@enterprise 6.1 - DMS

Dokumentenmanagementsystem
Konfiguration und Programmierung

Robert Eisner

November 2003

Groiss Informatics GmbH

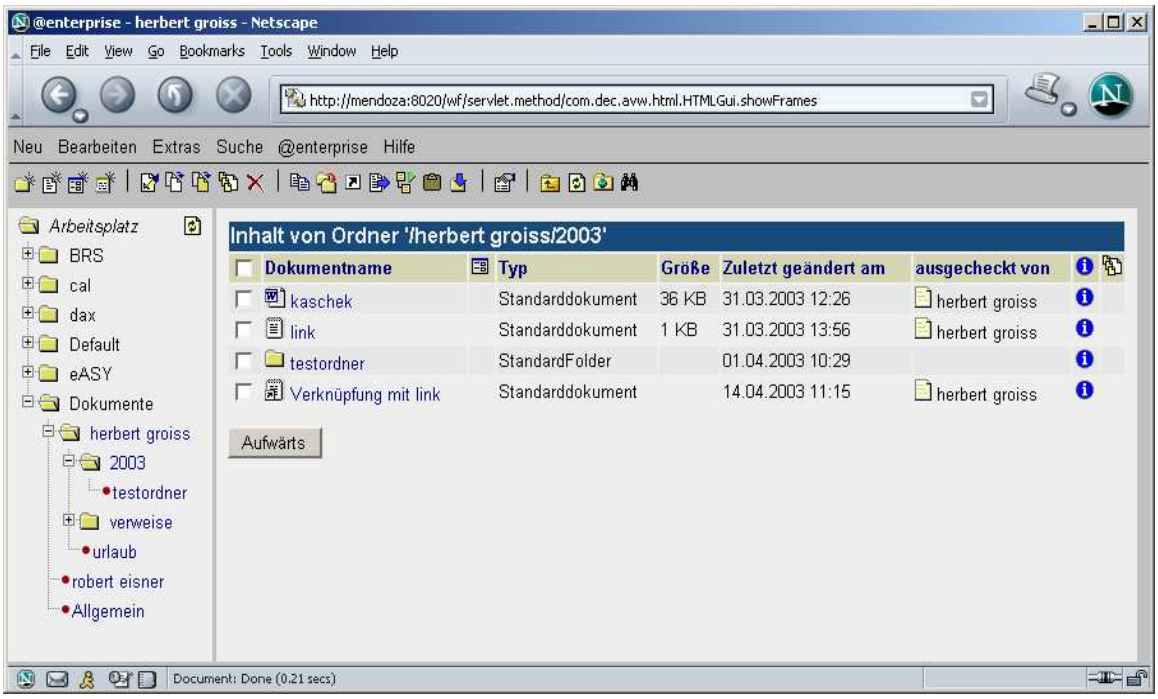
@enterprise – Kursunterlagen Dokumentenmanagement
November 2003

Inhalt

Teil 1: Key-Features

- Typisierte Dokumente (und Ordner)
- Flexible Speicherung der Dokumente
- Verwaltung der Metadaten in einer SQL-Datenbank
- Flexible Rechtevergabe aufgrund von Rollen und Organisationsstrukturen
- Verwendung von WebDAV
- Verknüpfung mit Workflow-Komponente
- Flexible Anpassungsmöglichkeiten durch definierte Schnittstellen

Integration im Client

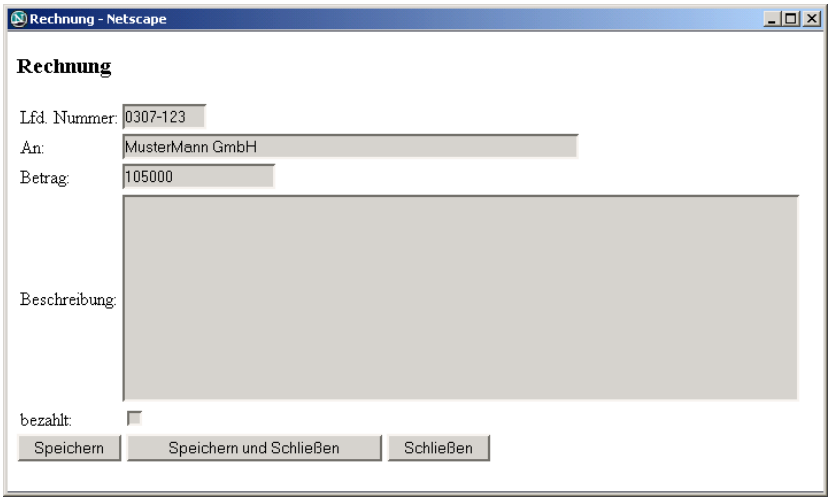


Typisierte Dokumente/Ordner

- Dokumente/Ordner gehören einem bestimmten Typ an (z.B. Urlaubsantrag)
- Jeder Typ verfügt über eine Reihe von Metadaten
- Zu jedem Typ kann eine Vorlage definiert werden
- Anlegen eines Dokumenten-/Ordnertyps erfolgt wie die Definition eines Formulars
- 3 systemdefinierte Typen:
  - Standardordner: verfügt über keine zusätzlichen Metadaten
  - Standarddokument: verfügt über keine zusätzlichen Metadaten
  - Notizen:
    - können eigenständig sein oder an andere Dokumente angehängt werden
    - angehängte Notizen können als privat oder öffentlich gekennzeichnet werden

Typspezifische Metadaten

- z.B. Dokumententyp ‚Rechnung‘:



- Typspezifische Metadaten werden in einer entsprechenden Datenbanktabelle abgelegt

Flexible Speicherung

- Speicherung eines Dokuments ist unabhängig von der Verwaltung der Metadaten
- Erfolgt über eine wohldefinierte Schnittstelle die den Austausch dieser Komponente ermöglicht
- Dadurch können externe Dokumentenmanagementsysteme integriert werden
- Standardimplementierung: Dokumente werden in der Datenbank des @enterprise-Servers abgelegt

Flexible Rechtevergabe

- Dokumente unterliegen dem Berechtigungssystem von @enterprise
- Rechte zum Erzeugen von neuen sowie zum Bearbeiten oder Lesen bestehender Dokumente können flexibel vergeben werden
- Rechte können an bestimmte Benutzer, Rollen, Abteilungen oder Arbeitsgruppen vergeben werden
- Rechte können auch auf Ordner vergeben werden. Diese beziehen sich nur auf den Ordner und nicht auf die darin enthaltenen Dokumente (vgl. Unix-Permissions)

Versionierung

- Dokumente können versioniert werden
  - Automatisch bei jeder Änderung
  - Automatisch nach jedem Bearbeiterwechsel
  - ‚on demand' durch den Benutzer
- Versionen verwenden denselben Speichermechanismus wie deren Dokumente

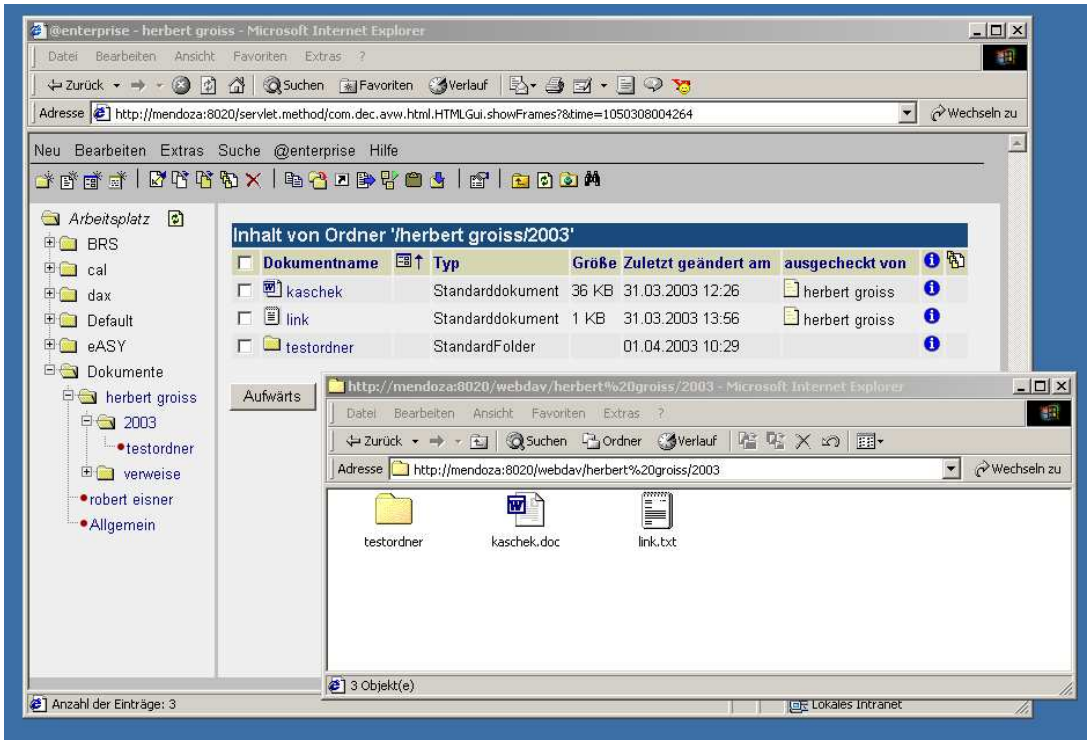
Suche

- Suchmöglichkeiten:
  - eingeschränkt nach allgemeinen Eigenschaften
    z.B. Name oder Änderungsdatum
  - eingeschränkt auf Metadatenfelder
    z.B. nach allen noch unbezahlten Rechnungen
  - Volltext: in Dokumenteninhalt als auch in den Metadaten (muss von der verwendeten Datenbank unterstützt werden)
- Berechtigungen werden berücksichtigt

WebDAV

- Browser Oberfläche
  - keine Client Installationen nötig
  - keine Plugins
- Komfortable Dokumentenbearbeitung mit WebDAV
  - WebDAV ist eine Erweiterung von HTTP (rfc2518)
  - Bearbeiten durch Doppelklick
  - Drag and Drop am Windows Desktop, Navigation im Windows Explorer
  - Kopieren, Verschieben von Ordnerstrukturen möglich.

WebDAV (2)

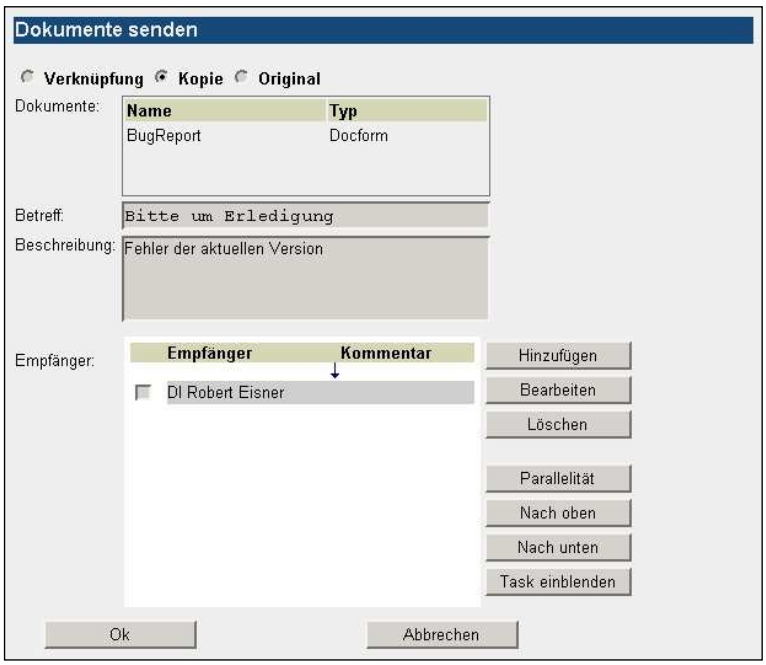


DMS und Workflow

- alle Kombinationen zwischen ad Hoc und voll strukturiertem Workflow möglich
- Unterstützt verschiedene Arbeitsformen:
  - Dokument wird ad hoc an Empfänger versandt
    - Notizen oder weitere Dokumente hinzufügen
    - senden an weitere Bearbeiter
    - zurücksenden zu einem früheren Akteur
    - Ablegen eines Verweises zu dem Prozess
    - ...
  - definierter Workflow mit Dokumenten
  - definierter Workflow mit definierten Formularen

DMS und Workflow – Ad Hoc Workflow

- Frei definierbarer Workflow, z.B. zum Versenden von Dokumenten



Anpassungsmöglichkeiten

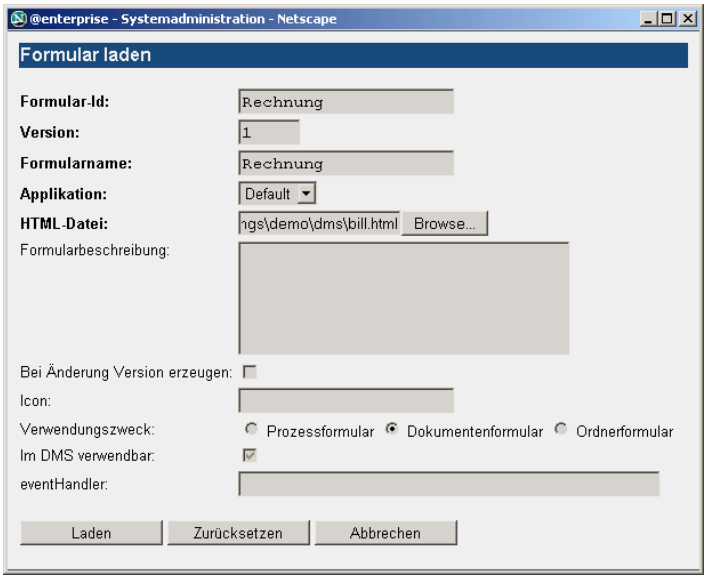
- Durch definierte Schnittstellen, z.B. für folgende Bereiche:
  - Speicherung der Dokumente
  - Archivierung der Dokumente
  - Integration eigener Dokument-/Ordnertypen
  - ...

Teil 2:
Administration - Baumkonfiguration

- Standard-Konfiguration enthält folgende Zeile in der Konfiguration des Hauptbaums:
  - <Node name="Dokumente " class="com.groiss.dms.html.DMSNode"/>
- wird ersetzt durch einen Einstiegspunkt ins DMS, der zumindest folgende zwei Unterordner hat:
  - Wurzel für den DMS-Baum des angemeldeten Benutzers
  - Wurzel für den öffentlichen DMS-Baum (‚Allgemein')
- Weiters gibt es noch für jeden vertretenen Benutzer die Wurzel seines DMS-Baums

Laden neuer Dokument/Ordnertypen

- Erfolgt über die Administrationskomponente



- erzeugt Klassen, deren Code angepasst werden kann

Ordnerkonfiguration

- Einfache Anpassungen des Ordners ohne zu programmieren
- Möglich sind:
  - Namensattribut
  - Icon für den Ordner
  - Art und Reihenfolge der anzuzeigenden Spalten des Ordnerinhaltes
    - systemdefinierte Spalten
    - Spalten, die Werte aus dem Metadatenformular enthalten. Syntax hierbei:
      form.<Formularfeldname>, z.B. form.paid
- Spaltenkonfiguration für den Typ (betrifft dann jeden Ordner dieses Typs) oder auch für einzelne Ordner möglich

Ordnerkonfiguration (2)

- Beispiel: eine zusätzliche Spalte mit dem Wert des Feldes ‚paid' des Metaformulars wird hinzugefügt:

Rechte

- Folgende Rechte können standardmäßig für Dokumente und Ordner vergeben werden:
  - Objekt erzeugen: macht nur Sinn bei Anwendung auf Klasse
  - Objekt bearbeiten: Rechteinhaber darf das Objekt bearbeiten. Im DMS hat der Rechteinhaber damit auch automatisch das Recht ‚Objekte ansehen'.
  - Objekt ansehen: Rechteinhaber hat das Recht das Objekt anzusehen, aber nicht es zu verändern
- Erzeuger (auch Owner genannt) eines Dokuments/Ordners hat implizit die Rechte ‚Objekt bearbeiten', ‚Objekt ansehen' und ‚Berechtigungen bearbeiten'
- Rechte können für einzelne Dokumente oder aber auch für alle Dokumente einer Klasse vergeben werden
- Verwendung von ACLs ermöglicht komfortable Vergabe ein oder mehrerer Rechte an ein oder mehrere Benutzer oder Rollen

Wozu brauche ich welche Rechte?

Aktion	nötiges Recht
Objekt erzeugen	create(docclass)
Objekt ändern/ersetzen, Metadaten ändern,	edit(o)
Objekt löschen / Folder mit Inhalt löschen	edit(o) / edit(folder), wenn o ein Folder: edit für alle Inhalte
Objekt ansehen / Folderinhalt auflisten	view(o) / view(folder)
Objekt verschieben	edit(source-folder), edit(destination-folder)
Link zu Objekt erzeugen	edit(folder)
Objekt kopieren	edit(destination-folder), view(o) wenn o ein Folder: view für alle Inhalte
Objekt umbenennen	edit(o)
Rechte auf Objekt ändern (Zugriff)	edit_acl(o)
Version erzeugen	edit(o)
Version ansehen	view(o)
Version löschen	edit(o)
Eigenschaften ansehen	kein Recht nötig

Wozu brauche ich welche Rechte? (2)

- Erklärung zur Tabelle:
  - o: bezeichnet das Objekt, das bearbeitet wird
  - docclass: die Klasse des Objekts
  - folder, source-folder: der Ordner, in dem das Objekt liegt
  - dest-folder: der Ordner, in den das Objekt kopiert, verschoben oder gelinkt werden soll
- Prozess-Objekte:
  - aktueller Akteur: hat auf alle Ordner und Dokumente des Prozesses die Rechte ‚view', ‚edit' und ‚edit_acl'
  - frühere Akteure: haben ‚view'-Recht auf alle Ordner und Dokumente
- Suche: ich kann alle Objekte finden, auf die ich view oder edit Recht habe.

OE-Kontext

- Ordner und Prozesse haben einen OE-Kontext
- Dokumente erben den OE-Kontext ihrer Ordner / ihres Prozesses
- Berechtigungen können über OE ausgewertet werden

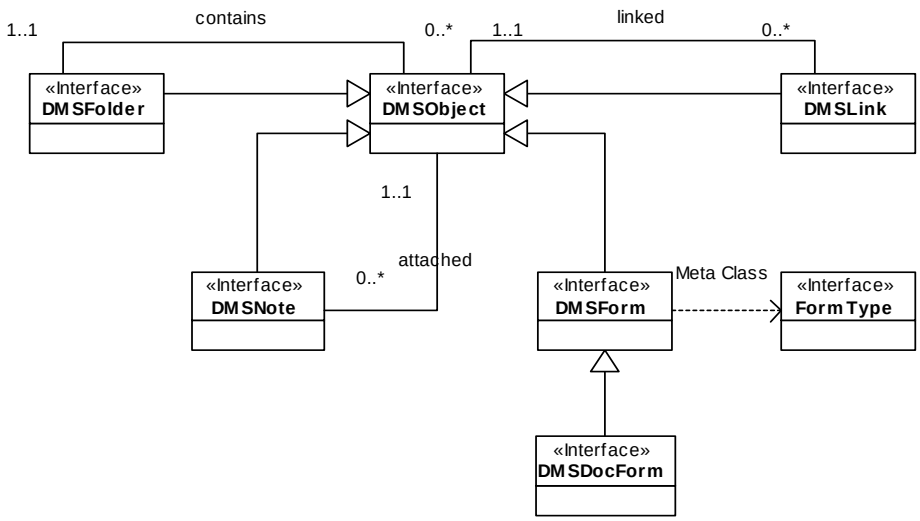
  Beispiel:
  - hugo hat das Recht view in der OE Abt.13.
  - Der Ordner Projekte123 gehört zur OE Abt.13.
  - Somit kann hugo alle Dokumente in Projekte123 ansehen.

DMS-spezifische Systemkonfiguration

Eigenschaft	Bedeutung
Dokumenten-Verzeichnis	Zugriffspfad des Servers auf das Dokumentenverzeichnis
File-Service vorhanden	Gibt an, dass die Clients Zugriff zum Dokumentenverzeichnis haben
Verzeichnis des File-Service	Zugriffspfad des Clients auf das Dokumentenverzeichnis (nur sinnvoll, wenn ‚File-Service vorhanden' angeklickt ist
Erweiterung anzeigen	Wenn gesetzt wird an diversen Stellen beim Namen eines Dokuments auch dessen Erweiterung (z.B. doc) angezeigt
Standard-Tabellenmodell	voll qualifizierter Name der Klasse, die zur Darstellung der Ordnerinhalte verwendet wird. Wenn leer ➔ Standardimpl.
Version bei Bearbeiterwechsel	erzeugt Version, wenn Dokument durch anderen Benutzer bearbeitet wird
ACL vererben	ACL wird an untergeordnete Dokumente und Ordner vererbt
DMS-Speicherklasse	voll qualifizierter Name der Klasse, die für das Speichern der Dokumenteninhalte zuständig ist. Wenn leer ➔ Standardimplementierung, die Inhalte in DB ablegt
DMS-Archivierungsklasse	voll qualifizierter Name der Klasse, die für das Archivieren der Dokumente zuständig ist. Wenn leer ➔ leere Standardimpl.

Teil 3:
Programmierung - Schema

@enterprise: DMS



DMS Interfaces

- DMSObject
  - Basisinterface für alle Objekte, die im DMS verwaltet werden
  - definiert die elementarsten Methoden, z.B. :
    - `getName()`
    - `getCreatedBy()`
    - `getChangedAt()`
- DMSFolder
  - Dient zur hierarchischen Strukturierung der Objekte des DMS
  - definiert dazu notwendige Methoden, wie z.B.:
    - `add(DMSObject obj)`
    - `remove(DMSObject obj)`
    - `listContents()`

DMS Interfaces (2)

- DMSForm
  - Basisinterface für alle Objekte des DMS, die strukturierte Daten enthalten
  - enthält entsprechende Methoden, um auf diese Daten zuzugreifen:
    - `getField(String name)`
    - `setField(String name, Object value)`
- FormType
  - ist eine Beschreibungsklasse für `DMSForm`
  - liefert Informationen über den Typ und die Klasse, die `DMSForm` implementiert:
    - `getClassName()`
    - `getType()`

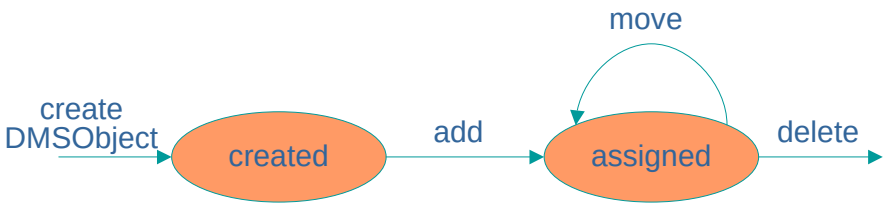
DMS Interfaces (3)

- DMSDocForm
  - Basisinterface für alle Objekte des DMS, die zusätzlich zu strukturierten Daten auch unstrukturierte (wie z.B. Textdateien) enthalten
  - wichtigste Methoden:
    - `byte [] getContent()`
    - `String getExtension()`
    - `int getStatus()`
- DMSNote
  - Spezialform eines `DMSForm`, das eine Notiz (Post-It®) darstellt
  - definiert zwei Datenfelder:
    - `SUBJECT` Betreff der Notiz
    - `CONTENT` Inhalt der Notiz
  - kann an jedes beliebige `DMSObject` angehängt werden

DMS Interfaces (4)

- DMSLink
  - stellt eine Verknüpfung zu einem beliebigen anderen `DMSObject` dar
  - Ausnahme: wenn Zielobjekt auch ein Link ➔ beide Links zeigen dann auf dasselbe Zielobjekt
  - enthält folgende Methoden:
    - `getLinkedObject()`
    - `isLinkValid()`

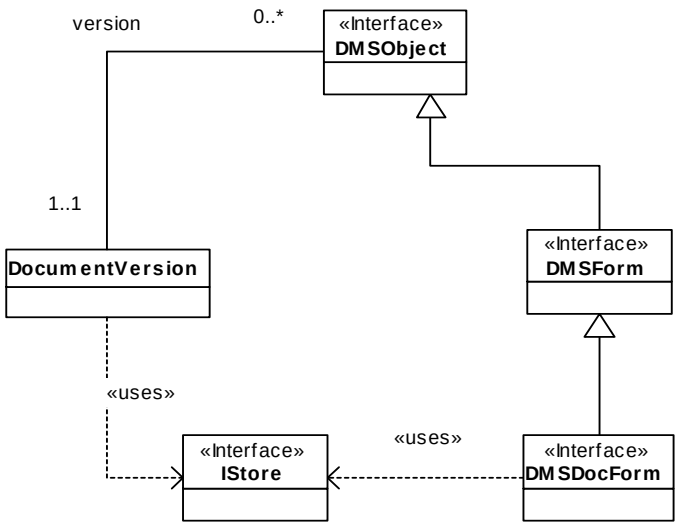
DMSObject Life Cycle



- `DMSObject` muss einem `DMSFolder` zugeordnet werden, wenn es vom DMS verwaltet werden soll
- Zuordnung kann geändert werden

Speichern und Versionieren

@enterprise: Storage and Versions



Speichern und Versionieren (2)

- IStore
  - kümmert sich um die Speicherung der aktuellen Inhalte der Dokumente als auch um die Inhalte der Versionen
  - `DMSForm` verwalten ihre strukturierten Daten selbständig, nur ihre Versionen werden auch vom `IStore` verwaltet
  - Default-Implementierung speichert Inhalte der Dokumente und der Versionen in der Datenbank ab
  - eigene Speichermechanismen können jederzeit implementiert und im System registriert werden (siehe Abschnitt ‚Systemkonfiguration – DMS-Speicherklasse')
- DocumentVersion
  - Speichert die Informationen zu einer Version, vor allem wann und von wem diese erzeugt wurde

@enterprise DMS API

- alle DMS-bezogenen Interfaces befinden sich im package `com.groiss.dms`
- enthält auch das Interface *DMS:*
  - ist **die** Schnittstelle des @enterprise DMS für den API-Programmierer
  - definiert folgende Gruppen von Methoden:
    - Erzeugen der verschiedenen DMS-Objekte
    - Verwaltung der Beziehungen zwischen DMS-Objekten
    - Bearbeiten von DMS-Objekten
    - Navigation innerhalb des DMS
    - DMS-spezifischen Berechtigungsprüfung
    - div. Utility-Methoden

Erzeugen von DMS-Objekten

- jede Art von `DMSObject` hat seine eigene create-Methode:
  - `DMSFolder createFolder(FormType ft, String name, DMSFolder template, User u, ACL acl)`
  - `DMSDocForm createDocForm(FormType ft, String name, String extension, DMSDocForm template, User u, ACL acl)`
  - `DMSForm createForm(FormType ft, DMSForm template, User u, ACL acl)`
  - `DMSNote createNote(String subject, String content, User u, ACL acl)`
- wie bekomme ich den `FormType`?
  - `FormType getFormType(String id, int version)`
  - `FormType getFormType(long oid)`
  - `List listCreateableFormTypes(String searchCond, String order, User user)`

Verwaltung von Beziehungen

- Beziehung zwischen `DMSObject` und `DMSFolder`:
  - `DMSObject add(DMSFolder f, DMSObject o, User u)`
  - `void delete(DMSFolder f, DMSObject o, User u)`
  - `DMSObject move(DMSFolder src, DMSFolder dest, DMSObject doc, short type, User u)`
- verschiede Einsatzmöglichkeiten für `move`-Methode (abhängig vom Parameter `type`):
  - `DMS.MOVE`: Objekt von einem Ordner in einen anderen verschieben
  - `DMS.COPY`: Kopie des Objekts in einen anderen Ordner einfügen
  - `DMS.LINK`: Verknüpfung auf das Objekt in einen anderen Ordner einfügen

Verwaltung von Beziehungen (2)

- Beziehung zwischen `DMSObject` und `DMSNote`:
  - `void attachNote(DMSObject target, DMSNote note, User user)`
  - `void removeNote(DMSObject target, DMSNote note, User user)`
  - `List listNotes(DMSObject target, User user)`
- Beziehung zwischen `DMSObject` und `DocumentVersion`
  - `DocumentVersion makeVersion(DMSObject obj, User user, String description)`
  - `void deleteVersion(DocumentVersion dv, User user)`
  - `List listVersions(DMSObject obj, User user)`

Bearbeiten von DMS-Objekten

- zusätzlich zu den Manipulationsmethoden in den verschiedenen Interfaces definiert das Interface DMS folgende:
  - `DMSObject renameDocument(DMSFolder folder, DMSObject obj, String newName, String newExtension, User u)`
  - `DMSDocForm reloadDocument(DMSFolder folder, DMSDocForm document, String newExtension, InputStream is, User user)`
  - `DMSForm changeType(DMSForm obj, FormType newType, DMSFolder folder, User user)`
  - `void update(DMSObject o)`

Navigation

- zur Navigation innerhalb des DMS stehen folgende Methoden zur Verfügung:
  - `DMSFolder getRootFolder(User user)`
  - `DMSFolder getFolder(DMSObject obj)`
  - `List listSubfolders(DMSFolder startFolder)`
  - `List listContents(DMSFolder folder, FormType ft, String cond, String order, Object[] vals, boolean recursive)`
  - `List listForms(FormType ft, String cond, String order, Object[] vals)`
  - `DMSForm getMainForm(DMSForm f)`
  - `List listSubforms(DMSForm f, int id)`

Berechtigungsprüfung

- zusätzliche Berechtigungsmethoden für DMS-Objekte:
  - `boolean mayView(User user, DMSObject obj)`
  - `boolean mayEdit(User user, DMSObject obj)`
  - `void checkView(User user, DMSObject obj)`
  - `void checkEdit(User user, DMSObject obj)`
  - `void disableRightChecks()`
  - `void enableRightChecks()`
- Grund für diese zusätzlichen Methoden: Auswertung der Rechte erfolgt teilweise anders als über das Standardberechtigungssystem
  - DMS-Objekte von Prozessen: Rechte auf den Prozess gelten auch für all seine DMS-Objekte
  - angehängte Notizen: Rechte des DMS-Objekts als auch der Typ der Notiz (privat/öffentlich) bestimmen die Rechte auf die Notiz

Utilities

- folgende Utility-Methoden stehen zur Verfügung:
  - `DMSObject getDMSObject(String classname, long oid)`
  - `String getIcon(String extension)`
  - `void checkValidName(DMSObject target, String name, String extension)`
  - `void checkDuplicateNames(DMSFolder targetFolder, DMSObject targetObject, String name, String extension)`
  - `boolean isDuplicateName(DMSFolder targetFolder, String name, String extension)`

@enterprise Workflow API

- alle Workflow-relevanten Interfaces befinden sich im Package `com.groiss.wf`
  - `ProcessDefinition` Repräsentiert die Definition eines Prozesses
  - `Task` ein interaktiver Schritt der Prozessdefinition
  - `ProcessInstance` die Instanz eines Prozesses
  - `ActivityInstance` die Instanz eines Prozessschrittes
  - `WfEngine` definiert Methoden zur Bearbeitung von Prozessinstanzen
- Eine Instanz von `WfEngine` wird mittels `ServiceLocator.getWfEngine()` angefordert
- Methoden der `WfEngine` sind in vier Gruppen unterteilt:
  - Prozessinstanzen erzeugen
  - Prozessinstanzen finden
  - Informationen über Prozessinstanzen abfragen
  - ändern des Zustands von Prozessinstanzen

Prozessinstanzen erzeugen

- folgende Daten werden dazu benötigt:
  - die Prozessdefinition
  - der Benutzer, der den Prozess erzeugen will
  - die Organisationseinheit, in der der Prozess erzeugt werden soll
  - das Datum, zu dem der Prozess beendet sein soll (ist optional)
- `WfEngine` bietet folgende Methoden zum Finden einer Prozessdefinition:
  - `ProcessDefinition getProcessDefinition(String id)`
  - `ProcessDefinition getProcessDefinition(String id, int version)`
  - `Vector listProcessDefinitions(Application appl)`
  - `Vector getStartableProcesses(Application appl)`
- gestartet wird der Prozess mit einer der folgenden `WfEngine`-Methode:
  - `ProcessInstance startProcess(ProcessDefinition p, User u, OrgUnit d, Date duedate, String id)`
  - `ProcessInstance startProcess(ProcessDefinition p, User u, OrgUnit d, Date duedate, String id, DMSForm f)`

Prozessinstanzen finden

- folgende Methoden stehen dazu zur Verfügung
  - `Vector getWorklist(Application a, boolean withRepr)`
  - `Vector getRoleWorklist(Application a)`
  - `Vector getSuspensionList(Application a)`
  - `ProcessInstance getProcess(String id)`
  - `ProcessInstance getProcess(long oid)`
  - `ProcessInstance getProcess(DMSForm f)`

Informationen zu Prozessinstanzen

- `ActivityInstance` bietet Zugang zu folgenden Informationen:

  Akteur, Startzeit, Zeitpunkt der Fertigstellung, Status, Organisationseinheit, Prozessdefinition, Prozessinstanz, Typ und Task

- `ProcessInstance` liefert folgende Informationen:

  Id, Subject und DMSFolder

- zusätzlich bietet `WfEngine` folgende Methoden an:
  - `Vector getActiveTasks(ProcessInstance process)`
  - `Vector getActiveTasks(ProcessInstance process, User u)`
  - `Vector getActivityInstances(ProcessInstance process)`
  - `DMSForm getForm(ProcessInstance pi, String name)`
  - `Vector getForms(ProcessInstance process)`
  - `Vector getDocuments(ProcessInstance process)`
  - `Vector getNotes(ProcessInstance process)`
  - `ProcessInstance getMainProcess(ActivityInstance ai)`
  - `ProcessInstance getParent(ActivityInstance ai)`

Prozessinstanzen bearbeiten

- API bietet Methoden für alle Aktionen, die im Worklist-Client ausgeführt werden können:

  `finish, take, untake, goBack, seeLater, seeAgain, setAgent, gotoTask,copyTo, makeBranch, setOrgUnit und setDescription.`

- Folgende Methoden können auf Prozessinstanzen angewendet werden:
  - `void abort(ProcessInstance process)`
  - `void reactivate(ProcessInstance process)`
  - `void archive(ProcessInstance process)`
  - `void setSubject(ProcessInstance process)`
  - `void setSubjectToString(ProcessInstance process, String str)`

Prozesskontext holen

- Methoden für Bedingungen und Systemschritte können die aktuelle Schrittinstanz wie folgt holen:

```
WfEngine e = ServiceLocator().getWfEngine();
ActivityInstance ai = e.getContext();
```


DMS-Utilities für HTML-Interface

- HTMLDMSObject
  - bietet HTML-spezifische Utility-Methoden an, wie z.B.:
    - `static Page showDocs(HttpServletRequest req)`
    - `static Page showDocs(DMSFolder f)`
    - `static String getDocsUrl(DMSFolder folder, ActivityInstance task, String actions, String pathToRoot)`
    - `static String getEditUrl(DMSObject object, DMSFolder folder, boolean readOnly)`
    - `static DMSObject getDMSObject(String classAndOid)`
- DMSObjectTable
  - Standard-Tablemodel zur Auflistung der Ordnerinhalte
  - notwendige Ableitungen sollten von dieser Klasse erben

DMS Änderungen 5.1 -> 6.0

- Package `com.groiss.dms` löst `com.dec.avw.dms` ab.
  Darin enthalten:
  - Die bereits erwähnten Interfaces
  - `DocForm` (vormals im `core`-Package)
  - `DMS` (vormals im Package `com.dec.avw.dms.api`)
  - `IStore`
- Package `com.groiss.dms.html`. Darin enthalten:
  - `HTMLDMSObject` (löst `com.dec.avw.html.HTMLDocument` ab)
  - `DMSObjectTable` (löst `com.dec.avw.html.DocumentTable` ab)

DMS Änderungen 5.1 -> 6.0 (2)

- Obsolete Klassen:
  - `core.DocForm`: ersetzt durch `dms.DocForm`
  - `Document2`: geht in `dms.DocForm` auf
  - `DMSDocument`: wird von DMS-Interfaces abgelöst
  - `ProcessDocument`: Prozesse verwenden das DMS zur Verwaltung ihrer Dokumente und Notizen
- FormType
  - `docattached` (boolean) wird durch `type` (short) abgelöst
  - mögliche Werte für `type`:
    - `FormType.PROCESS_FORM` implementiert `DMSForm`, erweitert `Form`
    - `FormType.DOCUMENT_FORM` implementiert `DMSDocForm`, erweitert `DocForm`
    - `FormType.FOLDER_FORM` implementiert `DMSFolder`, erweitert `FolderForm`

Änderungen 6.0 -> 6.1

- Einführung der Packages `com.groiss.org` und `com.groiss.wf`
  - alle darin enthaltenen Interfaces werden von den meist gleichnamigen `core`-Klassen implementiert (z.B. `com.dec.avw.core User` implementiert `com.groiss.org.User`)
  - nicht gleichnamige Interfaces und Klassen:
    - `OrgUnit` — `Dept`
    - `ActivityInstance` — `StepInstance`, die einen Schritt repräsentiert
    - `ProcessInstance` — `StepInstance`, die einen Prozess repräsentiert

- Änderungen im DMS:
  - Interfaces und Klassen im DMS verwenden die oben genannten Interfaces
  - Klasse `DMS` wurde zum Interface; Implementierung bekommt man mittels `ServiceLocator.getDMS()`

Beispiel 1
Neues Dokument zu Prozess

- Schritt 1: Maske zum Auswählen des Prozesses und des Dokuments

```
public Page showAddDocMask(HttpServletRequest req) throws Exception
{
   Vector v = ServiceLocator.getWfEngine().getWorklist(null,false);
   DropdownList l = new DropdownList("process");
   for (Enumeration e = v.elements(); e.hasMoreElements(); ) {
     ActivityInstance ai = (ActivityInstance)e.nextElement();
     ProcessInstance pi = ai.getProcessInstance();
     l.addOption("" + pi.getOid(), pi.toString());
   }
   HTMLPage page = new HTMLPage();
   page.setPage(
     "<form method=\"post\" enctype=\"multipart/form-data\" "+
     "action=\"com.groiss.demo.dms.DMSDemo.addDoc\">" +
     "Process:" + l.show() +
     "<br>File: <input type=\"file\" name=\"file\">" +
     "<br>Name: <input type=\"text\" name=\"name\">" +
     "<br><input type=\"submit\"></form>");
   return page;
}
```

Neues Dokument zu Prozess (2)

- Schritt 2: Eingaben auslesen und Dokument zum ausgewählten Prozess hinzufügen

```
public Page addDoc(HttpServletRequest req) throws Exception {
   //transform the request because we need a MultipartRequest
   //when handling files
   MultipartRequest mreq = MultipartRequest.createInstance(req);
   //get the current user
   User user = (User)ThreadContext.getThreadPrincipal();
   //get the selected process
   WfEngine e = ServiceLocator.getWfEngine();
   ProcessInstance process =
     e.getProcess(Long.parseLong(req.getParameter("process")));
   //get the specified name and divide it into the name and the
   //extension (e.g. doc for Word files)
   String tmpName = mreq.getParameter("name");
   int idx = tmpName.lastIndexOf(".");
   String name = tmpName.substring(0, idx);
   String extension = tmpName.substring(idx+1);
   //get the file
   File file = mreq.getFile("file");
```


Neues Dokument zu Prozess (3)

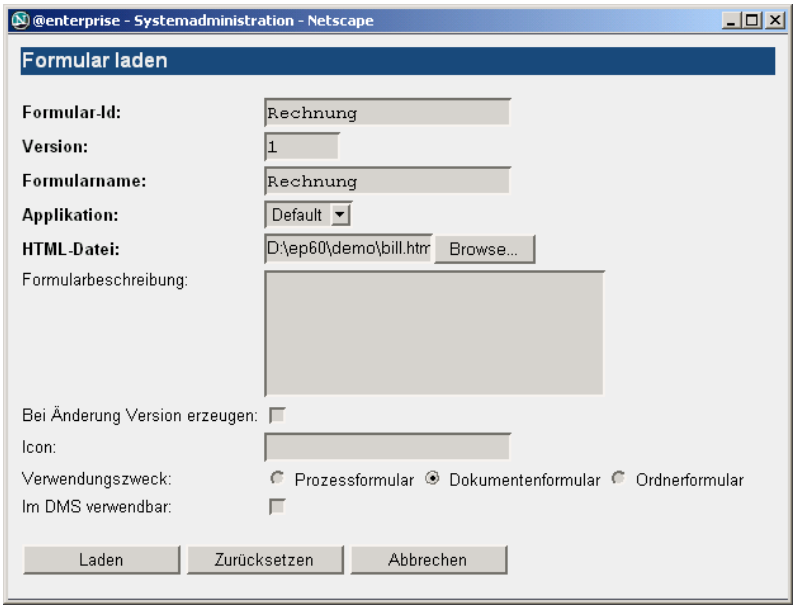
```
   //create a new standard document and add it to the process.
   DMS dms = ServiceLocator.getDMS();
   FormType ft = dms.getFormType(FormType.STANDARD_DOCUMENT);
   DMSDocForm newDoc = dms.createDocForm(ft, name, extension,
     null, user, null);
   dms.add(process.getDMSFolder(), newDoc, user);
   //check in the content of the file
   newDoc.checkIn(user, new FileInputStream(file));
   //return an answer
   HTMLPage page = new HTMLPage();
   page.setPage("<html>Upload done.</html>");
   return page;
}
```

Beispiel 2
Rechnungen und Rechnungsorder

- Aufgabenstellung:
  - lege ein Dokumentenformular für Rechnungen an
  - lege ein Ordnerformular für Rechnungen an
  - Ändere die Tabellendarstellung wie folgt:
    - eine zusätzliche Spalte soll angezeigt werden, die angibt, welche Rechnungen bereits bezahlt wurden und welche nicht
    - am Ende der Tabelle soll die Summe der Rechnungsbeträge angezeigt werden

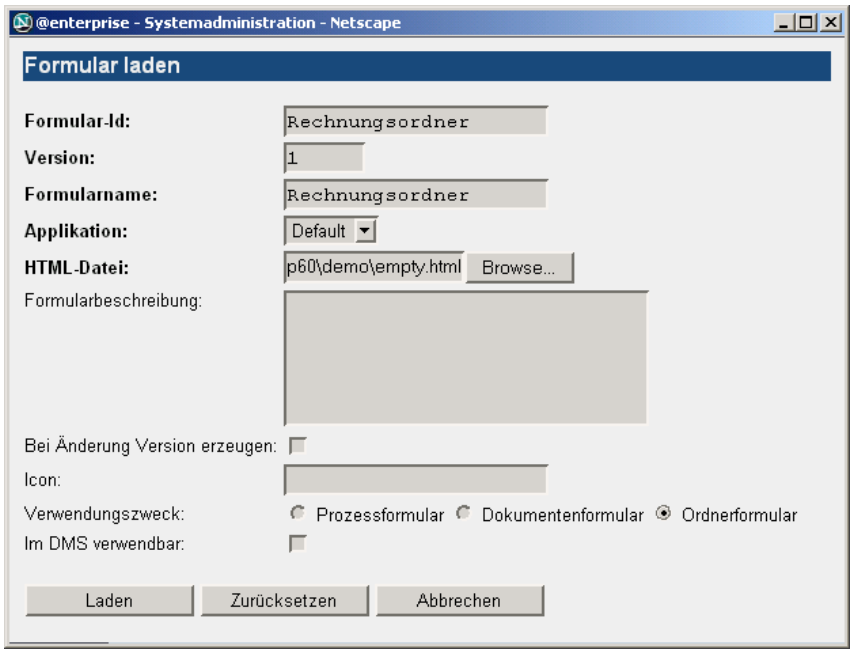
Rechnungen und Rechnungsorder (2)

- Schritt 1: Rechnungsformular in der Administration anlegen



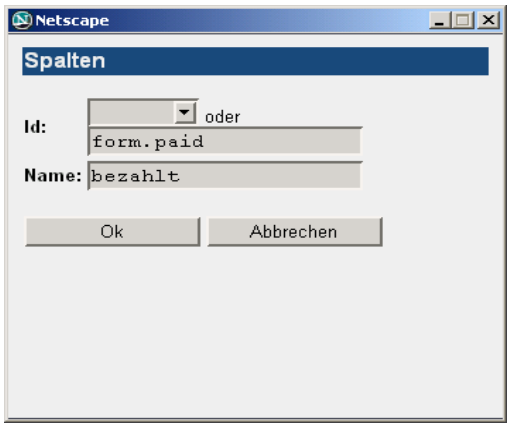
Rechnungen und Rechnungsorder (3)

- Schritt 2: Rechnungsordner über die Administration anlegen



Rechnungen und Rechnungsorder (4)

- Schritt 3: Zusätzliche Spalte ‚bezahlt' in der Rechnungstabelle
  - 1. Detailansicht des Rechnungsordners in der Administration öffnen
  - 2. Auf Button ‚Tabellendarstellung' klicken
  - 3. Im Dialog zur Tabellendarstellung auf Hinzufügen klicken
  - 4. Folgende Daten in die Maske eingeben



Rechnungen und Rechnungsorder (5)

- Schritt 4: Summe der Rechnungsbeträge anzeigen
  - 1. HTML-Maske für Tabelle kopieren und adaptieren

```
...
<form>
<input type=hidden name=order value="%order%">
<input type=hidden name=functionTask>
<input type=hidden name=func>
%tab%
<a id="webfolder" STYLE="behavior:url('#default#AnchorClick')" target=xx
   folder="http://%host%/webdav/%path%/"></a>
<p>
%sum%
</form>
...
```

Die zu kopierende Maske befindet sich in masks.jar unter dem Pfad \mask\dms\ und heißt tab.html

Rechnungen und Rechnungsorder (6)

- 2. Eigenes TableModel schreiben

```
public class BillFolderTable extends DMSObjectTable {

   public HTMLPage getTableMask() throws Exception {
     return new HTMLPage("/com/groiss/demo/billtab.html");
   }

   public HTMLPage getHTMLPage() throws Exception {
     HTMLPage p = super.getHTMLPage();
     Connection conn = DBConnPool.getConnection();
     Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery("select sum(amount) from " +
        "form_bill_1 where oid in (select item from " +
        "avw_dmsfldritemrel where folder="+folder.getOid()+")");
     rs.next();
     long sum = rs.getLong(1);
     p.substitute("sum", "summe: "+sum);
     return p;
   }
```

Rechnungen und Rechnungsorder (7)

- 3. Ordnerklasse ändern

```
public class BillFolder_1 extends FolderForm {
   ...

   public String getTableModel () {
    return "com.groiss.demo.BillFolderTable";
   }
```

Anmerkung: den Sourcecode der generierten Klasse BillFolder_1 finden Sie im tmp-Verzeichnis ihrer @enterprise-Installation

Beispiel 3
Rechnungsordner und Tabelle

- Aufgabenstellung:
  - Ändere das Ordnerverhalten wie folgt:
    - Der Ordner akzeptiert nur Rechnungen. Zusätzlich sollen alle eingefügten Standarddokumente automatisch in Rechnungen sowie alle eingefügten Standardordner automatisch in Rechnungsordner umgewandelt werden (um die Benutzung von WebDAV zu ermöglichen)
  - Ändere die Tabellendarstellung wie folgt:
    - die Spalte ‚bezahlt' soll den Wert ‚Ja' anzeigen, wenn die Rechnung bezahlt wurde, ansonsten ‚Nein'
    - überschreibe die verfügbaren Aktionen, so dass nur die Aktionen ‚Neuer Ordner', ‚Neues Dokument' und ‚bezahlt' zur Verfügung stehen

Rechnungsordner und Tabelle (2)

- Schritt 1: add-Methode des Ordners überschreiben

```
static FormType billFT;
static FormType billFolderFT;
static FormType standardDoc;
static FormType standardFolder;


static {
      DMS dms = ServiceLocator.getDMS();
      billFT = dms.getFormType("Bill", 1);
      billFolderFT = dms.getFormType("BillFolder", 1);
      standardDoc = dms.getFormType(FormType.STANDARD_DOCUMENT);
      standardFolder = dms.getFormType(FormType.STANDARD_FOLDER);
}
```


Rechnungsordner und Tabelle (3)

```
/** Only bills and bill folders are allowed within this folder*/
public DMSObject add(DMSObject o)throws Exception{
   DMS dms = ServiceLocator.getDMS();
   String oClass = o.getClass().getName();
   if(oClass.equals(billFT.getClassName()) ||
         oClass.equals(billFolderFT.getClassName())){
      return super.add(o);
   } else if(oClass.equals(standardDoc.getClassName())){
      //change the type of o to Bill
      return super.add(dms.changeType((DMSForm)o, billFT, this,
         o.getOwner()));
   } else if(oClass.equals(standardFolder.getClassName())){
      //change the type of o to BillFolder
      return super.add(dms.changeType((DMSForm)o, billFolderFT, this,
         o.getOwner()));
   } else {
      throw new ApplicationException("Only bills and bill folders " +
         "may be added to this folder.");
   }
}
```

Rechnungsordner und Tabelle (4)

- Schritt 2: formatierte Spalte ‚bezahlt' durch überschreiben der entsprechenden Methode im TableModel

```
protected Object getColumnValue(DMSObject doc, String col) throws
Exception {
   if (col.startsWith("form.paid")) {
      String colname = col.substring(5);
     if (doc instanceof Bill_1) {
        Bill_1 bill = (Bill_1) doc;
        if("1".equals(bill.paid)){
           return DefaultResource.getString("yes");
        } else {
           return DefaultResource.getString("no");
        }
     } else {
        return " ";
     }
   } else {  //all other columns are handled by the super class
     return super.getColumnValue(doc, col);
   }
}
```


Rechnungsordner und Tabelle (5)

- Schritt 3: verfügbare Aktionen festlegen
  - 1. Methode getActions() im TableModel überschreiben

```
public String getActions(){
  if(mode == RW){
   return "newFolder newDocument space com.groiss.demo.DMSDemo.billPaid";
  } else {
   return "newFolder_grey newDocument_grey space billPaid_grey";
  }
}
```

Rechnungsordner und Tabelle (6)

- 2. Methode billPaid in Klasse DMSDemo implementieren

```
public Page billPaid(HttpServletRequest req) Exception {
   //get the form
   long formOid = req.getParameter("oid");
   DMSForm bill = (DMSForm) HTMLDMSObject.getDMSObject(oid);
   //set it to be paid
   bill.setField("paid", "1");
   ServiceLocator.getDMS().update(bill);
   //reload the table
   return HTMLDMSObject.showDocs(req);
}
```


Rechnungsordner und Tabelle (7)

- 3. ResourceBundle überschreiben (siehe @enterprise Programmierhandbuch)

```
package com.dec.avw.resource;
import java.util.ListResourceBundle;

public class Strings_de_AT extends ListResourceBundle implements
   java.io.Serializable {

   public Object[][] getContents() {
     return contents;
   }

   static final Object[][] contents = {
     {"com.groiss.demo.DMSDemo.billPaid_icon","bezahlt"},
     {"billPaid_grey_icon","<font color=#848284>bezahlt</font>"}
   };
}
```