Navigation Framework

Matthias Kunze, David Tibbe, Tobias Vogel
Seminar Konzepte und Methoden der Webprogrammierung mit PHP und MySQL
Hasso-Plattner-Institut für Softwaresystemtechnik
matthias.kunze@student.hpi.uni-potsdam.de,
david.tibbe@student.hpi.uni-potsdam.de,
tobias.vogel@student.hpi.uni-potsdam.de

Abstract

Ziel des Projektes ist es, den Webauftritt www.tele-task.de neu zu gestalten und dabei grundlegende Prinzipien und Technologien der Web-Programmierung zu erlernen.

Diese Ausarbeitung veranschaulicht die für die Navigation entworfenen Konzepte, die dahinterliegende architekturelle Umsetzung und die Benutzung über Schnittstellen. Ferner werden die beteiligten Akteure, Basistechnologien und Objekte beschrieben.

Abgeschlossen wird mit einem Ausblick über Verbesserungs- und Erweiterungsmöglichkeiten der Seite.

1. Einleitung

Die Fähigkeit, zwischen Webseiten navigieren zu können, ist einer der Grundpfeiler des WWW. Um dieser besonderen Bedeutung Rechnung zu tragen, hat sich diese Gruppe speziell mit den Navigationsaufgaben befasst. Dazu gehört auch der Aufbau der Darstellung anhand von parametrisierten Modulen.

Kapitel 2 erläutert die wesentlichen Konzepte, die der Seitennavigation zugrunde liegen.

In Kapitel 3 wird die notwendige Architektur dargestellt, um diese Aufgaben zu erfüllen. Die eigentliche Arbeit übernehmen verschiedene Akteure, die sich mit unterschiedlichen Themen (URI-Auswertung und Page(let) Controller, aber auch Internationalisierung und Benutzerverwaltung) befassen. Außerdem wird die Datenstruktur erläutert, die den angesprochenen Akteuren spezielle Informationen über Benutzer, Parameter und Seiten liefert, als auch die Datenbankschemas, die diese Datenstrukturen aufbauen.

Kapitel 4 umfaßt eine Erläuterung der konkreten Anwendung der besprochenen Konzepte. So wird die Syntaxprüfung verdeutlicht, die Schnittstellen für Seiten, Pagelets und Page Controller bekanntgegeben und die zur Verfügung stehenden Ausnahmen beschrieben.

Kapitel 5 beschäftigt sich abschließend mit dem Administrieren der Seite. Der Seitengraf kann verändert, Page Controller dazugefügt und Pagelet Container angepasst werden.

Kapitel 6 enthält das Datenbankschema als Anlage.

2. Grundkonzepte

Die Navigation beruht auf drei wesentlichen Konzepten: der *page_id*, der *mode_id* und den Page Contollern.

Eine Seite im Navigationsbaum besitzt eine ID, die *page_id*. Anhand dieser werden in der Datenbank die Eigenschaften einer Seite identifiziert. Das können beispielsweise Titel, Meta-Tags oder Darstellungsrichtlinien sein.

Der Modus einer Seite hingegen bestimmt die „Sicht“, die der Benutzer auf sie hat. So wird eine Seite, auf der Nachrichten angezeigt werden, mindestens die beiden Modi „Anzeigen“ und „Editieren“ haben, was gleichzeitig die Standard-Modi sind, die jede Seite hat. Zusätzlich können beliebige weitere Modi zu einer Seite hinzugefügt werden, um feinergranulare oder ganz andere Sichten zu ermöglichen.

Zum Beschaffen des eigentlichen Inhalts, der im Hauptanzeigeteil der Webseite erscheinen soll, dient ein Page Controller (PC). Der *Request Manager* fügt dessen Code ein, damit der *Page Controller Manager* anschließend ein PC-Objekt instanziieren kann. Danach wird an diesem eine Methode aufgerufen und ihm die Parameter der Adresszeile übergeben, um den Inhalt als String zurückzubekommen. Es kann für jede *page_id*/*mode_id*-Kombination ein unterschiedlicher Page Controller festgelegt werden, der jeweils von der abstrakten Klasse *PageController* erbt und von den anderen Gruppen implementiert wird.
Neben dem eigentlichen Seiteninhalt gibt es soge-

nannte Pagelets, kleine Infokästchen, die am Rand der Seite angezeigt werden. Diese enthalten sekundäre Informationen wie Ankündigungen, Direktlinks zu aktuell laufenden Vorlesungen oder Benutzerinformationen. Pagelets werden in Pagelet Containern zusammengefasst, die – analog den Page Controllern – einer *page_id*/*mode_id*-Kombination zugeordnet sind.

3. Architektur

Das Navigation Framework wird mit der Skriptsprache PHP programmiert und an eine geeignete Webdatenbank (z.B. MySQL) angeschlossen. Um die Zugriffe auf die Datenbank zu optimieren und verschiedene Datenbanksysteme verwenden zu können, wird als Datenbankabstraktionsschicht das Open-Source-Framework ADO-DB[1] eingesetzt.

Die Ausgabe der Inhalte und des Seitenlayouts erfolgt durch die Template-Engine Smarty[2]. Damit können Daten unabhängig von ihrer Präsentation übergeben und mit Hilfe unterschiedlicher, HTML-basierter Templates gerendert werden. Außerdem bietet Smarty Leistungsoptimierungen an: Veränderte Templates werden vorkompiliert und serverseitig gecacht. Damit entfällt das aufwändige Rendern der Seiten bei jedem erneuten Aufruf.

Zugriff auf das Dateisystem erhalten alle Komponenten über die von PHP zur Verfügung gestellte File-API.

Zu beachten ist, dass alle erzeugten Objekte nur jeweils einem Request zugeordnet sind. Serverseitig wird jeder Request in einem eigenen Prozessraum verarbeitet. Das bedeutet, dass auf dem Server sehr viele identische Objekte erzeugt werden; diese haben aber keinen Bezug zueinander. Nur Objekte innerhalb eines Prozessraumes stehen in einem gemeinsamen Kontext, globale Objekte sind auch nur innerhalb dieses Kontextes global. Im Folgenden wird nur ein Prozessraum betrachtet auf den sich alle Äußerungen zu Aufbau und Ablauf beziehen.

3.1. Akteure

Abbildung 3.1 zeigt den kompletten Aufbau des Systems.

Der **Request Manager** nimmt per HTTP-Request alle Anfragen des Benutzers entgegen, verarbeitet die via `$_REQUEST`-Variable übergebenen Parameter, lädt den Quellcode der an der Generierung der Seite beteiligten Akteure im System und startet sie. Nachdem diese Akteure ihre Aufgabe erfüllt haben, erzeugt der Request Manager die HTML-Ausgabe der kompletten Seite (inkl. Navigationspanels usw.) und schickt sie als HTTP-Response zurück an den Benutzer. Im Idealfall erhält der Benutzer die gewünschte Ausgabeseite, es kann aber situationsbedingt auch auftreten, dass der Request Manager Fehlermeldungen, Login-Seiten oder Zugriffsverweigerungen zurückgibt.

Die Variable `$_REQUEST` wird von PHP beim Aufruf der Seite *index.php* angelegt und enthält alle Parameter, die von einer Seite per POST und GET übertragen werden. So enthält sie auch die Werte für die Parameter *mode_id* und *page_id*.

[1] siehe http://adodb.sourceforge.net/
[2] siehe http://smarty.php.net/

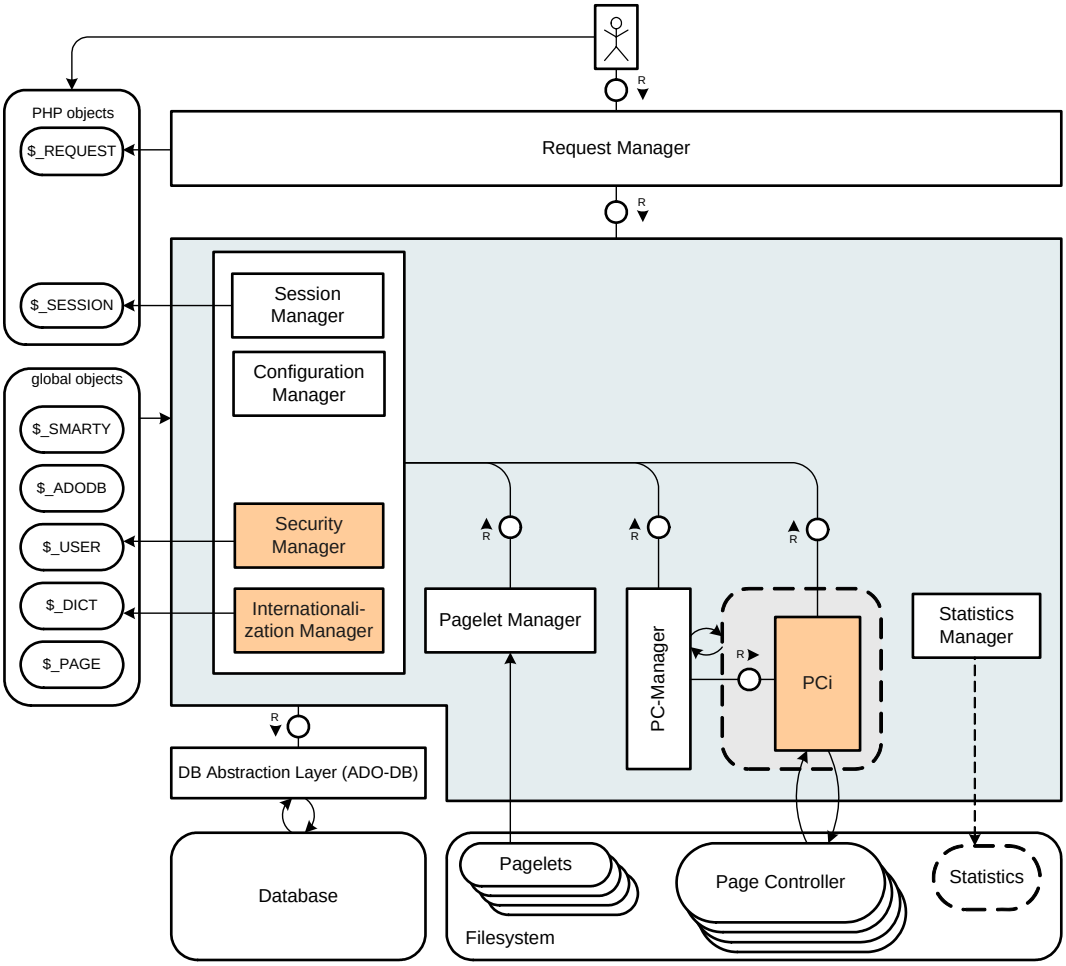


Abb. 3.1: Aufbaudiagram Systemstruktur

Der **User Manager** authentifiziert und autorisiert einen Benutzer beim Zugriff auf Inhalte. Solange für eine anzuzeigende Seite keine Autorisierung erforderlich ist, wird der Benutzer als Gastbenutzer authentifiziert und erhält den Status *nicht angemeldet*. Ist allerdings die Anzeige einer Seite geschützt, muss sich der Benutzer anmelden. Dazu wird ihm der Request Manager (weil er die entsprechende AccessException fängt) ein entsprechendes Login-Formular anbieten. Besitzt der Benutzer ein gültiges Benutzerkonto, kann er sich anmelden und wird im Folgenden auch als dieser Benutzer authentifiziert. Besitzt er kein gültiges Benutzerkonto, so bleibt er als Gastbenutzer authentifiziert, diesmal aber mit dem Status *angemeldet*, um weitere, automatische Login-Versuche zu verhindern.

Hat der (möglicherweise als Gastbenutzer) angemeldete Benutzer nicht die erforderlichen Rechte zum Anzeigen einer Seite, so wird ihm dies per Fehlermeldung mitgeteilt.

Der **Internationalization Manager** stellt dem gesamten System ein zentrales Wörterbuch zur Verfügung, mit dem Übersetzungen zu Inhalten aber auch Navigationselementen und Steuerobjekten erfragt werden können. Dazu muss jede Komponente, die beim Internationalization Manager Übersetzungen erfragt, die entsprechende Übersetzung selbst eintragen und registrieren.

Alle in der von PHP erzeugten `$_SESSION`-Variable gespeicherten Sitzungsdaten werden vom **Session Manager** verwaltet. Diese ist ein serverseitig gespeichertes Array, das beliebige Datentypen, z.B. wiederum Arrays, enthält. Diese Daten bleiben über einen Request hinaus gespeichert und stehen so einer Anwendung persistent zur Verfügung. Die Serialisierung und Deserialisierung dieser Variable übernimmt PHP.

Die Sitzungsdaten werden einem einzelnen Benutzer zugeordnet, dazu wird bei jedem HTTP-Request und –Response eine Session-ID übertragen, welche den zugehörigen Sitzungsda-

tensatz serverseitig identifiziert. Die Session-ID kann clientseitig in einem Cookie gespeichert oder bei jedem Request und Response mitübertragen werden.

Im Navigation Framework bietet das Konzept der Session also erhebliche Vorteile, insbesondere kann in den Sitzungsdaten gespeichert werden, ob und welcher Benutzer am System angemeldet ist. Allerdings birgt dieses Konzept auch Gefahren: Die Session-ID könnte an Dritte geraten, womit diese möglicherweise kritischen Zugriff auf das System erlangen. Außerdem kann es passieren, dass Komponenten des Systems Einträge in der `$_SESSION`-Variable gegenseitig überschreiben. Dies stellt der Session Manager durch die Verwendung separierter Namensräume sicher.

Die zentrale Komponente des Navigation Frameworks ist der **Page Controller Manager (PCM)**. Dieser startet den jeweiligen Page Controller, erfragt die Ausgabedaten und übermittelt sie an den Request Manager, der diese dann unter Zuhilfenahme der Smarty-Engine ausgibt.

Mithilfe der übertragenen *mode_id* und *page_id* identifiziert der PCM einen Page Controller. Bevor dieser gestartet wird, überprüft der PCM noch die per `$_REQUEST` übertragenen Parameter auf Gültigkeit. Dazu wurde beim Navigation Framework ein Validation-Pattern hinterlegt, mit dessen Hilfe die einzelnen Parameter auf syntaktische Gültigkeit unter Verwendung regulärer Ausdrücke geprüft werden können (siehe 4.3: Validation Pattern).

Werden die Parameter validiert, so lädt der PCM den Code des Page Controllers und erzeugt das initiale Objekt vom Typ des Page Controllers. Danach erfragt der PCM die Informationen für Ausgabe und eventuelle Zusatzinformationen wie JavaScript, CSS oder Metaangaben.

Für den Page Controller besteht die so genannte *Hard Access Validation*. Das bedeutet: Ist ein Benutzer nicht angemeldet, muss er sich anmelden um die Seite zu betrachten. Kann er sich nicht anmelden, oder besitzt er nicht ausreichende Berechtigung um die Seite zu betrachten, so wird ihm eine Fehlermeldung ausgegeben.

Ein **Page Controller** ist eine eigenständige Komponente, die den Inhalt einer Seite ausgibt. Im Systemaufbau (3.1) wird der Page Controller als Strukturvarianz dargestellt, schließlich wird er erst zur Laufzeit identifiziert und erzeugt. Über die globalen Objekte erhält der Page Controller Zugriff auf alle Informationen über den Benutzer und den aktuellen Request. (siehe auch 3.2: Globale Objekte) Weiterhin stehen ihm mit den bereits initialisierten Objekten `$_ADODB` und `$_SMARTY` eine bestehende Datenbankverbindung und ein Smarty-Objekt zur Verfügung. Der Page Controller gehört damit nur implizit zum Navigation Framework, da er eine eigenständige Komponente ist, die nur dann in das System integriert werden kann, wenn sie von der abstrakten Klasse *Page Controller* erbt und die virtuellen Methoden implementiert.

Alle Komponenten des Systems können ihre Konfiguration zentralisiert speichern und abrufen. Dazu wird das Objekt des **Configuration Managers** angelegt.

Durch die zentrale Speicherung wird Entwicklungsaufwand gespart, da keine eigene Schnittstelle zum Speichern von Daten benötigt wird. Außerdem werden diese in der Datenbank abgelegt, was das Lesen und Schreiben von Konfigurationsdaten nochmals optimiert.

Die Pagelets werden vom **Pagelet Manager** geladen, zu einem String zusammengefügt und zur Ausgabe an den Request Manager übergeben. Dabei enthalten Pagelets nur minimale Funktionalität, können nur einfache HTML-Inhalte ohne zusätzliche Cascading Style-sheets oder Javascript-Funktionen ausgeben werden. Im einfachsten Fall wird statisches HTML ausgegeben.

Die Pagelets können auch Benutzerrechte erfordern, allerdings werden diese nur mit *Soft Access Validation* überprüft: Besitzt ein Benutzer nicht die nötigen Rechte um ein Pagelet zu betrachten, so wird dieses einfach ausgeblendet.

Um Informationen über das Verhalten von Benutzern auf der Seite zu sammeln, wird jeder Request mit entsprechenden Informationen zu der besuchten Seite (page_*id*), dem Betrachtungsmodus (*mode_id*), Benutzer, eventuelle Fehlermeldungen und Zeitpunkt des Seitenaufrufs vom **Statistics Manager** gespeichert.

Damit lassen sich z.B. hoch frequentierte Seiten ermitteln und der Zugriff durch kürzere Navigationspfade optimieren. Der Statistics Manager stellt eine eigene Abstraktionsschicht zur Verfügung, mit der Statistiken direkt in einer Log-Datei oder in einer Datenbank gespeichert werden können.

3.2. Globale Objekte

Das Navigation Framework stellt allen beteiligten Komponenten globale Objekte zur Verfügung. Um diese Objekte zu verwenden, müssen sie im Kontext einer Funktion als global deklariert, oder als Index des von PHP bereitgestellten, globalen Array `$_GLOBALS` erfragt werden:

```
// Variante 1
private function myFunction() {
  global $_PAGE; // ist außerhalb deklariert
  echo $_PAGE->getPageID();
}

// Variante 2
private function myFunction() {
  echo $_GLOBALS['PAGE']->getPageID();
}
```

Das **Dictionary Object** (`$_DICT`)wird beim ersten Aufruf des Internationalization Managers erzeugt und ist in Wahrheit kein Objekt einer benutzerdefinierten Klasse. Es ist ein assoziatives Array, welches die Übersetzung des Systemwörterbuches enthält, also z.B. Übersetzungen für Menüs, Seitentitel oder Schaltflächen. Die Übersetzungen für Inhaltselemente der Page Controller müssen beim statischen Objekt der Klasse `Dictionary` (`$_DICT`) erfragt werden.

Das **User Object** (`$_USER`) wird vom Session Manager angelegt, nachdem dieser die Sitzungsdaten des Benutzers aufgelöst hat.

Mit Hilfe dieses Objektes kann eine Komponente im System die Autorisierung eines Benutzers erfragen, ohne den Benutzer kennen zu müssen. Mit der Methode `getPermisson(integer accessCode)` des User-Objecks kann die benötigte Autorisierung erfragt werden. Das Benutzerobjekt entscheidet, ob der Benutzer diese Autorisierung erfüllt oder nicht und gibt entsprechend *true* oder *false* zurück.

Weiterhin stellt das User Object Informationen über den Nutzer zur Verfügung: die User-Id, den Benutzernamen, den natürlichen Namen, die E-Mail Adresse und weitere.

Das **Page Object** (`$_PAGE`) wird vom Request Manager erzeugt. Es enthält alle Informationen zum aktuellen Request.

Über das Interface *IPageObject* können alle Informationen über den aktuellen Seitenaufruf erfragt werden, die nicht nutzerspezifisch sind. Das sind unter anderem: *page_id* und *mode_id*, die validierten Requestparameter (getrennt nach GET und POST) und die aktuell ausgewählte Sprache.

`$_SMARTY` ist das bereits geladene und initialisierte **Smarty Object**, mit welchem die Page Controller ihre Inhalte rendern und an den PCM zurückgeben können.

Die Page Controller müssen sich nicht um Installationspfade von Smarty kümmern und können das Objekt direkt verwenden. Außerdem optimiert die globale Verwendung desselben Objektes auch die Arbeitsspeicherausnutzung auf dem Server.[3]

Stellt eine Komponente bei der Verwendung einen Fehler mit dem Smarty Object fest, so muss es die Ausnahme *SmartyException* werfen.

Das **ADODB Object** (`$_ADODB`) leistet Ähnliches wie das Smarty Object, nur stellt es bereits eine geöffnete Verbindung zur Datenbank zur Verfügung. Damit müssen die Komponenten des Systems nicht mehr für die Authentifizierung am Datenbankserver sorgen.

Durch die Verwendung des ADODB Objects wird außerdem sichergestellt, dass alle Komponenten unabhängig vom verwendeten Datenbanksystem bleiben und dieses je nach Bedarf ausgetauscht werden kann. Die Verbindung wählt dabei keine Datenbank aus. Folglich müssen die SQL-Queries voll qualifizierte Relationsnamen inklusive der Datenbanknamen enthalten.

Genau wie beim Smarty Object ist auch hier im Fehlerfall eine eigens entworfene Ausnahmeklasse *ADODBException* zu verwenden, wodurch dem Navigation Framework eine Korruption der Datenbankverbindung angezeigt wird.

3.3. Infrastruktur

Das Datenbankmodell, das dem Navigation Framework zugrunde liegt, bildet die Infrastruktur des gesamten Systems. Erst damit können Seiten ausgewählt, Page Controller und Pagelets geladen und ausgegeben werden.

3.3.1. Identifizierung von Seiten und Page Controllern

Die Relation **pages** erzeugt den Seitenbaum, der der gesamten Struktur der Seite zugrunde liegt. *parent_id* identifiziert dabei den Elternknoten; für

[3] aktuell übliche Serverkonfigurationen stellen dem Prozessraum eines Requests lediglich 16 MB Arbeitsspeicher zur Verfügung.

alle Seiten auf der obersten Ebene ist *parent_id* = 0. Die Angaben zu *hide, start, stop* geben an, in welchem Zeitraum eine Seite angezeigt werden soll. Mit *hide* kann man *start* und *stop* überschreiben, um sie besonders einfach vorübergehend auszublenden.

Besonderes Augenmerk sollte *display_position* zugewandt werden, da Links zu Seiten in verschiedenen Menüs dargestellt werden sollen. Zum Beispiel könnten alle „normalen" Seiten im üblichen Navigationsmenü ausgegeben werden, wohingegen Links zu Sitemap, Impressum und Kontakt in einem separaten Menü (z.B. Footer) dargestellt werden sollen. Der Wert von *display_position* ist eine Bitmaske.

Diese Bitmaske wird für Smarty entschlüsselt und die boolean-Variablen dp_header, dp_footer usw. belegt und Smarty übergeben. Das Seiten-Template wird dann beispielsweise um die Header-Darstellung herum folgende Struktur haben:

```
{if $dp_header}
<!-- some html-tags -->
{header}
<!-- some html-tags -->
{endif}
```

Um ein Höchstmaß an Effektivität und Flexibilität in der Seitengestaltung zu erreichen, wurde das Attribut *mount_point* eingeführt. Einen Mount Point kann man sich wie einen symbolischen Link vorstellen, der auf eine andere, bereits existierende Seite (*page_id*) im Seitenbaum verweist. Ist er gesetzt, so wird anstelle des eigentlich zu benutzenden Page Controllers der der „gemounteten" Seite geladen. Damit lassen sich über hierarchische Strukturen hinausgehende Seitengraphen konstruieren.

modes speichert verschiedene Betrachtungsmodi für das Framework. Mit *disable* können konkrete Modi global deaktiviert werden. Das Attribut *name* wird zu Administrationszwecken verwendet und sollte kurz die Funktion des Modus' wiedergeben.

Die Relation **pagecontroller** enthält alle für das Framework notwendigen Daten, um einen bereits identifizierten Page Controller im Dateisystem zu lokalisieren *(filename)*, und die Klasse zu instanzieren *(classname)*. Dem Konstruktor werden keine Parameter übergeben. *description* wird wiederum zu Administrationszwecken verwendet und sollte Informationen zum Page Controller-Namen und dessen Funktion enthalten.

Die zentrale Rolle bei der Seitengenerierung spielt die Relation **page2mode**. Aus den per Request übergebenen Angaben zu *page_id* und *mode_id* wird das zugehörige Tupel identifiziert. *accesscode* repräsentiert die benötigten Benutzerrechte zum Betrachten der ausgewählten Seite. Sind sie ausreichend, werden wiederum die Request-Parameter mit dem *validation_pattern* verglichen. Können die Parameter validiert werden, so wird der entsprechende Page Controller über die *pagecontroller_id* identifiziert. *order* gibt die Reihenfolge von Links derselben Tiefe in einem Menü an.

3.3.2. Identifizieren von Pagelets zur gewählten Seite

Mit der Identifikation eines Tupels aus page2mode wird auch ein Pagelet Container identifiziert, der alle Pagelets enthält, die auf der aktuellen Seite angezeigt werden sollen.

Die n:m-Beziehung zwischen Pagelets und Pageletcontainern wird durch die Relation **pageletgraph** gespeichert, *order* gibt die Reihenfolge von Pagelets in einem bestimmten Container an.

Ähnlich zu den Page Controllern werden Dateipfad *(filename)* und Klassenname *(classname)* zu Pagelets gespeichert, damit diese initialisert werden können. *hide, start, stop* und *accesscode* bieten dieselbe Funktionalität wie in pages.

Die Relation **pageletcontainer** dient nur zur Zuordnung mehrerer Pagelets zu einer Seite, dies könnte man auch ohne eine separate Relation erreichen, allerdings wäre dann das Ausblenden *(start, stop, hide)* und Wiederverwenden aller Pagelets einer Seite nicht möglich. *description* enthält kurze Informationen zum Inhalt des Containers.

4. Anwendung

4.1. Interfaces der Klassen

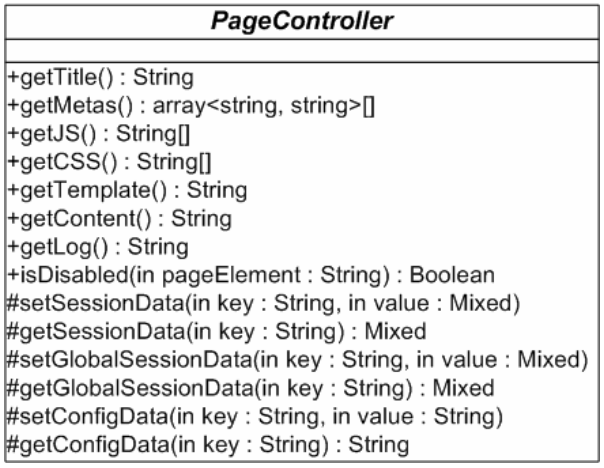
Für die Benutzung des Frameworks sind vor allem die Klassen Page Controller und Pagelet, sowie das Interface des Page-Objekts von Bedeutung. Diese werden im Folgenden vorgestellt. Weiter wird beschrieben, welche Informationen das Navigation Framework vom Benutzerobjekt

benötigt.

4.1.1. Page Controller

Der Page Controller wird als abstrakte Klasse bereitgestellt. Jeder Page Controller muss diese erweitern und seine spezifische Funktionalität einfügen. Dies ist notwendig, da die abstrakte Klasse bereits Basisfunktionalitäten wie das Rufen des Session- oder Configuration Managers implementiert hat, die auf jeden Fall zur Verfügung stehen müssen.

Implementiert ein Page Controller diese abstrakte Klasse nicht, so wird er aus Sicherheitsgründen nicht instanziiert und ausgeführt.



Die Methode `getTitle()` ist dafür vorgesehen, dass der Page Controller einen Seitentitel angeben kann. Dieser wird innerhalb des Title-Tags der Ausgabe eingefügt. Die Methode ist zudem als abstrakt gekennzeichnet, das heißt, ein Page Controller muss diese zwangsweise implementieren.

Will ein Page Controller Metaangaben zu der Seite ausgeben, so kann er dies mittels der Funktion `getMetas()` machen. In der abstrakten Klasse gibt sie ein leeres Array zurück.

Das Rückgabeformat der Funktion muss wie folgt aussehen:

```
array(
  0 => Array(
    [name       => "value of name",]
    [scheme     => "value of scheme",]
    [http-equiv => "value of http-equiv",]
    content     => "value of content"),
  1 => Array(
    [name       => "value of name",]
    [scheme     => "value of scheme",]
    [http-equiv => "value of http-equiv",]
    content     => "value of content"));
```

Es handelt sich dabei also um ein numerisches Array, dessen Elemente wiederum assoziative Arrays sind. Die eckigen Klammern um einige Elemente sollen bedeuten, dass diese Elemente optional sind. Je nachdem, welche Art von Meta-Tag ausgegeben werden soll, können diese gesetzt werden. Nur das `content`-Element (der eigentliche, sichtbare Inhalt) ist zwingend notwendig.

Die Anzahl der Meta-Tags ist nicht begrenzt, ein Page Controller sollte dennoch nur sinnvolle Angaben machen und die Ausgabe nicht unnötig vergrößern.

Mit den Funktionen `getJS()` beziehungsweise `getCSS()` kann der Page Controller mitteilen, welche JavaScript- oder CSS-Dateien er eingebunden haben möchte, um seinen Content-Rückgabestring formatieren zu lassen. Die Rückgabe muss jeweils ein Array mit den Dateinamen sein.

In der abstrakten Klasse geben diese Methoden beide ein leeres Array zurück.

Ein Page Controller kann in machen Situationen wünschen, dass ein anderes Template als das Standard-Template für die Gesamtdarstellung verwendet wird. Der Request Manager kann dies über die Funktion `getTemplate()` erfragen. In der abstrakten Klasse gibt diese Funktion den Pfad und Namen des Standard-Templates zurück; sie muss also gegebenenfalls überschrieben werden.

Soll auf einer Seite beispielsweise AJAX verwendet und dabei nicht auf die Features das Navigation Frameworks (User-Authentifizierung, Session- und Konfigurationsmanagement) verzichtet werden, so muss ein anderes Template verwendet werden, welches Inhalte mit XML formatiert ausgibt.

Die Funktion `getContent()` ist diejenige, die den eigentlichen Inhalt der Seite zurückgibt. Es handelt sich dabei um einen einzigen String, welcher an entsprechender Stelle im Template eingesetzt wird.

Dieser wird dabei nicht weiter untersucht. Der Page Controller ist also dafür zuständig, dass die verwendeten CSS-Definitionen und JavaScript-Funktionen mittels der dafür vorgesehenen Methoden eingebunden werden. Weiter muss der Page Controller selbst darauf achten, dass er valides XHTML erzeugt.

`getContent()` ist in der Basisklasse als abstrakt gekennzeichnet, sie muss also zwingend implementiert werden.

Über die Funktion `getLog()` kann ein Page Controller eine Nachricht angeben, welche in eine Datei oder Datenbank geloggt werden soll. Das Loggen übernimmt dabei der Statistics Manager.

In der Standardimplementierung gibt die Funktion einen leeren String zurück, erzeugt also keine Logging-Ausgabe.

In manchen Fällen ist es denkbar, dass ein Page Controller einzelne Bereiche der Seite ausblenden lassen möchte um mehr Platz für seinen Inhalt zu schaffen. Dies kann im Prinzip auch durch die Verwendung eines alternativen Templates geschehen, doch aus praktischen Gründen (nur einmalige Implementierung) kann der Page Controller dies auch über die Funktion `isDisabled($pageElement)` mitteilen, wobei `$pageElement` ein String ist. Der Rückgabewert ist *false*, falls das Element dargestellt werden soll, andernfalls *true*. Die Elementbezeichnungen können in Abschnitt 3.3.1 nachgelesen werden.

Das Request Manager fragt den Wert nun für unterschiedliche Seitenelemente ab. Zurzeit sind Angaben für die Elemente *header*, *footer*, *navigation* und *pagelets* möglich.

In der Standardimplementierung gibt diese Funktion immer *false* zurück, es werden folglich alle Seitenelemente dargestellt.

Die bisher vorgestellten Methoden sind alle als *public* gekennzeichnet. Dies ist notwendig, da das Navigation Framework all diese Funktionen aufrufen muss um die Informationen auf den Seiten einzubinden.

Die verbleibenden sechs Methoden `setSessionData()`, `getSessionData()`, setGlobalSessionData(), getGlobalSessionData(), `addConfigData()` und `getConfigData()` sind sowohl *protected* als auch *final*. Dies ist sinnvoll, denn nur von der Basisklasse abgeleitete (also eigene) Page Controller müssen auf diese Methoden zugreifen. Die Implementierung dieser Funktionen in der Basisklasse fügt zu den übergebenen Parametern den Namen der Klasse hinzu, mit dessen Hilfe die Parameter in unabhängigen Namensräumen vom Session oder Configuration Manager gespeichert werden können.

Als *final* wurden diese Funktionen gekennzeichnet, da ihre komplette Implementierung in der Basisklasse vorliegt und nicht überschrieben werden darf. Dieses Vorgehen hat, wie bereits erwähnt, zweierlei Vorteile: Zum einen muss sich der Page Controller nicht selbst um die Kommunikation mit den Managern kümmern. Zum anderen wird er auch von der Aufgabe befreit, sich um die Namensräume zu kümmern.

Eine direkte Kommunikation mit dem `$_SESSION`-Array kann zwar so nicht verhindert werden, jedoch ist davon strengstens abzuraten. Die Eindeutigkeit des gewählten Bezeichners ist so nicht gegeben und Daten anderer Page Controller könnten überschrieben werden.

Die Funktion `setSessionData()` erwartet zwei Parameter: Mit dem ersten String-Parameter wird ein Schlüssel gekennzeichnet, unter welchem der im zweiten Parameter übergebene Wert identifiziert werden soll. Der Wert kann dabei sowohl ein String, als auch ein Array, als auch ein Mixed sein.

Mit der Funktion `getSessionData()` wird ein gespeicherter Wert aus der Session gelesen und zurückgegeben. Als Parameter wird der Funktion dabei der bei `setSessionData()` verwendete Schlüssel übergeben.

Es kann Fälle geben, in welchen ein Page Controller Daten in einer Session speichert, die für einen anderen Page Controller (beispielsweise auf einer Folgeseite) bestimmt sind. Mittels der bisher vorgestellten Funktionen ist dies nicht möglich, da jeder Page Controller seinen eigenen Bereich (Namensraum) in der Session erhält.

Um die Kommunikation dennoch zu ermöglichen, gibt es die Funktionen `setGlobalSessionData()` und `getGlobalSessionData()`. Sie funktionieren völlig analog zu `setSessionData()` bzw. `getSessionData()`. Das Navigation Framework stellt hierbei jedoch nicht sicher, dass Werte anderer Page Controller überschrieben werden. Diese Methode sollte daher nur im geschilderten Fall verwendet werden.

Die Funktionen `setConfigData()` und `getConfigData()` arbeiten analog zu `setSessionData()` und `getSessionData()`. Eine Einschränkung gibt es jedoch: Da die Konfigura-

tionen im Gegensatz zur Session in der Datenbank abgelegt werden sollen, können sie nur als Strings gespeichert werden; um die Serialisierung etwaiger Arrays muss sich der Page Controller selbst kümmern.

4.1.2. Pagelet

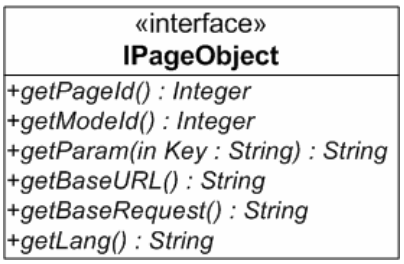
Wie auch der Page Controller wird das Pagelet als abstrakte Basisklasse bereitgestellt, die erweitert werden muss. Sie ist jedoch weitaus simpler: Nur eine abstrakte Funktion `getContent()` ist enthalten, welche einen String zurückgeben muss.

Dieser String wird unverändert an die entsprechende Stelle im Seitenlayout eingefügt. Folglich müssen alle Formatierungen innerhalb dieses Strings getätigt werden, da ein Einbinden einer externen CSS-Datei für ein Pagelet nicht möglich ist.

Wie auch der Page Controller trägt das Pagelet Sorge dafür, dass der erzeugte String valides XHTML ist.

4.1.3. Page Object

Das Page-Objekt stellt Informationen über die aktuell angezeigte Seite zur Verfügung. Dazu zählen beispielsweise die *mode_id* oder die *page_id*. Das Interface dieses Objekts sieht wie folgt aus:



Die Methoden `getPageId()` bzw. `getModeId()` geben die ID der aktuellen Seite bzw. des aktuellen Modus' zurück. Es ist aus folgendem Grund sinnvoll, diese Werte gesondert, also nicht über die Request-Parameter-Liste, bereitzustellen: Im Falle eines Zugriffsfehlers aufgrund fehlender Parameter kann eine andere Seite gewählt werden, die stattdessen dargestellt wird. In den vom Page-Objekt bereitgestellten Funktionen kann dies bequem durch einen anderen Rückgabewert berücksichtigt werden.

Die Funktion `getParam()` gibt das gewünschte Element aus dem $_REQUEST-Array zurück. Ist es nicht vorhanden, so wird null zurückgegeben. Über diesen Weg sind keine Informationen abrufbar, die ausschließlich das Navigtion Framework betreffen (beispielsweise die *page_id*).

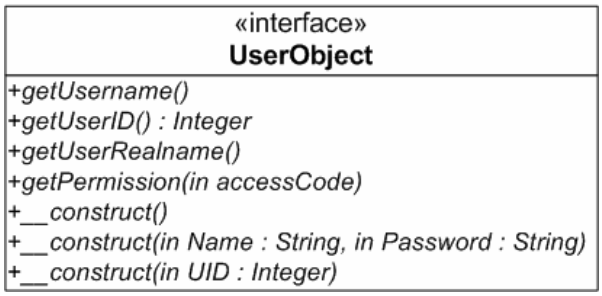
Der Zugriff auf $_REQUEST oder auch $_GET bzw. $_POST kann nicht direkt verhindert werden. Jedoch ist jeder Page Controller daran gehalten von der hier beschriebenen Funktion Gebrauch zu machen, da es aus Sicherheitsgründen denkbar ist, jene Variablen vor der Instanziierung des Page Controllers zu löschen.

Über die Funktionen `getBaseURL()` bzw. `getBaseRequest()` können Informationen über den aktuellen URL erfragt werden. `getBaseURL()` enthält dabei den kompletten Pfad der Ressource (i.A. index.html), `getBaseRequest()` hat zusätzlich einen Query-String angehängt, in welchem die *page_id* und *mode_id* der aktuellen Seite bereits notiert sind.

Es ist denkbar, dass ein Benutzer für die Dauer einer Sitzung die gewünschte Sprache der Seite ändern möchte. Diese Sprache kann mittels der Funktion *getLang()* abgerufen werden, was besonders für den Internationalization Manager interessant sein dürfte, um die Sprache zu bestimmen, in welche Textfragmente übersetzt werden sollen.

4.1.4. User-Objekt

Das nun folgende Interface ist ein Required Interface, es stellt also nur die Funktionen dar, die das Navigation Framework vom Benutzerobjekt mindestens erwartet.



Über die Funktionen `getUsername()` bzw. `getUserID()` soll der Loginname bzw. die User-ID abrufbar sein. Der Name kann für viele Anwendungsfälle wichtig sein, beispielsweise für eine Anzeige im Chat. Die User-ID kann unter anderem dafür verwendet werden, ein Benutzerobjekt anhand jener ID zu erzeugen. Dies wird

gleich näher erläutert.

Der echte Name (Realname) des Benutzers soll über eine Funktion `getUserRealName()` abrufbar sein. Dieser Name kann für personalisierte Seiten verwendet werden, beispielsweise für eine Begrüßung nach der Anmeldung.

Die Funktion `getPermission()` soll eine Überprüfung eines Access-Codes ermöglichen. Wie bereits erläutert, sind verschiedene Elemente nur mit einer hinreichenden Berechtigung zugänglich, welche durch diesen Code ausgedrückt wird. Als Parameter wird der Funktion der Code übergeben, als Rückgabe ein boolescher Wert erwartet: *true* falls der Zugriff gestattet und *false* falls er verboten ist. Bisher ist ein Zahlenwert vorgesehen, sollte die Benutzergruppe einen anderen Typ für sinnvoller erachten, so kann auch dieser verwendet werden.

Erstellt werden kann ein Benutzerobjekt mittels drei verschiedener Konstruktoren. Der Standardkonstruktor ohne Parameter soll einen Benutzer mit Gastrechten erzeugen.
Wird dem Konstruktor eine Benutzername/Passwort-Kombination übergeben, so soll im Falle einer richtigen Kombination der jeweilige Benutzer, andernfalls ein Gastbenutzer eingeloggt werden.
Der dritte Konstruktor erwartet lediglich eine User-ID. Dieser Konstruktor wird immer dann aufgerufen, wenn sich ein Benutzer zuvor erfolgreich angemeldet hatte und seine ID nun bekannt ist. Sie kann in der Session gespeichert sein. Dieses Vorgehen wurde einer Speicherung von Benutzernamen und Passwort in der Session aus Sicherheitsgründen vorgezogen.

4.2. Ausnahmebehandlung

Während ein Page Controller den Inhalt der Seite generiert, kann es passieren, dass bei ihm ein Fehler auftritt. Dies kann zum Beispiel trotz der grundsätzlichen Berechtigung zur Anzeige der konkreten Seite eine nicht ausreichende Berechtigung für bestimmte Inhaltselemente sein. In diesem Fall kann in der Methode `getContent()` eine Ausnahme ausgelöst werden.

Bisher sind dem Navigation Framework drei Ausnahmeklassen bekannt, es können jedoch weitere definiert werden. Alle Ausnahmen sind von der vordefinierten Klasse `Exception` abgeleitet bzw. müssen von dieser abgeleitet werden.

Die bekannten Ausnahmeklassen sind:

1. `AccessException`
   Diese Ausnahme soll immer dann ausgelöst werden, wenn ein Rechteproblem während der Verarbeitung des Requests auf Seiten des Page Controllers aufgetreten ist. Der Request Manager kann in diesem Fall ein Login-Formular oder eine Fehlermeldung anzeigen.
2. `ADODBException`
   Das Navigation Framework stellt eine Standard-Datenbank zur Verfügung, welche von allen Komponenten verwendet werden kann. Sollte bei der Verwendung ein Fehler auftreten, so kann dieser mittels der Exception an den Request Manager mitgeteilt werden. Es kann daraufhin beispielsweise eine spezielle Logmeldung generiert oder eine E-Mail an den Administrator gesendet werden.
3. `SmartyExeption`
   Ebenfalls wird ein globales Smarty-Objekt zur Verfügung gestellt, welches von Page Controllern verwendet werden kann. Analog zu einem Datenbankfehler kann mittels dieser Meldung gesagt werden, dass es bei der Verwendung Probleme gab, welche wiederum geloggt werden können.

Die bereits definierten Ausnahmen überschreiben die magischen Funktionen `__construct()` und `__toString()` der Elternklasse um darin eigene Fehlermeldungen zu ermöglichen. Die Funktionen sehen wie folgt aus:

```
public function __construct($message="") {
  parent::__construct("Access denied: " .
$message, 1);
}

public function __toString() {
  return get_class($this) . ": [{$this->
code}]: {$this->message}\n";
}
```

Wie man erkennt, kann im Konstruktor eine optionale, detailliertere Fehlermeldung angegeben werden. Mit dieser wird anschließend der Konstruktor der Elternklasse aufgerufen.

Die Ausgabe der Funktion `__toString()` erzeugt einen String, welcher sowohl den Namen der speziellen Ausnahmeklasse, den Fehlercode sowie die bei der Konstruktion angegebenen Fehlermeldung beinhaltet.

Die Fehlernummern der definierten Klassen

sind entsprechend der obigen Nummerierung gegeben. Definiert ein Page Controller eine eigene Fehlermeldung, so muss er für die Fehlernummer eine Nummer größer als 100 wählen. Die Definition von Ausnahmen mit kleineren Nummern unterliegt dem Navigation Framework; sie dürfen also nur im Zusammenhang mit den bereitgestellten Ausnahmen verwendet werden.

Wie bereits erwähnt, erfolgt die Abfrage der Ausnahmen beim Aufrufen der `getContent()`-Funktion des Page Controllers. Ein `try-catch`-Block wird dabei um den Funktionsaufruf gelegt um die Exceptions abzufangen:

```
try {
  $pagecontroller->getContent();
  // getCSS(), getJS(), ...
}
catch(AccessException ae) {
  // Behandlung der AccessException
}
catch(SmartyException se) {
  // Behandlung der SmartyException
}
catch(ADODBException adoe) {
  // Behandlung der ADODBException
}
catch(Exception e) {
  // Behandlung der allgem. Exception
}
```

4.3. Validation Pattern

Oft kann die Ausgabe von Seiten nicht immer nur abhängig von *page_id* und *mode_id* ermittelt werden, da zusätzliche Informationen nötig sind. Will man zum Beispiel den Stream einer bestimmten Vorlesung sehen, so ist es nötig, zusätzlich noch die Einordnung des Streams in Topic, Series, LectureGroups und Lecture mit anzugeben.

Diese Informationen werden per GET mit dem HTTP-Request übertragen. Es kann allerdings vorkommen, dass die Requestvariablen nicht gültig oder unvollständig sind, weil der Benutzer sie eventuell manuell in die Adresszeile seines Browsers eingegeben hat. In diesem Fall kann keine korrekte Ausgabe erfolgen, und der Benutzer muss darüber informiert werden.

Um nun aber die Serverlast und die Wartezeit des Benutzers zu verringern, wird die Möglichkeit geschaffen die GET-Parameter schon vor dem Laden des Page Controllers zu überprüfen. Das geschieht mit Hilfe des für eine *page_id* und *mode_id* registrierten Validation Patterns. Das Validation Pattern ist dabei ein String, genauer: eine Kette regulärer Ausdrücke, der angibt, welche Parameter zwingend erforderlich sind und welchen syntaktischen Bedingungen diese genügen müssen. Dieser Validation Pattern-String wird bei der Verknüpfung eines bestimmten Page Controllers mit einer *page_id* und *mode_id* angegeben.

4.3.1. Syntax

Das Validation Pattern muss eine bestimmte Syntax erfüllen, damit es überhaupt ausgewertet werden kann. Ist das Pattern fehlerhaft, so können auch gültige Parameter nicht validiert werden und der Page Controller wird im Produktivbetrieb nicht einsetzbar sein.

Prinzipiell setzt sich das Pattern aus Schlüssel-Wert-Paaren zusammen, die durch das *kaufmännische Und* („&“) getrennt sind. Der Schlüssel bezeichnet einen Parameter, der mindestens gesetzt sein muss. Der Wert hingegen ist ein gültiger regulärer Ausdruck[4].

Beispiel:

```
key1=/expression1/&key2=/expression2/&...
```

Für den regulären Ausdruck gelten folgende Regeln:

- Der Wert, also der reguläre Ausdruck, kann durchaus leer sein, dadurch wird der entsprechende Parameter nicht auf Syntax geprüft, muss aber zumindest gesetzt sein. Es kann trotzdem ein Gleichheitszeichen („=“) auf den Schlüssel folgen, muss aber nicht.
- Der reguläre Ausdruck muss gültige Delimiter enthalten, damit er ausgewertet werden kann.
- Reguläre Ausdrücke können neben dem eigentlichen Pattern noch Modifieren[5] enthalten, mit dem die Bedingungen noch weiter spezifiziert werden. Beispielsweise ist `preg_match()` standardmäßig *casesensitiv*, ein „i“ nach dem Schließen des Delimiters sollte verwendet werden, um die Methode *caseinsensitiv* zu machen.

Das folgende Beispiel zeigt einen syntaktisch korrekten Validation Pattern und das dazugehörige, validierbare Array.

[4] Mehr zu regulären Ausdrücken unter : http://www.php.net/manual/de/reference.pcre.pattern.syntax.php

[5] Mehr zu Modifiern: http://www.php.net/manual/de/reference.pcre.pattern.modifiers.php

```
apple=/[a-z]*/i&banana&cherry=/[d-r]{3}/

array('apple' => 'green',
      'banana' => 'blue',
      'cherry' => 'red')
```

5. Administration

Unter Administration wird in Zusammenhang mit diesem Dokument das Hinzufügen, Verändern und Löschen jeweils von Seiten, Modi, Page Controllern, Pagelets usw. verstanden. Dazu steht ein grafisches Interface zur Verfügung, das ebenfalls in die Seite integriert ist (*page_id*=0). Dort ist der aktuelle Seitengraf dargestellt, zu dessen Blättern (Seiten) Informationen abgefragt und verändert werden können.

Will man beispielsweise einen Modus zu einer Seite dazufügen, wählt man die entsprechende Seite aus und klickt auf „Modus hinzufügen". Man kann bereits definierte Modi aus der sich öffnenden Auswahlliste aussuchen oder einen neuen Modus erstellen.

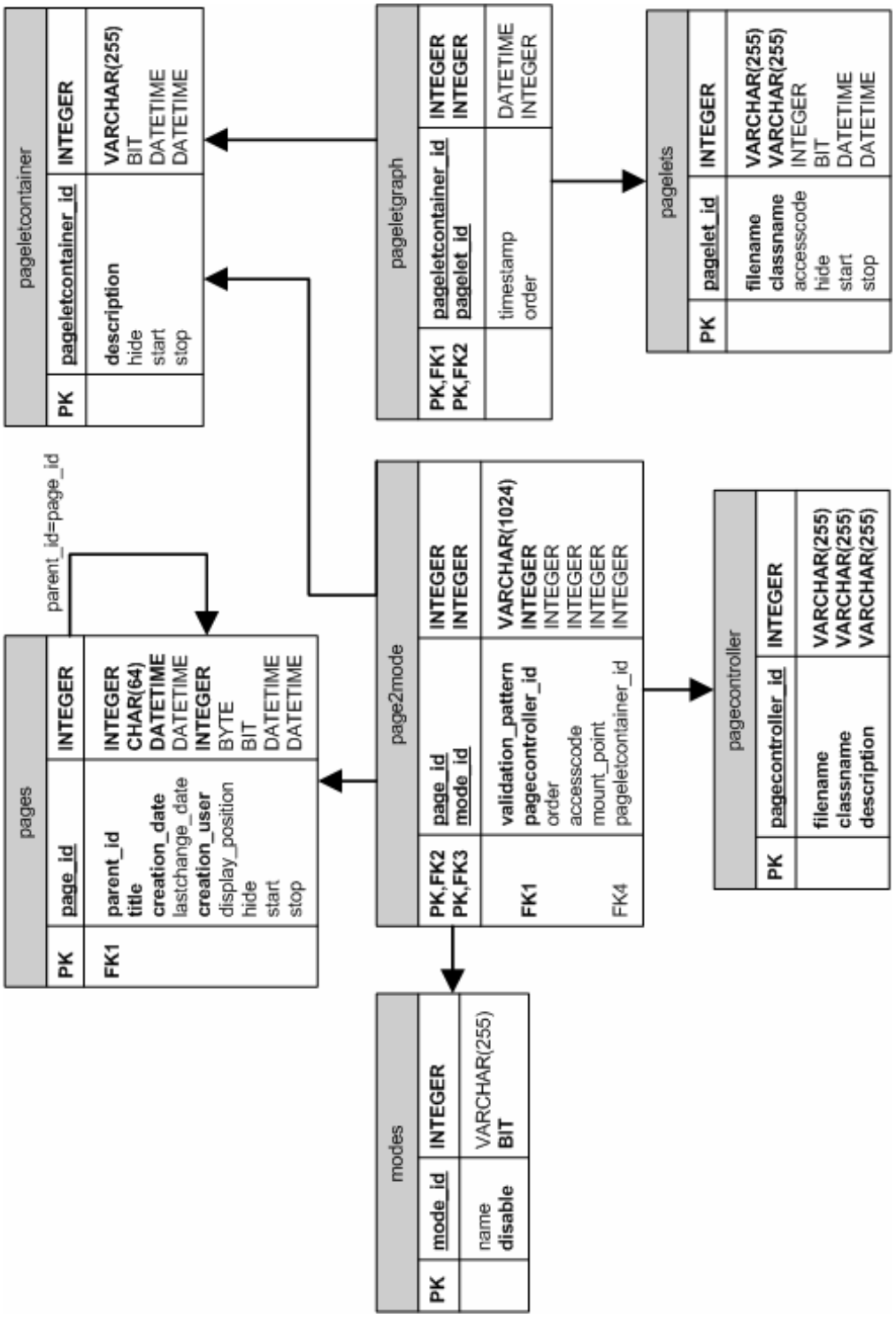
Das Hinzufügen von Seiten erfolgt analog. Hier wird allerdings zu einer gewählten *page_id* eine Unterseite angelegt.

Will man einen Page Controller hinzufügen, klickt man auf das entsprechende Icon und gibt Dateinamen, Pfad und eine Beschreibung an.

Grundsätzlich gilt: jegliches Hinzufügen/Ändern/Verknüpfen von Features fragt alle in den beteiligten Tabellen in der Datenbank enthaltenen Attribute ab, die über ein passendes Formular eingegeben werden können. Abkürzend und vereinfachend wirkt lediglich, daß dies direkt an der grafischen Repräsentation im Baum geschehen kann. (siehe „Modus hinzufügen") Das bloße „Tabellenändern" sollte allerdings nicht unterschätzt werden, weil beispielsweise der komplette Graf verändert werden kann, nur dadurch, dass die *parent_id* oder ein Mount Point geändert wird.

Ein weiterer Anwendungsfall zum Aufsuchen der Administrationsseite ist das Auslesen von passenden Seiten-/Modus-IDs, um daraus eine Verknüpfung für die aktuell bearbeitete Seite zu erstellen. Wahrscheinlich wird diese rein informative Funktionalität aber auf einen anderen Modus der Administrationsseite gelegt. Denkbar ist auch, daß dies lediglich als kleines „Tool" verstanden wird, das sich später in einem Popup-Fenster öffnen lassen kann.

Anhang


pageletcontainer
PK
pageletcontainer_id
INTEGER
description
hide
start
stop
VARCHAR(255)
BIT
DATETIME
DATETIME
pageletgraph
PK,FK1
PK,FK2
pageletcontainer_id
pagelet_id
INTEGER
INTEGER
timestamp
order
DATETIME
INTEGER
pagelets
PK
pagelet_id
INTEGER
filename
classname
accesscode
hide
start
stop
VARCHAR(255)
VARCHAR(255)
INTEGER
BIT
DATETIME
DATETIME
parent_id=page_id
pages
PK
page_id
INTEGER
FK1
parent_id
title
creation_date
lastchange_date
creation_user
display_position
hide
start
stop
INTEGER
CHAR(64)
DATETIME
DATETIME
INTEGER
BYTE
BIT
DATETIME
DATETIME
page2mode
PK,FK2
PK,FK3
page_id
mode_id
INTEGER
INTEGER
FK1
FK4
validation_pattern
pagecontroller_id
order
accesscode
mount_point
pageletcontainer_id
VARCHAR(1024)
INTEGER
INTEGER
INTEGER
INTEGER
INTEGER
pagecontroller
PK
pagecontroller_id
INTEGER
filename
classname
description
VARCHAR(255)
VARCHAR(255)
VARCHAR(255)
modes
PK
mode_id
INTEGER
name
disable
VARCHAR(255)
BIT