüber, wie gering der Aufwand ist mit der DDT neue Datenbank-basierte Datenstrukturen und dazugehörige webbasierte Pflege-Oberflächen zu erzeugen.

Die Funktionen der DDT noch einmal im Überblick:

- einfache Erstellung neuer Datenstrukturen einschließlich Informationen für ihre Darstellung
- schnelle Erstellung von Pflege-Oberfläche, die schrittweise verfeinert und angepasst werden können
- transparente Mehrsprachigkeit
- Unterstützung von Kategorien/Hierarchien

- SQL wird automatisch erzeugt, automatisch erzeugtes SQL kann manuell optimiert werden.
- Sämtliche Datenstrukturen und Inhalte können repliziert werden - ohne Verluste oder Einschränkungen.

Weiter Informationen erhalten Sie unter <http://www.open-eis.com>